

UNIVERSIDADE SENAI CIMATEC
COMPUTER ENGINEERING

Orlando Mota Pires

Hybrid Quantum-Classical Graph Partitioning Algorithm for Domain Decomposition

SALVADOR
2026



Orlando Mota Pires

HYBRID QUANTUM-CLASSICAL GRAPH PARTITIONING ALGORITHM FOR DOMAIN DECOMPOSITION

Monograph presented to Examining Bench of Universidade Senai Cimatec, as part of the requirements for obtaining the title of Bachelor in Computer Engineering under orientation of MSc. Stefan Schröder and Dr. Anderson Buarque.

SALVADOR
2026

Gostaria de dedicar esse trabalho a minha mãe que sempre teve respostas para perguntas difíceis, ao meu pai que foi o maior incentivador a continuar a buscar e se manter curioso e irmão que sempre foi fonte de inspiração descontração e porto seguro. Aos amigos do peito, gostaria de agradecer por toda a colaboração, parceria e paciência.

ACKNOWLEDGMENTS

I would first like to thank my great professor Dr Marcio Renê Brandão Soussa for all the years of collaboration, guidance and learning. You were undoubtedly the best teacher I've ever had the pleasure of having throughout my journey.

I am immensely grateful to my supervisor MSc. Stefan Schröder for his collaboration during my time at Fraunhofer IPT, which was a period of much learning and re-sharpening of my understanding of myself, and also to my supervisor Dr Anderson Buarque for his help and guidance throughout the work process.

I would also like to mention all the friends I have made during this journey and say that you have made it lighter, more productive and that without you none of this would have been possible.

Finally, I would like to thank three colleagues in particular, Adrian Widmer, Antônio Horácio and Fernando Schettini for all their collaboration from the beginning of my degree with the creation of the Programming Club until the end of my journey.

It's a dangerous business, Frodo, going out your door. You step onto the road, and if you don't keep your feet, there's no knowing where you might be swept off to.
– Baggins, Bilbo

RESUMO

Simulações em grande escala com elementos finitos costumam gerar sistemas lineares esparsos, cuja solução é o que mais pesa no custo computacional da análise de engenharia. O desempenho de solucionadores diretos e iterativos pré-condicionados é fortemente influenciado pela estrutura de esparsidade das matrizes de rigidez e de massa. Este trabalho investiga um fluxo híbrido quântico-clássico de partição de grafos para decomposição de domínio, com o objetivo de reordenar matrizes de elementos finitos em estruturas mais favoráveis à construção e aplicação de pré-condicionadores.

O fluxo proposto converte a matriz de rigidez em um grafo ponderado, reduz esse grafo por meio de coarsening multinível, particiona o grafo reduzido usando estratégias inspiradas na computação clássica e quântica, projeta a partição de volta no grafo original e aplica a ordenação resultante tanto à matriz de rigidez K quanto à matriz de massa M. Os backends de particionamento implementados incluem METIS, recozimento quântico simulado e em hardware, e o Algoritmo Quântico de Otimização Aproximada (QAOA) emulado. As matrizes finais são avaliadas por métricas de largura de banda, como a largura de banda média e a fração de elementos não nulos em faixas percentuais próximas à diagonal, além do número de iterações no pior caso. Essas métricas são interpretadas como indicadores estruturais para o pré-condicionamento, e não como medidas diretas do tempo de resolução.

Dois demonstradores derivados do ANSYS foram analisados: uma geometria de lâmina sem suporte na base e uma geometria de lâmina com uma base de 5 mm. Os resultados mostram que o método preserva permutações válidas da matriz e pode alterar significativamente o perfil de esparsidade. No primeiro demonstrador, a melhor execução bem-sucedida reduziu o número máximo de iterações em aproximadamente 63,8% para K e M. No segundo demonstrador, a topologia mais densa do grafo limitou a redução na largura de banda máxima, alcançando uma redução de 1,2% no número máximo de iterações, mas vários métodos ainda melhoraram a concentração local próxima à diagonal. O estudo confirma que a geometria, a escala de coarsening e o backend de particionamento interagem fortemente, e que nenhuma métrica estrutural isolada é suficiente para avaliar a qualidade da permutação. Conclui-se que o particionamento híbrido multinível de grafos é uma estratégia viável de pré-processamento para melhorar as condições estruturais na construção de pré-condicionadores, enquanto medições diretas do número de pré-condicionamento continuam sendo um importante trabalho futuro.

Palavras-chave: Finite Element Meshes , Domain Decomposition , Graph Partitioning , Quantum Annealing.

ABSTRACT

Large-scale finite element simulations often produce sparse linear systems whose solution dominates the computational cost of engineering analysis. The performance of preconditioned direct and iterative solvers is strongly influenced by the sparsity structure of the stiffness and mass matrices. This work investigates a hybrid quantum-classical graph partitioning flow for domain decomposition, with the aim of reordering finite element matrices into structures more favourable to the construction and application of preconditioners.

The proposed flow converts the stiffness matrix into a weighted graph, reduces this graph by means of multilevel coarsening, partitions the reduced graph using classical and quantum computing-inspired strategies, projects the partition back onto the original graph and applies the resulting ordering to both the stiffness matrix K and the mass matrix M. The partitioning backends implemented include METIS, simulated and hardware quantum annealing, and emulated Quantum Approximate Optimization Algorithm (QAOA). The final matrices are evaluated by bandwidth metrics, Avg. Bandwidth and fraction of non-null elements within percentage bands close to the diagonal and number of iterations in the worst case. These metrics are interpreted as structural indicators for preconditioning, and not as direct measures of solver time.

Two ANSYS-derived demonstrators were analysed: a blade geometry without base support and a blade geometry with a 5 mm base. The results show that the method preserves valid matrix permutations and can significantly alter the sparsity profile. In the first demonstrator, the successful best run reduced the maximum number of iterations by approximately 63.8% for K and M. In the second demonstrator, the denser topology of the graph limited the reduction in maximum bandwidth, achieving a 1.2% reduction in the maximum number of iterations, but several methods still improved local concentration near the diagonal. The study confirms that geometry, coarsening scale and partitioning backend interact strongly, and that no single structural metric is sufficient to assess permutation quality. It is concluded that hybrid multilevel graph partitioning is a viable preprocessing strategy to improve the structural conditions for building preconditioners, while direct measurements of the precondition number remain important future work.

Key-words: Finite Element Meshes , Domain Decomposition , Graph Partitioning , Quantum Annealing.

LIST OF FIGURES

Figure 1 – Physical displacement of a specific node in a specific direction. . . 24
Figure 2 – Comparison of original and permuted matrix K. At the left one can see the K original before applying the permutation and at the right the K matrix after applying the permutation. 25
Figure 3 – Bloch sphere representation of a single-qubit state. 31
Figure 4 – Circuit decomposition of the QFT 35
Figure 5 – Circuit overview for the QPE 36
Figure 6 – Example of the Multilevel Graph Bisection 42
Figure 7 – Quantum tunneling overview 49
Figure 8 – Circuit overview for the QAOA 52

Figure 9 – Domain decomposition architecture 61
Figure 10 – Multilevel Scheme Algorithms. Where HEM stands for Heavy Edge Matching and QAOA for Quantum Approximation Optimization Algorithm 62

Figure 11 – Example of graph coarsening 64
Figure 12 – Hierarchical tree structure of the sets 66

Figure 13 – Recursive k-way bisection 70
Figure 14 – Single bisection of coarsed graph G_2. 70
Figure 15 – Partitioning Strategies. Where HEM Alg. stands for Heavy Edge Matching Algortihm, SB Alg. for Spectral Bissection Algorithm, RKL Alg. for Recursive Kernighan-Lin Algorithm, QUBO for Quadratic Unconstrained Binary Optimization and QPU for Quantum Processing Unit 71
Figure 16 – Dwave-Leap Platform 73

Figure 17 – Example of graph uncoarsening with preserved partitions 80

Figure 18 – Visual representation of matrix permutation 85

Figure 19 – Demonstrator 01 on the left, and demonstrator 02 on the right . . 88
Figure 20 – Partition-colored K-graph views for Demonstrator 01 big coarsen size: initial graph, coarsened graph, and uncoarsened graph. 93
Figure 21 – Partition-colored K-graph views for Demonstrator 03 big coarsen size: initial graph, coarsened graph, and uncoarsened graph. 94
Figure 22 – Demonstrator 01 K-matrix METIS big coarsen size 94
Figure 23 – Demonstrator 01 K-matrix Quantum Annealing Hardware medium coarsen size 95

Figure 24 – K-only fraction-within-band comparison for Demonstrator 01: QA hardware medium versus METIS medium. 96
Figure 25 – Demonstrator 01 K-matrix Quantum Annealing Emulated small coarsen size 97
Figure 26 – K-only fraction-within-band comparison for Demonstrator 01: QA hardware small versus METIS small. 97
Figure 27 – Demonstrator 01 K-matrix QAOA Emulated small coarsen size . . 98
Figure 28 – K-only fraction-within-band comparison for Demonstrator 01: QAOA emulated small versus METIS small. 98
Figure 29 – Partition-colored K-graph views for Demonstrator 02: initial graph, coarsened graph, and uncoarsened graph. 101
Figure 30 – Demonstrator 02 K-matrix METIS big coarsen size 101
Figure 31 – Demonstrator 02 K-matrix Quantum Annealing Hardware small coarsen size 102
Figure 32 – K-only fraction-within-band comparison for Demonstrator 02: QA hardware small versus METIS small. 103
Figure 33 – Demonstrator 02 K-matrix Quantum Annealing Emulated medium coarsen size 103
Figure 34 – K-only fraction-within-band comparison for Demonstrator 02: QA emulated medium versus METIS medium. 104
Figure 35 – Demonstrator 02 K-matrix QAOA Emulated small coarsen size . . 105
Figure 36 – K-only fraction-within-band comparison for Demonstrator 02: QAOA emulated small versus METIS small. 105
Figure 37 – Best-method K original-versus-permuted comparisons by maximum-bandwidth criterion: Demonstrator 01 with QA emulated small on the left and Demonstrator 02 with QA emulated medium on the right. 109
Figure 38 – Demonstrator 01 K-matrix METIS medium coarsen size 119
Figure 39 – Demonstrator 01 K-matrix METIS small coarsen size 119
Figure 40 – Demonstrator 02 K-matrix METIS medium coarsen size 120
Figure 41 – Demonstrator 02 K-matrix METIS small coarsen size 120
Figure 42 – K-only fraction-within-band comparison for Demonstrator 01: QA hardware medium versus METIS medium. 121
Figure 43 – Demonstrator 01 K-matrix Quantum Annealing Hardware small coarsen size 121
Figure 44 – K-only fraction-within-band comparison for Demonstrator 01: QA hardware small versus METIS small. 122

Figure 45 – Demonstrator 02 K-matrix Quantum Annealing Hardware medium coarsen size 122
Figure 46 – K-only fraction-within-band comparison for Demonstrator 01: QA hardware medium versus METIS medium. 123
Figure 47 – Demonstrator 01 K-matrix QA Emulated big coarsen size 123
Figure 48 – K-only fraction-within-band comparison for Demonstrator 01: QA Emulated Big versus METIS Big. 124
Figure 49 – Demonstrator 01 K-matrix Quantum Annealing Emulated medium coarsen size 124
Figure 50 – K-only fraction-within-band comparison for Demonstrator 01: QA hardware medium versus METIS medium. 125
Figure 51 – Demonstrator 02 K-matrix Quantum Annealing Emulated big coarsen size 125
Figure 52 – K-only fraction-within-band comparison for Demonstrator 02: QA emulated big versus METIS big. 126
Figure 53 – Demonstrator 02 K-matrix Quantum Annealing Emulated small coarsen size 126
Figure 54 – K-only fraction-within-band comparison for Demonstrator 02: QA hardware small versus METIS small. 127

LIST OF TABLES

Table 2 – General simulation parameters 63
Table 3 – Output data 63
Table 4 – Coarsening Parameters 69
Table 5 – Common partitioning parameters. 75
Table 6 – Parameters specific to the recursive Quantum Annealing strategy. 76
Table 7 – Parameters specific to the recursive QAOA strategy. 77
Table 8 – Uncoarsening and refinement parameters. 83
Table 9 – Matrix profiling metrics computed for original and permuted matrices. 86
Table 10 – Overview of the demonstrators 89
Table 11 – Common simulation parameters used across the analyzed batch cases 90
Table 12 – Batch cases selected for both demonstrator geometries 91
Table 13 – Quantum backend and platform parameters used in the simulation campaign 92
Table 14 – K-matrix numerical summary for Demonstrator 01 99
Table 15 – M-matrix numerical summary for Demonstrator 01 99
Table 16 – K-matrix nB^2 metrics for Demonstrator 01 99
Table 17 – K-matrix numerical summary for Demonstrator 02 106
Table 18 – M-matrix numerical summary for Demonstrator 02 106
Table 19 – K-matrix metrics for Demonstrator 02 106

LIST OF ABBREVIATIONS

CG	Conjugate Gradient
CNOT	Controlled Not
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DD	Domain Decomposition
FEA	Finite Element Analysis
FEM	Finite Element Method
FETI	Finite Element Tearing and Interconnect
FM	Fiduccia-Mattheyses
FPGA	Field Programmable Gate Array
GMRES	Generalized Minimal Residual
GPU	Graphics Processing Unit
HEM	Heavy Edge Matching
HSS	Hybrid Solver Service
H	Hadamard
ICE	Integrated Control Electronics
ILU	Incomplete Lower-Upper Factorization
KL	Kernighan-Lin
KaHIP	Karlsruhe High Quality Partitioning
MHEM	Modified Heavy Edge Matching
NNZ	Number of Non-Zeros
NP	Non-deterministic Polynomial
QAOA	Quantum Approximate Optimization Algorithm
QA	Quantum Annealing
QFT	Quantum Fourier Transform
QGP	Quantum Graph Partitioning
QLM	Quantum Learning Machine
QPE	Quantum Phase Estimation
QPU	Quantum Processing Unit
QUBO	Quadratic Unconstrained Binary Optimization
SHEM	Sorted Heavy Edge Matching

LIST OF SYMBOLS

A	General matrix
A_p	Permuted matrix PAP^T
B	Bandwidth
$B(A)$	Maximum bandwidth
C	Damping matrix
$C(U)$	Controlled-unitary gate
$C_{\text{factorization}}$	Factorization complexity
D	Constitutive matrix
E	Young's modulus
E_ℓ	ℓ-th energy eigenvalue
$E_{\text{altered}}(s)$	Perturbed energy function on hardware
E_{ij}	Set of edges crossing between blocks V_i and V_j
F	Spring/nodal force scalar
F_1, F_2	Nodal forces in two-node spring element
$F_b(A)$	Fraction of nonzeros within diagonal band b
$G_0, \ldots, G_m$	Hierarchy of coarsened graphs
H	Mass-normalised stiffness matrix
H_C	Cost Hamiltonian
H_D	Driver/initial Hamiltonian
H_M	Mixing Hamiltonian
H_P	Problem/phase Hamiltonian
H_β	Mixing Hamiltonian
H_f	Cost-function-based Hamiltonian
H_{GP}	Graph-partitioning Ising Hamiltonian
I	Identity matrix/identity operator
J_{ij}	Quadratic coupling strength
K	Global stiffness matrix
K', M'	Permuted stiffness and mass matrices
K^e	Element stiffness matrix
$K_m(A, r_0)$	Krylov subspace of dimension m
L	Number of coarsening levels

$L_{\max}$	Maximum allowed block weight $(1+\epsilon)\,c(V)/k$
M	Global mass matrix
$M^{-1/2}$	Inverse square root of mass matrix
M^{-1}	Inverse preconditioner
$M^{1/2}$	Symmetric square root of mass matrix
N	Number of degrees of freedom
$N_b(A)$	Count of nonzeros within band b
P	Permutation matrix
$P(\Delta E)$	Boltzmann acceptance probability
Q	QUBO coefficient matrix
R_k	Controlled phase-rotation gate
$R_x(\theta)$	Single-qubit rotation about the x-axis
$R_y(\theta)$	Single-qubit rotation about the y-axis
$R_z(\theta)$	Single-qubit rotation about the z-axis
S	Phase gate
$S_\lambda(\omega)$	Power spectral density of noise source λ
T	Total annealing runtime
$T(n,p)$	Per-iteration time model
T_1	Energy-relaxation time
T_{sim}	Total simulated Hamiltonian evolution time
U	Unitary/evolution operator
U_H	Block-encoding unitary
$U_M(\beta)$	QAOA mixer unitary $e^{-i\beta H_M}$
$U_P(\gamma)$	QAOA phase unitary $e^{-i\gamma H_P}$
V	Set of graph vertices
V_i	i-th partition block
W	Orthogonal eigenvector matrix
W_{ij}	Adjacency weight between nodes i and j
W_{uj}	Element of eigenvector matrix
X	Pauli-X gate
X_j	Pauli-X operator acting on qubit j
Y	Pauli-Y gate
Z	Pauli-Z gate

Z_i	Pauli-Z operator acting on qubit i
ΔE	Energy difference/change
$\Delta\lambda$	Eigenvalue estimation error
$\Delta\phi$	Phase estimation error
Λ	Diagonal spectral matrix containing eigenvalues
Ω	Physical domain
Ω_i	i-th subdomain
α	Complex probability amplitude
$\bar{B}(A)$	Average bandwidth
β	Complex probability amplitude
$\beta_1, \ldots, \beta_p$	QAOA mixer angle parameters
$\delta J_{i,j}$	Programmable coupling error on hardware
δh_i	Programmable bias error on hardware
δ	Fractional phase approximation error
ϵ	Balance imbalance tolerance parameter
ϵ_{sim}	Hamiltonian simulation error tolerance
η	Penalty/regularisation parameter
γ	Global phase factor
$\gamma_1, \ldots, \gamma_p$	QAOA phase angle parameters
$\hat{D}_\lambda$	Transition dipole moment for noise source λ
$\hbar$	Reduced Planck constant
κ	Condition number
$\kappa(A)$	Condition number of matrix A
$\kappa(M^{-1}A)$	Condition number of preconditioned operator
λ_j	j-th eigenvalue
CNOT	Controlled-NOT gate
QFT	Quantum Fourier Transform operator
$\text{cut}(\Pi)$	Total weighted edge cut of partition Π
$\text{cut}(s)$	Edge cut expressed in spin variables
$\text{cut}(x)$	Edge cut expressed in binary variables
$\text{frac}_{\rho\%}(A)$	Locality fraction at percentage band ρ
$\text{nnz}(A)$	Number of nonzero entries
ω	Angular frequency

ω_j	j-th natural frequency
ϕ	Bloch sphere azimuthal angle
π	Partition assignment vector $\pi : V \to \{1, \ldots, k\}$
ψ	Quantum state
ρ	Percentage value for band definition
σ	Stress
σ_i^x	Pauli-X spin operator at site i
σ_i^z	Pauli-Z spin operator at site i
τ	Balance tolerance perturbation
θ	Bloch sphere polar angle
φ	Mode shape vector
a	Arithmetic/memory cost parameter
b	Band radius
$b_\rho(A)$	Percentage-defined band radius
c	Node weight function $c : V \to \mathbb{R}_{>0}$
$c(V)$	Total graph weight
$c(V_i)$	Weight of block V_i
c^T	Row vector of node weights
c_i	Weight of vertex v_i
d	Memory-access cost parameter
e	Integer tolerance parameter
f	Global nodal force vector
$f(x)$	QUBO objective function value
$f(y)$	Cost-function evaluation at bitstring y
f^e	Element-level nodal force vector
$g_{\min}$	Minimum spectral gap between ground and first excited level
h_i	Linear bias coefficient
i	Row/node index
j	Column/node index
k	Spring constant
k_B	Boltzmann constant
m	QPE measurement outcome integer
m_u	Lumped mass at node u

n	Matrix/graph dimension
p	Semi-bandwidth; number of processors
q	QUBO linear coefficient vector
r	Per-word communication transfer time
r_0	Initial residual vector
s	Message startup latency
s_i	Spin variable, $s_i \in \{+1, -1\}$
t	Continuous evolution time
u	Global nodal displacement vector
$u(t)$	Time-dependent displacement vector
u^e	Element-level nodal displacement vector
u_1, u_2	Nodal displacements in two-node element
u_u	Displacement of node u in a specific direction
v	Mass-normalised eigenvector
$w(u, v)$	Edge weight between vertices u and v
x	Binary variable vector
x_i	Binary variable, $x_i \in \{0, 1\}$
$x_{i,r}$	One-hot binary variable
y	Modal coordinate vector
y_j	j-th modal coordinate
z_i	Spin variable in $\{+1, -1\}$

TABLE OF CONTENTS

1	INTRODUCTION	15
1.1	Motivation	15
1.2	Justification	15
1.3	Objectives	16
1.4	Work Structure	17
2	THEORETICAL BACKGROUND	19
2.1	Finite Element Analysis	19
2.1.1	Motion Equation Formulation	20
2.1.2	The Eigenvalue Problem Formulation	22
2.2	Domain Decomposition	23
2.2.1	Divide and Conquer	23
2.2.2	Parallelism Costs	25
2.2.3	Preconditioning and Solver Efficiency	26
2.3	Quantum Computing Fundamentals	30
2.3.1	Building Blocks	30
2.3.2	Quantum Circuits	32
2.3.3	Quantum Phase Estimation	33
2.4	Graph Partitioning	39
2.4.1	From mesh to graph	39
2.4.2	Graph Partitioning Problem Definition	39
2.4.3	The Multilevel Strategy	41
2.5	Quantum Optimization Algorithms	43
2.5.1	The Ising Model	44
2.5.2	Quadratic Unconstrained Binary Optimization	45
2.5.3	The Adiabatic Theorem	47
2.5.4	Quantum annealing	48
2.5.5	Quantum Approximate Optimization Algorithm	51
2.6	Quantum Annealing Hardware	53
2.6.1	Physical Annealers	53
2.6.2	Emulated Annealers	54
2.6.3	Hybrid Solutions	55
3	STATE OF THE ART	57
3.1	Domain Decomposition and Sparse Solvers	57
3.2	Graph Partitioning for Domain Decomposition	58
3.3	Matrix Reordering and Preconditioning	58

3.4 Quantum Optimization for Graph Partitioning . . . 59
3.5 Platform Limitations and Research Gap . . . 60

4 PIPELINE ARCHITECTURE . . . 61
4.1 Domain decomposition pipeline . . . 61
4.2 Input Data and Initialization . . . 62
4.3 Output Data . . . 63

5 COARSENING . . . 64
5.1 Coarsening process . . . 64
5.1.1 Random Heavy Edge Matching . . . 64
5.1.2 Sorted Heavy Edge Matching . . . 65
5.1.3 Modified Heavy Edge Matching . . . 66
5.1.4 Light Heavy Edge Matching . . . 68
5.2 Coarsening Parameters . . . 69

6 PARTITIONING . . . 70
6.1 Partitioning Process . . . 70
6.1.1 METIS Partitioning . . . 71
6.1.2 Quantum Annealing . . . 72
6.1.3 Quantum Approximation Optimization Algorithm . . . 74
6.2 Partitioning Parameters . . . 75
6.3 Running the Hardware Backends . . . 77

7 UNCOARSENING . . . 80
7.1 Uncoarsening Process . . . 80
7.2 Uncoarsening parameters . . . 82

8 PERMUTATION AND PROFILING . . . 84
8.1 Matrix Permutation . . . 84
8.2 Matrix Profiling . . . 85

9 RESULTS . . . 87
9.1 Metrics . . . 87
9.2 Demonstrators . . . 88
9.3 Experimental Setup . . . 89
9.4 Results for Demonstrator 01 . . . 92
9.4.1 METIS Partitioning . . . 92
9.4.2 Quantum Annealing Hardware Partitioning . . . 94
9.4.3 Quantum Annealing Emulated Partitioning . . . 95
9.4.4 QAOA Emulated Partitioning . . . 97

9.4.5 Numerical comparison across methods . . . 99
9.5 Results for Demonstrator 02 . . . 100
9.5.1 METIS Partitioning . . . 100
9.5.2 Quantum Annealing Hardware . . . 102
9.5.3 Quantum Annealing Emulated Partitioning . . . 103
9.5.4 QAOA Emulated Partitioning . . . 105
9.5.5 Numerical comparison across methods . . . 106
9.6 Cross-Demonstrator Comparison . . . 107
9.7 Experimental Limitations . . . 109

10 CONCLUSION . . . 111
10.1 Main Contributions . . . 111
10.2 Main Findings . . . 111
10.3 Limitations . . . 112
10.4 Future Work . . . 112
10.5 Final Remarks . . . 113

Bibliography . . . 114

A Appendix . . . 119

1 INTRODUCTION

At this chapter will be presented the fundaments that guide the development of this work. Discussing in section 1.1 the motivation and the context that brings the necessity of evaluating the use of quantum graph partitioning in domain decomposition. Further on will be depicted briefly in section 1.2 the procedure utilized at this work to achieve the domain decomposition of the meshes. In section 1.3 will be cleared the general and specific objectives of this work. And in section 1.4 is described how this thesis chapters were organized.

1.1 Motivation

Improving machining dimensional accuracy, surface integrity, productivity, and tool life has direct economic and engineering impact (Arrazola et al., 2013, p.695-696). Process improvement often relies on repeated trial-and-error: changing cutting parameters, tooling, or process strategies and validating outcomes experimentally. This iterative approach can be expensive and time-consuming.

To reduce experimental effort, physics-based simulations are widely used to predict the workpiece response during and after machining. Such simulations aim to reproduce relevant phenomena at a useful fidelity while remaining computationally feasible (Cimino et al., 2019, Klocke and Straube, 2004). A typical simulation workflow includes geometry preparation, discretization, assembly of system operators, parallelization of the computational workload, and finally the evaluation of structural responses such as vibrations under applied loads.

This paradigm is closely related to the ideas of virtual prototyping and digital twins, where a digital model is used to analyze and optimize a physical system before physical testing (Negri et al., 2017, p.1). The combination of complex geometries, nonlinear material behavior, and multi-physics couplings can challenge both accuracy and computational cost. Consequently, improving simulation efficiency without sacrificing essential physical fidelity remains a central research direction.

1.2 Justification

In large-scale structural simulation, discretization via the finite element method produces sparse system matrices, stiffness and mass operators. For industrial models, the number of degrees of freedom can easily exceed millions, which makes high-fidelity simulation dependent on parallel computing. Domain decomposition methods address this by splitting the global problem into subproblems that can be processed concurrently, while

enforcing coupling across interfaces (Smith et al., 1996, Toselli and Widlund, 2005). In many software stacks, this parallelization step is implemented by mapping the discretized model to a graph and then partitioning that graph to balance workload and minimize communication (Buluç et al., 2013, p.4,5). Mesh tool chains commonly expose this functionality. For example, Gmsh provides mesh partitioning and can use external partitioners such as METIS.

Graph partitioning is the computational kernel of this pipeline step. Given a graph representation of the discrete problem, the goal is to split vertices into p parts such that the parts are balanced and the edge cut is small, thereby reducing inter-processor communication. These constraints are crucial because communication and synchronization typically dominate runtime at scale, and poor balance leaves processors idle while others remain overloaded.

Balanced graph partitioning is a NP-complete combinatorial problem, motivating the use of heuristics (Garey and Johnson, 1979, p.87). State-of-the-art solvers therefore employ multilevel strategies: they coarsen the graph, compute an initial partition on the coarse instance, and then uncoarse while refining the cut. This approach has proven effective across many application domains, but it still leaves room for improvement, especially for very large instances and repeated runs in simulation pipelines.

Because graph partitioning is combinatorial and can be expressed in quadratic binary forms, it also connects naturally to quantum optimization formulations such as Ising and Quadratic Unconstrained Binary Optimization (QUBO) models (Lucas, 2014, p.4). This has motivated research on quantum and hybrid quantum-classical approaches such as Quantum Annealing (QA) and Quantum Approximate Optimization Algorithm (QAOA) for tackling such optimization problems (Farhi et al., 2014, p.2,7) (Johnson et al., 2011). In this thesis, in order to develop the domain decomposition of the mesh, the focus is on the partitioning step within the multilevel pipeline and investigating how emerging quantum optimization methods could be positioned relative to established classical heuristics.

1.3 Objectives

The main goal of this thesis is to develop and evaluate a hybrid multilevel graph-partitioning workflow for domain decomposition of large finite-element models, using METIS as the classical reference and quantum or quantum-inspired backends as alternative coarse-level partitioning strategies. The study focuses not only on obtaining valid partitions, but also on understanding how the resulting symmetric matrix permutations affect structural properties that are relevant for preconditioned linear solvers.

1. To construct the graph-partitioning workflow required for domain decomposition, starting from the finite-element matrices and including graph construction, coarsening, coarse-level partitioning, uncoarsening, refinement, and the final permutation of the stiffness and mass matrices.

2. To establish a classical baseline using METIS within the same multilevel framework, so that the quality of the recovered partitions and permutations can be compared against quantum and quantum-inspired approaches under consistent settings.

3. To formulate the coarse graph-partitioning problem as a balanced-cut QUBO/Ising model suitable for execution with quantum annealing and QAOA-based methods, making explicit the mapping from the reduced graph to a quantum-computable optimization problem.

4. To implement and integrate the selected backends, namely hardware and emulated quantum annealing together with emulated QAOA, into a common simulation pipeline that can recover a complete partition and apply it to the original matrices.

5. To evaluate the resulting reorderings on representative demonstrator geometries using structural metrics such as bandwidth, average bandwidth, and fraction of nonzeros inside narrow diagonal bands, and to compare these results with the classical baseline.

6. To discuss the practical implications of these results for domain decomposition and preconditioned solver construction, including the current strengths, limitations, and realistic role of quantum and quantum-inspired partitioning approaches in large-scale computational engineering.

1.4 Work Structure

At the next chapters of this work will be discussed in details the architecture pipeline developed to implement the domain decomposition. Will also be pinpointed the requirements to run the pipeline using both the stiffness and mass matrices and all the parameters to configure the permutation.

Starting with the chapter 2, where will be given the fundamental background necessary to understand the methodology used at the thesis and in 3 where the state of the art algorithms and software utilized for solving the problems stated here at this thesis. Later, in chapter 4 will be discussed how the pipeline were developed, what one need in order to run the software and what are the possibilities of results. Continuing with chapters 5, 6, and 7, where will be discussed the details of implementations of the coarsening, partitioning and uncoarsening algorithms used both in the METIS Library

and the Quantum Graph Partitioning (QGP) respectively. In chapter 8 will be breafly discussed how the proffiling and permutation of the matrix were performed and the main metrics extracted from the matrix. Chapter 9 presents the results obtained for the demonstrators used as entries for the pipeline, here the main differences between the METIS and QGP approach will be pinpointed and evaluated. And chapter 10 will be exploited the final considerations, and future work that can be done in order to continue further with the development of better performing approach for solving the domain decomposition problem.

2 THEORETICAL BACKGROUND

This chapter reviews the theoretical background, summarizing the foundational concepts across the areas and formulate mathematically the computational challenge tackled at this work. First in section 2.1, will be built the intuition for the main problem been solved at this thesis. Later on section 2.2 will be discussed how this problem is aimed to be solved using domain decomposition technics. Section 2.3, will be given the building blocks for understanding the quantum algorithms utilized at the thesis. In section 2.4, will be discussed how the domain decomposition can be mapped into a graph partitioning problem. Later on section 2.5 is shown the quantum algorithms utilized to solve the graph partitioning problem and what are their underlying functionalities. And in section 2.6 is depicted both the emulated and physical hardware used to run the quantum algorithms.

2.1 Finite Element Analysis

Finite element analysis (FEA) is a numerical method used to approximate the response of physical systems to loads and boundary conditions. It is widely applied in engineering and the applied sciences to study, for example, structural deformation, vibrations, heat transfer, and fluid flow (Fadiji et al., 2018, p.21). In an engineering context, the objective is often to predict how a structure responds to external forces, which supports design decisions by providing quantitative estimates of displacements, stresses, temperatures, or other field variables. Although FEA is strongly associated with structural mechanics, the same discretization principles are used across many domains, including heat conduction, electromagnetics, and fluid dynamics. As computational resources have improved, FEA has become an effective alternative to many expensive or time-consuming experimental procedures, particularly during early design stages . A validated numerical model enables parametric studies, rapid evaluation of design modifications, and the simulation of loading scenarios that may be unsafe or impractical to test physically. In addition, finer meshes and improved solution methods allow for more detailed representations of complex geometries, which can increase reliability and confidence when the model is appropriately verified (Fadiji et al., 2018, p.21).

The advantages of FEA include the ability to obtain spatially resolved field quantities, to reuse a single model under multiple loading conditions, and to visualize results such as stress or temperature distributions through post-processing tools.

In FEA, a physical continuous domain is approximated to a model called a mesh. The mesh consists of elements connected at nodes. The core principle is discretization, where a continuous boundary value problem is replaced by a finite-dimensional approximation defined on the mesh. In typical formulations, the field is

approximated locally within each element by piecewise polynomial shape functions, which produces a system of algebraic equations for nodal (Cook et al., 2002, p.4). At the element level, a standard linear relationship can be written as

$$K^e u^e = f^e, \tag{2.1}$$

where K^e is the element stiffness matrix, u^e collects element nodal displacements, and f^e is the equivalent nodal force vector (Fadiji et al., 2018, p.22). The assembly over all elements yields the global system

$$Ku = f, \tag{2.2}$$

where K is the global stiffness matrix, u is the global displacement vector, and f is the global force vector. After boundary conditions are applied, the system is solved for nodal displacements, and secondary fields such as strain and stress are recovered at the element level (Cook et al., 2002, p.4) . In practice, FEA workflows are commonly described by three stages: pre-processing, analysis (solver), and post-processing (Srirekha and Bashetty, 2010). Pre-processing includes geometry definition, mesh generation, material property assignment, and specification of loads and boundary conditions (Fadiji et al., 2018, p.22). The choice of mesh density and element type strongly influences accuracy and cost. Regions with geometric curvature or high gradients typically require refinement, while simpler regions can often be represented with fewer elements. In the analysis phase, the finite element code forms element matrices, assembles the global system, applies boundary conditions, and solves the resulting algebraic equations. Depending on the physics and modeling assumptions, the system may be linear or nonlinear, and the solver may be direct or iterative (Nikishkov, 2004, p.5). Large-scale problems often require iterative solvers and preconditioning to achieve acceptable performance. Post-processing converts numerical results into interpretable outputs such as contour plots, deformed shapes, and animations (Fadiji et al., 2018, p.23). Verification steps, such as mesh refinement studies, are typically performed to assess numerical stability and solution sensitivity with respect to discretization and input parameters.

2.1.1 Motion Equation Formulation

Consider two nodes connected by a linear elastic spring in one dimension. If one node is fixed and the other is displaced by u, Hooke's law gives

$$F = ku, \tag{2.3}$$

this is the same relation represented in Equation 2.2. If both nodes are free to move, with displacements u_1 and u_2, the spring force depends on the relative elongation $u_2 - u_1$. The corresponding equilibrium relation can be written as

$$\begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \tag{2.4}$$

which is a local stiffness matrix for a two-node spring element. In structural FEA, element stiffness matrices arise from the constitutive law and the element interpolation, typically through an integral of the form $\int B^T D B \, d\Omega$ (Cook et al., 2002, p. 110). The global stiffness matrix K is then obtained by assembling contributions from all elements according to mesh connectivity (Nikishkov, 2004, p.5). For linear elastic materials, stress and strain are related by a constitutive equation. In the simplest one-dimensional form,

$$\sigma = E\epsilon, \tag{2.5}$$

where E is Young's modulus. In multidimensional elasticity, this generalizes to tensor form, but the underlying role is the same: material parameters enter the stiffness through the constitutive matrix (Cook et al., 2002, p.20-21).

Once the element contributions are assembled, the stiffness matrix K represents the internal restoring forces generated by elastic deformation. In a static setting, K connects nodal displacements to nodal forces. In a dynamic setting, the same restoring forces oppose motion and, together with inertia, determine how the structure vibrates. Introducing inertia via the mass matrix M and dissipation via C leads to the semi-discrete structural dynamics model (Danz et al., 2025, p.3):

$$M\ddot{u}(t) + C\dot{u}(t) + Ku(t) = f(t), \tag{2.6}$$

where $u(t) \in \mathbb{R}^N$ collects nodal displacements, $f(t)$ is the external force vector, M is the mass matrix, C is the damping matrix, and K is the stiffness matrix. In many applications, M and K are symmetric, and M is positive definite (Danz et al., 2025, p. 3).

For modal analysis, one typically studies the natural dynamics of the structure by considering free vibration, with no external forcing and often neglecting damping. Removing these two elements, the Equation 2.6 then becomes

$$M\ddot{u}(t) + Ku(t) = 0. \tag{2.7}$$

This equation is a linear time-invariant second-order system. Such systems admit solutions that oscillate at characteristic frequencies with fixed spatial patterns. These patterns

are the mode shapes, and the associated frequencies are the natural frequencies. This motivates seeking solutions in separable form, where the time dependence is purely oscillatory.

2.1.2 The Eigenvalue Problem Formulation

Assume a harmonic solution for equation 2.7 of the form

$$u(t) = \phi e^{i\omega t}, \tag{2.8}$$

where $\phi \in \mathbb{R}^N$ is the mode shape and ω is the angular frequency. Substituting Equation 2.8 into equation 2.7 yields the generalized eigenvalue problem

$$(K - \omega^2 M)\phi = 0, \tag{2.9}$$

and is the main equation to be solved using the domain decomposition at this work. For computational approaches, and in particular for mappings used in quantum algorithms, it is convenient to convert Equation 2.9 into a standard eigenvalue problem. Since M is positive definite, it admits a symmetric factorization $M = M^{1/2}M^{1/2}$ (Danz et al., 2025, p.3). Defining the mass-normalized eigenvector $v = M^{1/2}\phi$ allows us to formulate the symmetric mass-normalized stiffness matrix

$$H = M^{-1/2}KM^{-1/2}. \tag{2.10}$$

Substituting these three definitions into Equation 2.9 yields the standard eigenvalue problem

$$Hv = \omega^2 v \tag{2.11}$$

or, as it typically appears in standard Hermitian eigenproblems with $\lambda = \omega^2$,

$$Hv = \lambda v. \tag{2.12}$$

Equation 2.12 defines the K and M system as a standard eigenvalue problem. To later use this formulation to find the frequencies of vibration ω and the direction of the vibration ϕ, one must first diagonalize the matrix H. Because H is symmetric (Danz et al., 2025, p.3), it can be diagonalized by an orthogonal matrix W,

$$\Lambda = W^T HW, \tag{2.13}$$

where Λ is a diagonal matrix with entries $\lambda_j = \omega_j^2$. To determine the natural frequencies and mode shapes of the system, the coupled equations of motion must be transformed into

a set of independent equations. This is achieved by projecting the physical displacements onto the normal mode coordinates. By applying the coordinate transformation

$$y = W^T M^{1/2} u, \tag{2.14}$$

where y represents the vector of modal coordinates and W is the eigenvector matrix, the system is diagonalized. This transformation effectively decouples the complex multi-degree-of-freedom system into a set of independent, single-degree-of-freedom oscillators, expressed in matrix form as

$$\ddot{y} = -\Lambda y, \tag{2.15}$$

where Λ is the diagonal spectral matrix containing the eigenvalues. Written component-wise, each independent oscillator is governed by the simple harmonic equation

$$\frac{d^2 y_j}{dt^2} = -\lambda_j y_j, \tag{2.16}$$

where λ_j corresponds to the squared natural frequency of the j-th normal mode.

Thus, each mode evolves as a simple harmonic motion with frequency $\omega_j = \sqrt{\lambda_j}$. The physical displacement can be recovered via $u = M^{-1/2} W y$. In the special case of a lumped mass matrix, $(M^{-1/2})_{uu} = 1/\sqrt{m_u}$, which yields

$$u_u = \frac{1}{\sqrt{m_u}} \sum_{j=1}^{N} W_{uj} y_j. \tag{2.17}$$

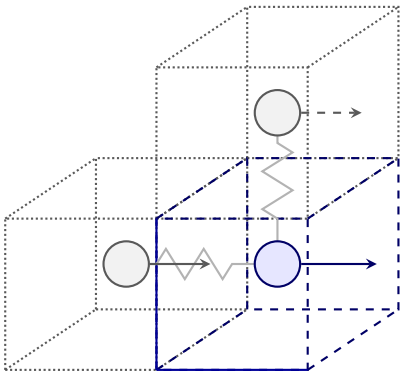
Equation 2.17 describes the displacement of a specific node in a specific direction, see Figure 1. For reaching at the equation 2.17, one must solve the problem stated at equation 2.9 that highlights the main computational bottleneck: computing eigenpairs of large sparse matrices. Dense eigensolvers have cubic time complexity, $O(N^3)$, in the matrix dimension, while sparse methods typically compute only a subset of eigenpairs using repeated sparse matrix-vector products, whose cost scales with the number of non-zeros in K and M and with the number of iterations required (Danz et al., 2025). For very large models, this motivates the use of strategies such as domain decomposition, model reduction, and parallelism, which are discussed in the following sections.

2.2 Domain Decomposition

2.2.1 Divide and Conquer

As established previously, the finite element discretization of large industrial components often leads to large-scale linear systems and, in dynamic analyses, to generalized eigenvalue problems of the form of Equation 2.9 where K and M are the stiffness and mass

Figure 1: Physical displacement of a specific node in a specific direction.



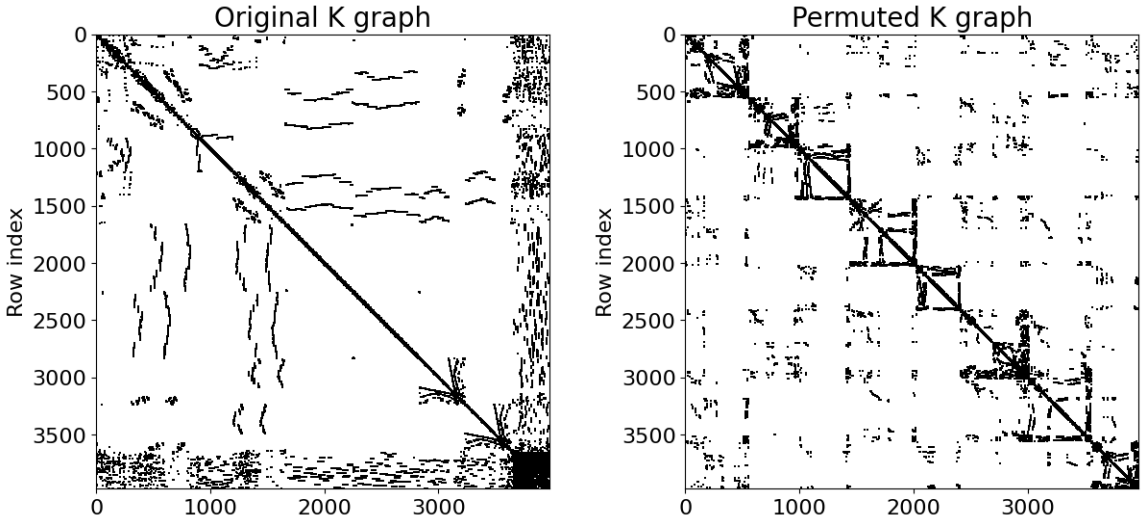
Source: Yannick Bergs 2024.

matrices, respectively. The size of these sparse matrices can become a severe limitation in terms of both memory and runtime. Champaney et al., in (Champaney et al., 1997, p.255, 258), used a model with more than 3.4×10^5 degrees of freedom to demonstrate practical computational limits. Modern industrial models can easily exceed 10^6 degrees of freedom, making monolithic approaches increasingly difficult due to its cubic, $\mathcal{O}(N^3)$, complexity nature.

Although K and M are sparse, direct factorization based methods can still become prohibitive at large scales due to fill-in and memory growth. In the worst case, dense direct solvers scale as $\mathcal{O}(N^3)$ in time and $\mathcal{O}(N^2)$ in memory, where N is the number of degrees of freedom. This motivates methods that exploit structure and parallelism strategies such as Divide and Conquer.

Domain Decomposition (DD) address this bottleneck by splitting a large problem into smaller subproblems that can be solved concurrently (Gropp and Keyes, 1989). The central idea is to decompose the physical domain into subdomains, solve local problems in parallel. This enables scalable parallel algorithms that follow a divide and conquer strategy on modern scientific computing platforms (Su and Fulton, 1993). At this thesis, the DD is the main objective and the ultimate goal of the work. Regarding matrices, domain decomposition is just performing the permutation of the matrix lines, so the nodes are grouped mainly at the diagonal in such a way that when the matrix is scattered into CPU's, the communication between the CPU's is minimized and the workload of each processor can be balanced. An example of the result of this lines permutation process can be seen at the Figure 2. The goal of the lines permutation, as already stated before, is to group the the nodes in p blocks (subdomains) throughout the diagonal therefore ideally each subdomain would have n/p nodes, where n is the total number of nodes.

Figure 2: Comparison of original and permuted matrix K. At the left one can see the K original before applying the permutation and at the right the K matrix after applying the permutation.



Source: Author (2026)

Let the physical domain Ω be partitioned into P non-overlapping subdomains $\Omega_1, \ldots, \Omega_P$ such that

$$\Omega = \bigcup_{i=1}^{P} \Omega_i \quad \text{and} \quad \Omega_i \cap \Omega_j = \varnothing \quad \forall i \neq j. \tag{2.18}$$

If subdomains were completely independent, assigning each Ω_i to a separate processor could yield an ideal super-linear speedup (Herrera-Revilla et al., 2019, p.4). In practice, coupling occurs at interfaces, since discrete degrees of freedom interact across subdomain boundaries. Therefore, the conquer step requires both the solution of local problems and the enforcement of interface conditions that maintain global consistency.

2.2.2 Parallelism Costs

Parallel implementations introduce costs beyond floating-point operations. In (Gropp and Keyes, 1989), Gropp and Keyes emphasize three main contributors: intrinsically serial operations such as global reductions, synchronization overheads required to coordinate access to shared data in shared memory models, and hardware and communication costs, typically characterized by bandwidth limits and message startup latency at message passing architectures.

2.2.2.1 Shared Memory Architectures

In shared memory architectures, processors access a global memory space, which makes communication implicit but introduces contention and bandwidth bottlenecks.

Synchronization primitives such as barriers and critical sections also introduce overheads, frequently modeled with $\log p$ terms (Jordan, 1987).

For shared memory, Gropp and Keyes (Gropp and Keyes, 1989) describe models where performance is limited by contention on shared resources and synchronization. A representative bound for strip decompositions is

$$T(n,p) \approx \frac{n^2}{p}\left(a + \frac{b \cdot p}{\min(p, P_1)}\right) + 2\left(\frac{n^2}{p} + \log p\right) + 2\left(\frac{n^2}{p}\log\frac{n}{p}\right)\left(c + \frac{dp}{\min(p, P_2)}\right) + n\log n + 3\log p, \tag{2.19}$$

where P_1 and P_2 model saturation limits of shared resources, and a, b, c, d represent local arithmetic and memory access costs.

2.2.2.2 Message Passing Architectures

At message passing architectures, each processor has private memory and communicates by sending explicit messages. A common cost model for transferring n words is $\mathcal{O}(s+rn)$, where s is the startup latency and r is the per-word transfer time. Global reductions depend strongly on network topology. For example, the cost can scale like $\mathcal{O}((s+rn)\log p)$ on a hypercube and like $\mathcal{O}((s+rn)\sqrt{p})$ on a mesh. Following Gropp and Keyes (Gropp and Keyes, 1989), a representative per-iteration time model for a strip decomposition on a message passing system can be expressed as

$$\begin{aligned} T(n,p) \approx{}& \frac{n^2}{p} + \left(\frac{n^2}{p} + \log p\right) + \frac{n^2}{p}\log\frac{n}{p} + n\log n \\ &+ (s+rn) + (s+r)\log p + (s+rn) + (s+rn)\,, \end{aligned} \tag{2.20}$$

where n denotes the number of unknowns per side, p is the number of processors, s is message startup latency, and r is the transfer time per word.

In practice, domain decomposition algorithms trade increased per-iteration work and communication for a significantly reduced overall iteration count. Specifically, Gropp and Keyes (Gropp and Keyes, 1989) discuss representative regimes, including decoupled methods (minimal communication), local communication preconditioning, and approaches that introduce global coupling steps for improved convergence. Such trade-offs are central to choosing an algorithm that achieves both numerical efficiency and parallel scalability.

2.2.3 Preconditioning and Solver Efficiency

Ultimately, the principal performance improvement when using these domain decomposition techniques is realized when solving the final permuted matrix with a

preconditioned solver. By intelligently exploiting the newly concentrated matrix structure, this type of solver is able to drastically reduce the amount of iterations necessary to converge to a robust solution. This efficiency gain is fundamentally driven by the use of structural heuristics, which carefully balance the trade-off between finding an optimal structural ordering and minimizing the total convergence time. To fully appreciate how preconditioning leverages these structural changes, it is first necessary to establish the operational baseline of a system solved without them as in the Unconditioned Solvers.

2.2.3.1 Unconditioned Solvers

The most common foundational approach relies on unconditioned solvers, which operate directly on the original linear system. Krylov subspace methods, such as the Conjugate Gradient (CG) method and the Generalized Minimal Residual (GMRES) method, are highly effective iterative techniques for approximating solutions to large, sparse linear systems. These methods operate by constructing an approximate solution from a sequence of expanding Krylov subspaces, defined as the span of vectors $\mathcal{K}_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}$ (Fuhrmann, 2024, p.10) (Kumlien, 2024, p.10). The baseline iteration complexity and the convergence rate of these unconditioned solvers are intrinsically tied to the eigenvalue distribution and the condition number of the coefficient matrix. Because Krylov methods are strictly spectrally driven, applying a permuted matrix directly to an unconditioned solver will not decrease the number of iterations required for convergence. The similarity transformation associated with structural permutations preserves the eigenvalues exactly (Golub and Van Loan, 2013, p.67), meaning the inherent mathematical difficulty of the problem remains completely unchanged from the perspective of an unconditioned Krylov solver.

2.2.3.2 Preconditioned Solvers

While unconditioned methods cannot exploit the reordered matrix structure to reduce iterations, introducing a preconditioner fundamentally changes this dynamic. To combat the slow convergence associated with ill-conditioned matrices, the original linear system $Ax = b$ is transformed using a preconditioner M into an equivalent system, such as $M^{-1}Ax = M^{-1}b$, which possesses a significantly smaller condition number κ and a more favorable eigenvalue clustering. A standard and highly efficient approach for generating M is through sparse approximations like incomplete LU (ILU) or incomplete Cholesky factorizations, which manage memory usage and computational overhead by employing dropping rules to discard elements outside a specific bandwidth or below a certain magnitude threshold (Kumlien, 2024, p.19). It is within this preconditioned framework that matrix permutation unlocks its greatest computational advantage. By structurally reordering the matrix to concentrate the largest-magnitude entries near the main diagonal,

the preconditioning algorithm is forced to drop only the sparse, less vital outliers. This ensures that the resulting preconditioner M retains a much higher-fidelity approximation of the original matrix without exceeding memory limits, which drastically accelerates the convergence of the Krylov solver.

2.2.3.3 Matrix Permutation Eigenvalues Preservation

While extracting a preconditioner from a concentrated matrix structure yields clear computational benefits, a natural concern is whether this structural scrambling alters the fundamental mathematical problem. To ensure that reordering the matrix does not compromise the underlying physical or mathematical system, it is vital to understand the properties of the permutation transformation. The structural reordering of a symmetric stiffness matrix K can be expressed as PKP^T, where P is an orthogonal permutation matrix satisfying $P^{-1} = P^T$. Because of this orthogonal property, the transformation PKP^{-1} is a standard similarity transformation (Golub and Van Loan, 2013, p.20). A foundational property of similarity transformations is that they perfectly preserve the characteristic polynomial, and consequently, the exact eigenvalues of the original matrix. Therefore, the condition number, the system's stability, and the spectral limits remain completely unaffected. The PKP^T permutation acts purely as a structural optimizer to facilitate the extraction of an efficient preconditioner, mathematically guaranteeing that the reordering process will never break the matrices down.

2.2.3.4 Structural Metrics for Matrix Profiling

To accurately measure and predict the resulting reduction in solver iterations, structural profiling metrics are utilized to quantify the matrix's newly formed density pattern. The primary metric is the matrix bandwidth B, formally defined as the maximum distance of any non-zero element from the main diagonal: $B = \max\{|i - j| : a_{ij} \neq 0\}$ (Golub and Van Loan, 2013, p. 176). For symmetric matrices, this is often evaluated using the semi-bandwidth p. While absolute bandwidth dictates the maximum spread of elements, effective diagonal concentration is equally critical. It measures the weight or fraction of non-zero elements bounded within a tightly localized range from the main diagonal. High diagonal concentration ensures that the most critical mathematical mass of the matrix is heavily packed into the center, a structural prerequisite that directly predicts the effectiveness and accuracy of banded dropping strategies in preconditioning (Kumlien, 2024).

2.2.3.5 Factorization Complexity

Ultimately, the purpose of optimizing these structural metrics is to directly minimize the algebraic operations required to process the matrix. The computational complexity

of the factorization algorithms utilized to generate direct solvers or preconditioners is strictly governed by the structural metrics defined above. For banded systems, the theoretical worst-case complexity of factorization scales quadratically with the bandwidth, demonstrating an $\mathcal{O}(NB^2)$ scaling law. More specifically, the complexity ceiling for the standard Band Cholesky factorization is strictly bounded to floating-point operations based on the semi-bandwidth p:

$$C_{\text{factorization}} = n(p^2 + 3p) \tag{2.21}$$

Consequently, reducing the maximum absolute bandwidth mathematically lowers the theoretical ceiling of the factorization time and limits maximum memory allocation (Golub and Van Loan, 2013). However, as observed in matrix profiling paradigms, evaluating absolute bandwidth alone is insufficient to capture the true factorization complexity. Because incomplete factorizations use dynamic dropping rules based on fill-in limits, the algorithm does not rigidly execute dense operations out to the extreme absolute bandwidth edge. Instead, a densely packed inner boundary acts as the effective bandwidth. The factorization algorithm iterates heavily over this localized dense core and cheaply discards the sparse elements located in the matrix's periphery.

To understand how Avg. Bandwidth reduction can accelerate the solver, one must look at the condition number, which measures the mathematical difficulty of the original system:

$$\kappa(A) = \|A\| \cdot \|A^{-1}\| \tag{2.22}$$

By minimizing the Avg. Bandwidth, the permutation ensures a greater compactness of the non-zero values along the main diagonal. When an incomplete factorization algorithm applies its dropping rules to discard elements outside the effective band, it only discards mathematically insignificant outliers. This minimizes the approximation error E, where $A = M + E$, ensuring that the resulting preconditioner M is a high-fidelity approximation of A. Because the Avg. Bandwidth reduction produced a highly accurate preconditioner where $M \approx A$, the preconditioned operator $M^{-1}A$ strongly approaches the identity matrix I. The condition number formulation of this newly preconditioned system becomes

$$\kappa(M^{-1}A) = \|M^{-1}A\| \cdot \|(M^{-1}A)^{-1}\|. \tag{2.23}$$

This mathematically forces the preconditioned condition number $\kappa(M^{-1}A)$ to drop significantly closer to its ideal minimum of 1 (Kumlien, 2024, p.25).

According to Krylov subspace convergence theory, the number of iterations required for the solver to converge is strictly bounded by the square root of this condition number. This gives the final computational complexity

$$\text{Iterations} = \mathcal{O}\left(\sqrt{\kappa(M^{-1}A)}\right). \tag{2.24}$$

Therefore, an optimal permutation method achieves computational reduction by simultaneously lowering the theoretical $\mathcal{O}(NB^2)$ absolute bandwidth ceiling while condensing the vast majority of the calculation mass into a tight, effective computational zone. This diagonal concentration inherently minimizes the preconditioner's approximation error, plummets the condition number $\kappa(M^{-1}A)$, and ultimately guarantees a drastic reduction in the $\mathcal{O}(\sqrt{\kappa(M^{-1}A)})$ iterations required to reach a solution.

2.3 Quantum Computing Fundamentals

2.3.1 Building Blocks

In classical computation, the basic unit of information is the bit, which takes values in the set $\{0, 1\}$ (Nielsen and Chuang, 2010, p.15). In quantum computation, the analogous unit is the quantum bit, qubit. Mathematically, a single qubit is described by a unit vector in a two-dimensional complex Hilbert space, with an orthonormal computational basis $\{|0\rangle, |1\rangle\}$. Thus, a general pure qubit state can be written as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle, \tag{2.25}$$

where $\alpha, \beta \in \mathbb{C}$ and the normalization condition requires

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2.26}$$

Only relative phases are physically relevant: multiplying $|\psi\rangle$ by a global phase factor $e^{i\gamma}$ does not change measurement probabilities and can typically be ignored. Up to a global phase, any single-qubit pure state can be parameterized by two real angles $\theta \in [0, \pi]$ and $\varphi \in [0, 2\pi)$ as

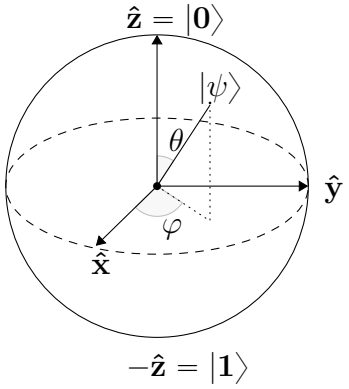
$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle. \tag{2.27}$$

This representation leads to the Bloch sphere in figure 3, where θ and φ correspond to spherical coordinates of a point on the unit sphere (Nielsen and Chuang, 2010, p.15).

2.3.1.1 Measurements

A projective measurement of the state in Eq. 2.25 in the computational basis yields outcome 0 with probability $|\alpha|^2$ and outcome 1 with probability $|\beta|^2$. Conditioned on the measurement result, the post-measurement state collapses to the corresponding basis state

Figure 3: Bloch sphere representation of a single-qubit state.



Source: Author

$|0\rangle$ or $|1\rangle$. For multiple qubits, the state space grows by tensor products. A two-qubit computational basis is $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, and a general pure state has the form

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle, \tag{2.28}$$

with normalization $\sum_{x\in\{0,1\}^2} |\alpha_x|^2 = 1$. If only the first qubit is measured and the outcome is 0, then the outcome occurs with probability

$$p(0) = |\alpha_{00}|^2 + |\alpha_{01}|^2, \tag{2.29}$$

and the post-measurement state becomes

$$|\psi'\rangle = \frac{\alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}. \tag{2.30}$$

More generally, an n-qubit pure state is specified by 2^n complex amplitudes, which illustrates the rapid growth of the underlying state space with system size.

2.3.1.2 Entanglement

Multi-qubit systems exhibit correlations that cannot be reproduced by classical probability models. A standard example is the Bell (EPR) state

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{2.31}$$

Measuring the first qubit yields outcomes 0 and 1 each with probability 1/2, and the second qubit is perfectly correlated with the first: conditioned on outcome 0 the state becomes $|00\rangle$, while conditioned on outcome 1 it becomes $|11\rangle$. Such entangled states

are a key resource in quantum information processing and will reappear in later sections when discussing quantum circuits and algorithms.

2.3.2 Quantum Circuits

2.3.2.1 Single-Qubit Gates

Quantum logic gates are unitary operations that evolve the state of one or more qubits. For a single qubit, gates are represented by 2×2 unitary matrices acting on state vectors in the computational basis $\{|0\rangle, |1\rangle\}$. Common single-qubit gates include the Pauli gates (X, Y, Z), phase gates (S, T), the Hadamard gate (H), and continuous rotation gates:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \tag{2.32}$$

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \qquad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \tag{2.33}$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \qquad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}. \tag{2.34}$$

A useful continuous family are rotations about the Bloch-sphere axes, defined via Pauli matrices:

$$R_x(\theta) = e^{-i\theta X/2} = \begin{bmatrix} \cos\dfrac{\theta}{2} & -i\sin\dfrac{\theta}{2} \\ -i\sin\dfrac{\theta}{2} & \cos\dfrac{\theta}{2} \end{bmatrix} \tag{2.35}$$

$$R_y(\theta) = e^{-i\theta Y/2} = \begin{bmatrix} \cos\dfrac{\theta}{2} & -\sin\dfrac{\theta}{2} \\ \sin\dfrac{\theta}{2} & \cos\dfrac{\theta}{2} \end{bmatrix} \tag{2.36}$$

$$R_z(\theta) = e^{-i\theta Z/2} = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}. \tag{2.37}$$

2.3.2.2 Multi-Qubit Gates

For multiple qubits, the state space is described by tensor products, and gates are represented by $2^n \times 2^n$ unitaries (Nielsen and Chuang, 2010, p.94). The prototypical two-qubit gate is the controlled-NOT (CNOT), which flips the *target* qubit if and only if the *control* qubit is $|1\rangle$:

$$|00\rangle \mapsto |00\rangle, \quad |01\rangle \mapsto |01\rangle, \quad |10\rangle \mapsto |11\rangle, \quad |11\rangle \mapsto |10\rangle. \tag{2.38}$$

Equivalently, it can be written as

$$\text{CNOT} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X, \tag{2.39}$$

where I is the single-qubit identity.

More generally, a controlled-unitary gate applies a unitary U to the target conditioned on the control being $|1\rangle$:

$$C(U) = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U. \tag{2.40}$$

Controlled operations are central to many quantum algorithms, including phase estimation.

2.3.2.3 Universality and Standard Circuit Diagrams

Quantum circuits are typically drawn as horizontal wires, qubits, evolving from left to right in time. Single-qubit gates are shown as boxes acting on one wire, while controlled gates are drawn with a filled control dot on the control wire connected to a target symbol (e.g., $\oplus$ for CNOT).

A gate set is called universal if any multi-qubit unitary can be approximated to arbitrary accuracy using a finite circuit over that set. A standard theoretical result is that:

- Arbitrary single-qubit unitaries together with CNOT form a universal family for quantum computation.
- In practice, one often uses a finite universal gate set such as $\{H, T, \text{CNOT}\}$, which can approximate any unitary to desired precision.

A useful decomposition for single-qubit operations is the Euler-angle form: any 2×2 unitary U can be written as a sequence of rotations, for example

$$U = e^{i\alpha}\, R_x(\beta)\, R_y(\gamma)\, R_z(\delta), \tag{2.41}$$

for real parameters $\alpha, \beta, \gamma, \delta$. This shows how a small set of elementary rotations can generate general single-qubit gates.

2.3.3 Quantum Phase Estimation

As derived previously, free vibration in a finite element model leads to the generalized eigenvalue problem

$$(K - \omega^2 M)\phi = 0, \tag{2.42}$$

where K and M are the stiffness and mass matrices. Since M is symmetric positive definite in standard structural models, Eq. 2.42 can be mapped to a standard Hermitian eigenvalue problem by introducing the mass-normalized variables $v = M^{1/2}\phi$:

$$Hv = \lambda v, \qquad H := M^{-1/2} K M^{-1/2}, \qquad \lambda := \omega^2. \tag{2.43}$$

Therefore, computing natural frequencies and mode shapes reduces to extracting eigenpairs (λ, v) of a large sparse Hermitian matrix.

In principle, Quantum Phase Estimation (QPE) provides an exact quantum procedure to estimate eigenvalues of such operators, but QPE requires access to a unitary operator. For a Hermitian matrix H, a standard choice is

$$U := e^{-iHt}, \tag{2.44}$$

for some evolution time t. If $H|v\rangle = \lambda|v\rangle$, then $U|v\rangle = e^{-i\lambda t}|v\rangle$, and QPE can be used to estimate the phase $e^{-i\lambda t}$, and thus λ up to scaling and periodicity.

The central practical difficulty is implementing controlled powers of U, namely controlled-U^{2^j} operations, for this reason we assume that we have available black boxes, some times called oracles, capable of performing this unitary which dominates the cost of QPE at scale. A widely used theoretical framework to construct such unitaries is block-encoding. In block-encoding, one embeds a matrix (such as H or L) into a larger unitary U_H acting on additional ancilla qubits such that

$$\left(\langle 0|^{\otimes a} \otimes I\right) U_H \left(|0\rangle^{\otimes a} \otimes I\right) = \frac{H}{\alpha}, \tag{2.45}$$

for some normalization constant $\alpha \geqslant \|H\|$. Once a block-encoding is available, standard quantum techniques can be used to implement approximations of e^{-iHt} and its controlled powers. Consequently, QPE is conceptually applicable to eigenvalue problems arising from FEM operators, but its feasibility depends on the cost of constructing and simulating the corresponding block-encoded operators.

QPE addresses the following task: given a unitary operator U and an eigenvector $|u\rangle$ such that

$$U|v\rangle = e^{2\pi i\varphi}|v\rangle, \qquad \varphi \in [0, 1), \tag{2.46}$$

estimate the eigenphase φ to N bits of precision. The standard QPE procedure uses the Quantum Fourier Transform (QFT) as a circuit subroutine (Kitaev, 1995, p.3). The QFT on N qubits is defined by the mapping

$$|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i jk/N}|k\rangle, \qquad j \in \{0, \ldots, N-1\}. \tag{2.47}$$

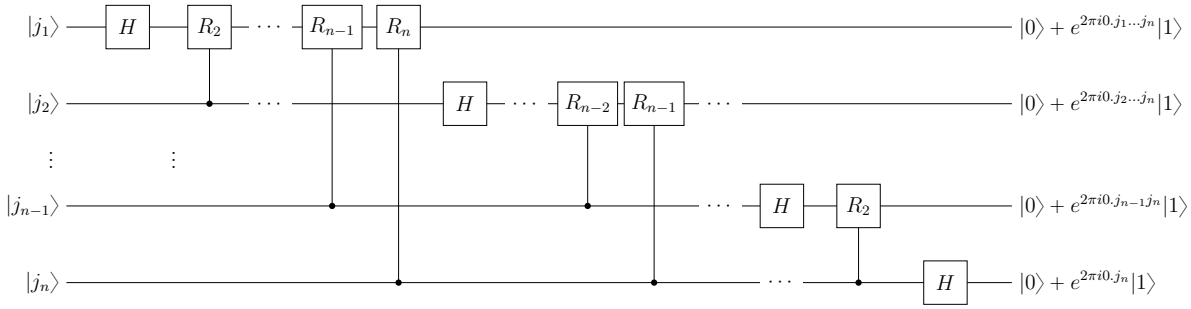
Equivalently, the action on a arbitrary state may be written

In quantum algorithms, the inverse transform $\mathrm{QFT}^\dagger$ is particularly important because it converts phase information encoded across computational basis states into a classical bitstring upon measurement (Coppersmith, 1994, p.4,5), (Nielsen and Chuang, 2010, p.221). A standard decomposition of the QFT uses Hadamard gates and controlled phase rotations

$$R_k := \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}, \tag{2.48}$$

followed by a qubit-reversal (SWAP) stage, see Figure 4 for circuit overview.

Figure 4: Circuit decomposition of the QFT



Source: (Nielsen and Chuang, 2010).

QPE circuit, represented at Figure 5, goes as follows: using two registers, first one as a phase register of t qubits initialized in $|0\rangle^{\otimes t}$ and a system register initialized in $|u\rangle$. After applying Hadamards to the phase register, controlled powers of U imprint the eigenphase onto the phase register via phase kickback. Finally, $\mathrm{QFT}^\dagger$ is applied to concentrate the amplitude on bitstrings close to the binary expansion of φ.

To connect QPE to Hermitian eigenvalue problems, one typically sets $U = e^{-iHt}$ for a Hermitian operator H. If $H|v\rangle = \lambda|v\rangle$, then $U|v\rangle = e^{-i\lambda t}|v\rangle$, which corresponds to an eigenphase φ satisfying

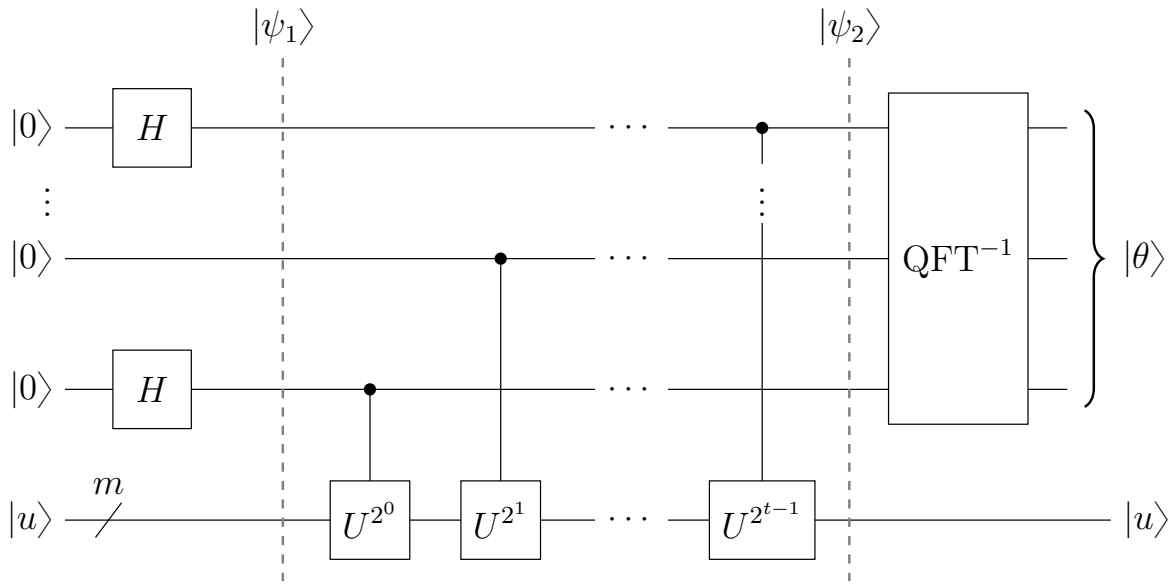
$$e^{-i\lambda t} = e^{2\pi i\varphi} \quad \Rightarrow \quad \varphi \equiv -\frac{\lambda t}{2\pi} \pmod 1. \tag{2.49}$$

Therefore, recovering λ requires a suitable scaling choice and an efficient implementation of controlled-U^{2^j}, which is typically the dominant cost in practical settings.

2.3.3.1 Approximate phases and success probability

The discussion so far describes the ideal case in which the eigenphase φ admits an exact t-bit binary expansion. In practice, φ is generally not exactly representable with t bits,

Figure 5: Circuit overview for the QPE



Source: Yannick Bergs 2025

and QPE returns a *best* t-bit approximation with high probability. Let $b \in \{0, \ldots, 2^t - 1\}$ be the integer such that

$$\frac{b}{2^t} \leqslant \varphi < \frac{b+1}{2^t}, \qquad \delta := \varphi - \frac{b}{2^t}, \qquad 0 \leqslant \delta < 2^{-t}. \tag{2.50}$$

After the controlled-U^{2^j} stage and the application of $\mathrm{QFT}^\dagger$, the first register is measured and returns an integer $m \in \{0, \ldots, 2^t - 1\}$. Nielsen and Chuang show that the probability of obtaining an outcome far from b can be bounded as follows: for any integer tolerance $e \geqslant 1$,

$$\mathbb{P}\big(|m - b| > e\big) = \frac{1}{2(e-1)}. \tag{2.51}$$

This implies that outcomes concentrate near b, and therefore QPE yields an accurate approximation to φ with high probability. If φ is exactly representable with t bits, i.e. $\varphi = b/2^t$ (equivalently $\delta = 0$), then the inverse QFT produces the basis state $|b\rangle$ deterministically and the measurement returns $m = b$ with probability one. In the generic case $\delta \neq 0$, the output distribution is peaked around b (and the nearest integers), so that rounding $m/2^t$ gives a good estimate of φ. The phase register size sets the resolution: increasing t refines the binary grid spacing 2^{-t}. In the Hamiltonian setting of Eq. 2.49, where $\varphi \equiv -\lambda t/(2\pi) \pmod 1$, an additive phase error $\Delta\varphi$ corresponds to an eigenvalue error

$$|\Delta\lambda| \approx \frac{2\pi}{t} |\Delta\varphi|, \tag{2.52}$$

so that achieving n accurate phase bits requires a phase register whose effective precision scales accordingly. To obtain an estimate accurate to n bits, one may choose $e = 2^{t-n} - 1$

in Eq. 2.51. Writing $t = n + p$ (i.e. using p extra qubits beyond the target precision), Nielsen and Chuang derive that the probability of obtaining an n-bit accurate estimate is at least

$$\mathbb{P}\big(n\text{-bit accurate}\big) \;\geqslant\; 1 - \frac{1}{2(2^p - 2)}. \tag{2.53}$$

Thus, a small number of additional phase qubits increases the success probability substantially. In practice, confidence can also be boosted by repeating QPE multiple times and taking a robust statistic of the returned phases.

The above guarantees assume the system register is initialized in an eigenstate $|v\rangle$ of U as in Eq. 2.46. If instead one prepares an arbitrary state expanded in the eigenbasis,

$$|\psi\rangle = \sum_k c_k \, |v_k\rangle, \qquad U \, |v_k\rangle = e^{2\pi i \varphi_k} \, |v_k\rangle, \tag{2.54}$$

then QPE produces the phase φ_k with probability $|c_k|^2$ (Nielsen and Chuang, 2010, p.224). Therefore, estimating a specific eigenphase requires non-negligible overlap between the prepared state and the corresponding eigenvector. Standard QPE uses t phase qubits and an inverse QFT on the phase register. Iterative and semiclassical variants reduce the number of ancilla qubits at the cost of longer circuit depth and additional classical feedforward (Griffiths and Niu, 1995). Likewise, approximate implementations of the QFT reduce the number of controlled rotations by omitting very small-angle gates, trading a controlled approximation error for improved resource requirements.

2.3.3.2 Resource requirements and practical limitations

QPE is conceptually a direct route from an operator eigenproblem to eigenvalue estimates, but its practicality is governed by the resources required to implement the controlled powers c-U^{2^j} and by the target precision (Nielsen and Chuang, 2010, p.221). The standard QPE circuit uses a phase register of t qubits, a system register to represent the eigenvector $|v\rangle$, and additional ancillas required by the implementation of U. Thus, the qubit count is not only t plus the system size, but also includes non-negligible overhead from the chosen operator access model.

QPE applies controlled powers c-U^{2^j} for $j = 0, \ldots, t-1$. When U is generated from a Hermitian operator via $U = e^{-iHt}$, the power U^{2^j} corresponds to evolution for time $2^j t$. Therefore, the total simulated evolution time scales as

$$T_{\text{sim}} \;=\; \sum_{j=0}^{t-1} 2^j \, t \;=\; \mathcal{O}((2^t - 1)\, t), \tag{2.55}$$

which grows exponentially in the number of phase qubits. Since $t = n + p$ is chosen to achieve n accurate bits with high success probability, circuit depth quickly becomes the dominant bottleneck (Nielsen and Chuang, 2010, p. 224).

The feasibility of QPE hinges on efficiently implementing e^{-iHt} and its controlled versions. A common theoretical route is to construct a block-encoding U_H and then use Hamiltonian simulation techniques to approximate e^{-iHt} to some error ϵ_{sim} (Gilyén et al., 2018, p.40) (Low and Chuang, 2019, p.3). In such frameworks, the gate and query complexity depends on the normalization factor α, the evolution time, and the tolerated simulation error. In particular, the large evolution times appearing in Eq. 2.55 translate into strong requirements on long coherent controlled simulation. QPE returns a phase modulo 1. To recover λ unambiguously, one needs a priori spectral bounds and a scaling choice for the evolution time t such that distinct eigenvalues do not map to nearly identical phases. The cleanest QPE guarantee assumes the system register is initialized in an eigenstate $|v\rangle$. For general inputs, outcomes are sampled according to eigenstate overlaps (Eq. 2.54), so obtaining a specific eigenvalue may require additional procedures to prepare or amplify the relevant eigencomponent, which introduces further overhead beyond the QPE circuit itself (Nielsen and Chuang, 2010, p. 223).

For large FEM operators, the combination of high precision requirements, deep controlled time evolutions, and overhead from block-encoding and eigenstate preparation can make full-scale QPE impractical. This motivates strategies that reduce the global problem into smaller pieces, such as domain decomposition and graph partitioning, and it explains why hybrid quantum-classical optimization approaches are often preferred when the full operator is too large to simulate coherently.

QPE provides a principled route to extract eigenvalues by estimating phases of an evolution operator $U = e^{-iHt}$. While conceptually attractive, the practical cost of implementing the required controlled time-evolution operators at scale quickly becomes prohibitive for large engineering models, making exact QPE-based workflows unrealistic for the problem sizes of interest (Gilyén et al., 2018, p.41).

For this reason, near-term quantum optimization focuses on methods that target the solution quality rather than exact spectral information. Two prominent families are the Quantum Approximate Optimization Algorithm (QAOA) (Farhi et al., 2014, p.2,7) and quantum annealing (QA) (Kadowaki and Nishimori, 1998, p.2), both of which are designed for discrete, combinatorial optimization tasks. In particular, problems such as graph partitioning naturally admit formulations where a candidate solution is represented by binary decision variables and the goal is to minimize an objective function under constraints.

2.4 Graph Partitioning

2.4.1 From mesh to graph

Graphs are widely used as abstractions for modeling relationships and data dependencies in computational problems. As a result, partitioning a graph into smaller pieces is a fundamental algorithmic operation, particularly in scientific computing and parallel processing (Hendrickson and Kolda, 1999, p.1520). Even when the final task is traversal, path finding, or flow computation, partitioning large graphs often appears as an important subproblem for complexity reduction and parallelization (Buluç et al., 2013, p.1). In parallel computing, vertices often represent units of computation and edges represent data dependencies. A good partition distributes workload evenly and reduces communication between processing units.

In many scientific computing settings, graph partitioning is used to distribute large numerical simulations across processors for making the decomposition of the domain. In the Finite Element Method (FEM), the discretized dynamics are governed by sparse system matrices, such as the mass matrix M and stiffness matrix K. A significant portion of the computational cost arises from sparse linear algebra operations, for example sparse matrix vector products in iterative solvers.

This mesh structure can be represented as an undirected graph $G = (V, E)$ that encodes the sparsity pattern of the system matrices. A common construction is:

- **Vertices (V):** Each vertex corresponds to a degree of freedom (or node) in the FEM discretization.
- **Edges (E):** An edge (v_i, v_j) exists if the global matrix contains a nonzero entry at (i, j), meaning that degrees of freedom i and j are coupled through at least one element.

To model load and communication more accurately, weights are often assigned to vertices and edges. Vertex weights represent computational work, and edge weights represent the amount of data exchange implied by coupling (Hendrickson and Kolda, 1999, p. 1520). With this mapping, distributing the simulation becomes a combinatorial graph partitioning problem.

2.4.2 Graph Partitioning Problem Definition

Let $G = (V, E)$ be an undirected graph with edge weights $\omega : E \to \mathbb{R}_{\geqslant 0}$, and optionally node weights $c : V \to \mathbb{R}_{>0}$. A k-way partition is a collection $\Pi = \{V_1, \ldots, V_k\}$ such that $V_1 \cup \cdots \cup V_k = V$ and $V_i \cap V_j = \varnothing$ for all $i \neq j$. Equivalently, a partition can be represented by an assignment vector P of length $n = |V|$ where $P[v] \in \{1, \ldots, k\}$ indicates the block of vertex v (Buluç et al., 2013, p.2,3).

2.4.2.1 Inter-block Connectivity

Given a partition Π, an edge $\{u, v\} \in E$ is called a cut edge if it connects two different blocks, i.e., if $P[u] \neq P[v]$. For two blocks V_i and V_j, define the set of edges crossing between them by

$$E_{ij} := \big\{\{u, v\} \in E \ : \ u \in V_i, \ v \in V_j\big\}, \qquad i \neq j. \tag{2.56}$$

The pairwise cut between V_i and V_j is then

$$\mathrm{cut}(V_i, V_j) := \omega(E_{ij}) := \sum_{e \in E_{ij}} \omega(e). \tag{2.57}$$

Summing over all pairs yields the total edge cut

$$\mathrm{cut}(\Pi) := \sum_{1 \leqslant i < j \leqslant k} \omega(E_{ij}) = \sum_{\{u,v\} \in E \ : \ P[u] \neq P[v]} \omega(\{u, v\}). \tag{2.58}$$

This is the most commonly used objective because it is a simple proxy for communication cost in parallel computing.

2.4.2.2 Balance Construction

To model computational workload, vertex weights $c(v)$ are used. The weight of a block is

$$c(V_i) := \sum_{v \in V_i} c(v), \qquad c(V) := \sum_{v \in V} c(v). \tag{2.59}$$

A standard balance requirement enforces that every block weight stays close to the average:

$$c(V_i) \leqslant L_{\max} := (1 + \epsilon)\frac{c(V)}{k} \qquad \text{for all } i \in \{1, \ldots, k\}, \tag{2.60}$$

where $\epsilon \geqslant 0$ is an imbalance parameter (Buluç et al., 2013, p.2,3). In the unweighted case, one sets $c(v) = 1$ and obtains $c(V_i) = |V_i|$. The balanced k-way graph partitioning problem can now be written as

$$\begin{aligned} \min_{\Pi = \{V_1, \ldots, V_k\}} \quad & \mathrm{cut}(\Pi) \\ \text{s.t.} \quad & c(V_i) \leqslant (1 + \epsilon)\frac{c(V)}{k} \quad \forall i \in \{1, \ldots, k\}. \end{aligned} \tag{2.61}$$

The outer min emphasizes that the goal is to find, among all valid partitions, the one with minimum inter-block connectivity.

2.4.2.3 Single-objective Formulation

It is often convenient to combine the cut objective and the balance constraint into a single scalar objective using a penalty term. One common choice is

$$\min_{\Pi=\{V_1,\ldots,V_k\}} \left[\text{cut}(\Pi) \;+\; \eta \sum_{i=1}^{k} \max\Big(0,\; c(V_i) - (1+\epsilon)\tfrac{c(V)}{k}\Big)^2 \right], \tag{2.62}$$

where $\eta > 0$ is a penalty parameter. The hinge term is zero when the balance constraint is satisfied and grows quadratically when it is violated.

For strict balancing or for QUBO/Ising-style encoding, a symmetric quadratic penalty is also widely used:

$$\min_{\Pi=\{V_1,\ldots,V_k\}} \left[\text{cut}(\Pi) \;+\; \eta \sum_{i=1}^{k} \Big(c(V_i) - \tfrac{c(V)}{k}\Big)^2 \right], \tag{2.63}$$

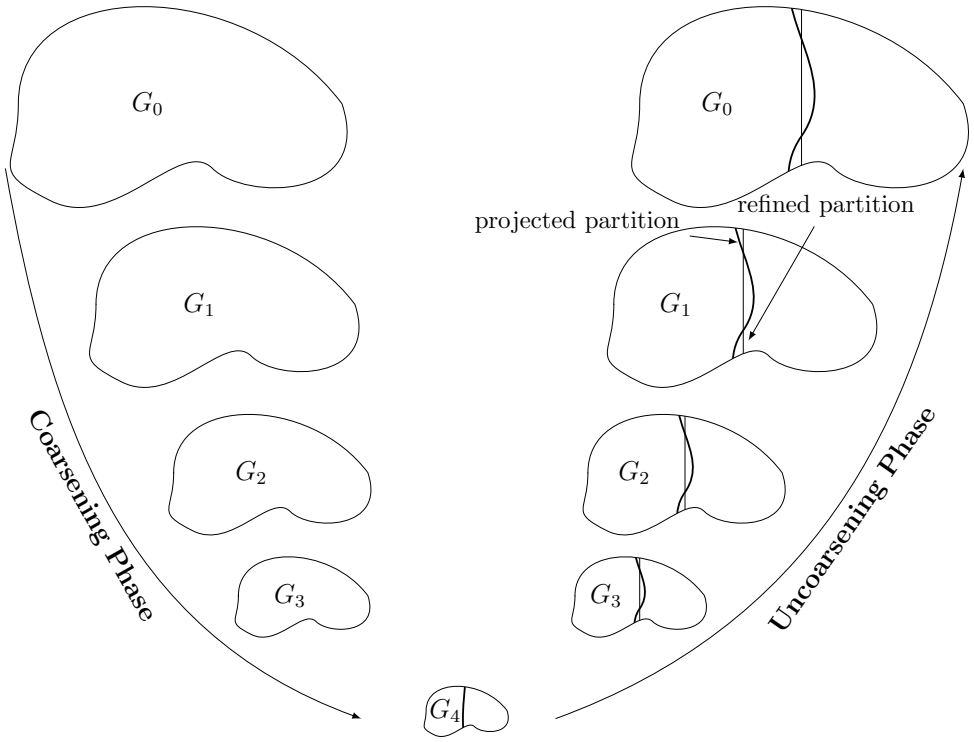
which encourages all blocks to match the average weight. This formulation is particularly useful when transforming the problem into an unconstrained binary optimization model, what we will need latter on in order to use the Quantum Annealing and Quantum Approximate Optimization Algorithm for finding a solution for this problem.

2.4.3 The Multilevel Strategy

Balanced graph partitioning is NP-complete as a decision problem and NP-hard in optimization form (Garey et al., 1976, p. 242-243). This motivates the use of heuristic algorithms that produce high-quality solutions efficiently on large graphs (Buluç et al., 2013, p.3-4) (Karypis and Kumar, 1998a, p.363). A highly successful family of heuristics for large graphs is the multilevel approach, see figure 6. Rather than partitioning the original graph directly, multilevel methods proceed in three phases: coarsening, initial partitioning, and uncoarsening with refinement.

Several software packages implement multilevel partitioning heuristics. These tools aim for fast, high-quality approximate solutions rather than exact optimality, reflecting the NP-hardness of the problem. METIS is one of the best-known partitioning packages and implements multilevel recursive bisection and k-way partitioning (Karypis and Kumar, 1995). It is widely used due to its combination of speed and strong partition quality on many benchmark families. At this work, the METIS will be used as a baseline for evaluating the performance of the quantum partitioning approach, which will also use the same strategies for coarsening and uncoarsening as METIS, only changing the partitioning approach from classical to quantum.

Figure 6: Example of the Multilevel Graph Bisection



Source: George Karypis 2003

2.4.3.1 Coarsening Phase

At the coarsening phase, starting from $G_0 = G$, a sequence of progressively smaller graphs $G_1, \ldots, G_m$ is constructed such that $|V_0| > |V_1| > \cdots > |V_m|$ (Karypis and Kumar, 1998a, p.363). To achieve this, the Heavy Edge Matching (HEM) heuristic is utilized. HEM computes a maximal matching by visiting the vertices of the graph in a random order. When an unmatched vertex u is visited, it is matched with an unmatched adjacent vertex v such that the weight of the edge (u, v) is the maximum among all valid incident edges. By aggressively selecting edges with the largest weights, HEM significantly reduces the total edge-weight of the coarser graph, which mathematically inherently leads to a smaller edge-cut in the partition. The computational complexity for computing the Heavy Edge Matching scales linearly with the number of edges, requiring $\mathcal{O}(|E|)$ time.

2.4.3.2 Partitioning

Once the graph is coarsed, an initial k-way partition is computed. A crucial computational advantage of the multilevel approach is that the size of this coarser graph is strictly bounded to a very small number of vertices, typically $|V_m| < 100$. Because the size of the graph partitioned at this stage is small, strictly bounded, and completely independent of

the original macroscopic graph size, the execution time required to bisect it is negligible. Consequently, the asymptotic time complexity of this specific phase is effectively constant, $\mathcal{O}(1)$, relative to the overall size of the original problem (Karypis and Kumar, 1998a, p.367).

Because this localized step is so cheap, it permits the use of more expensive methods if desired (Buluç et al., 2013, p.15,16). At this phase, a 2-way bisection of the coarsest graph is computed. Nevertheless, thanks to the strictly bounded size of the coarsest graph, this computational expense is capped, preserving the overall $\mathcal{O}(1)$ constant time complexity of the partitioning phase relative to the original macroscopic problem.

2.4.3.3 Uncoarsening

During the uncoarsening phase, the partition of the coarsest graph is iteratively projected back through the intermediate graphs to the original finer graph. Because the finer graphs have more degrees of freedom, the projected partition is typically not at a local minimum and can be further improved. To achieve this, partition refinement algorithms are applied at each uncoarsening level to swap subsets of vertices between the partitions and iteratively decrease the edge-cut. The primary refinement policy is based on the Kernighan-Lin (KL) algorithm, specifically implemented using the Fiduccia-Mattheyses (FM) variant. The time complexity of this FM-KL refinement algorithm is $\mathcal{O}(|E|)$ per pass (Karypis and Kumar, 1998a, p.368). To further minimize the practical execution time without changing the asymptotic complexity, a Boundary KL refinement strategy can be applied, which limits the data structure insertions and heuristic evaluations exclusively to the vertices located precisely on the boundary of the partition cut.

2.5 Quantum Optimization Algorithms

To apply QAOA or QA, the combinatorial objective must be mapped to a spin model. This is done by encoding the decision variables into spin variables and rewriting the optimization target as the energy of an Ising Hamiltonian. In practice, this is often expressed as a Quadratic Unconstrained Binary Optimization (QUBO) model, in which both the original objective and the constraints are embedded into a single quadratic function via penalty terms (Kochenberger et al., 2014, p.6) (Lucas, 2014). Minimizing this QUBO is then equivalent to finding the ground state of the corresponding Ising model, which can be approximately approached either variationally (QAOA) or through annealing dynamics (QA) (Farhi et al., 2014, p.2,7) (Ushijima-Mwesigwa et al., 2017, p.2).

2.5.1 The Ising Model

To connect graph partitioning to quantum optimization, start with the bisection case, $k = 2$, where each vertex is assigned to one of two blocks. We encode the assignment by spin variables

$$s_i \in \{+1, -1\}, \qquad s_i = +1 \text{ if } v_i \in V_1, \quad s_i = -1 \text{ if } v_i \in V_2.$$

Let W be the weighted adjacency matrix, i.e., $W_{ij} = \omega_{ij}$ if $\{i, j\} \in E$ and $W_{ij} = 0$ otherwise, and let the edge cut can be computed as

$$\text{cut}(s) = \sum_{\{i,j\} \in E} \omega_{ij} \frac{1 - s_i s_j}{2}. \tag{2.64}$$

Using $s_i \in \{+1, -1\}$, note that

$$(s_i - s_j)^2 = s_i^2 + s_j^2 - 2 s_i s_j = 2(1 - s_i s_j) \tag{2.65}$$

so Eq. (2.64) can be rewritten as

$$\text{cut}(s) = \frac{1}{4} \sum_{\{i,j\} \in E} \omega_{ij} (s_i - s_j)^2. \tag{2.66}$$

For bisection $k = 2$, let $s_i \in \{+1, -1\}$ indicate the block assignment of vertex i: $s_i = +1$ for V_1 and $s_i = -1$ for V_2. Let $c_i = c(v_i)$ be the vertex weight and define the total weight of each block as

$$c(V_1) = \sum_{i:\, s_i = +1} c_i, \qquad c(V_2) = \sum_{i:\, s_i = -1} c_i. \tag{2.67}$$

A perfectly balanced split requires $c(V_1) = c(V_2)$, equivalently $c(V_1) - c(V_2) = 0$. Using $s_i = \pm 1$, observe that

$$\sum_{i=1}^{n} c_i s_i = \sum_{i:\, s_i = +1} c_i - \sum_{i:\, s_i = -1} c_i = c(V_1) - c(V_2). \tag{2.68}$$

Therefore, perfect balance can be written compactly as

$$c^{\mathsf{T}} s = 0, \tag{2.69}$$

where $c^{\mathsf{T}} = (c_1, \ldots, c_n)^{\mathsf{T}}$. In the unweighted case ($c_i = 1$), this reduces to $\sum_i s_i = 0$, i.e., the two blocks contain the same number of vertices. As a standard way to remove the

explicit constraint is to add a quadratic penalty term and now grouping both the edge cut and balance constraint at its most compactly forms, we can write it as

$$\min_{s\in\{+1,-1\}^n} \left[\frac{1}{4}s^{\mathsf{T}}Ls + \eta\,(c^{\mathsf{T}}s)^2\right], \qquad \eta > 0, \tag{2.70}$$

which is an Ising-type energy and enforces balance increasingly strongly as η grows. This is the key bridge to quantum optimization: minimizing Eq. 2.70 is equivalent to finding the ground state of an Ising Hamiltonian (Lucas, 2014).

In a quantum encoding, each spin variable corresponds to the Pauli-Z operator on a qubit, whose eigenvalues are ± 1. Replacing s_i by Z_i turns the classical energy into a Hamiltonian whose ground state encodes the best partition:

$$H_{\mathrm{GP}} = \sum_{\{i,j\}\in E} \frac{\omega_{ij}}{2}\big(I - Z_iZ_j\big) \;+\; \eta\left(\sum_{i=1}^{n} c_iZ_i\right)^2, \tag{2.71}$$

up to additive constants. This Hamiltonian is directly compatible with quantum annealing and also provides the cost operator used inside QAOA.

2.5.2 Quadratic Unconstrained Binary Optimization

Many optimization solvers use binary variables $x_i \in \{0,1\}$ instead of spins $s_i \in \{+1,-1\}$. A standard change of variables is

$$s_i = 2x_i - 1, \qquad x_i = \frac{1+s_i}{2}. \tag{2.72}$$

Under this mapping, an edge $\{i,j\}$ contributes to the cut iff $x_i \neq x_j$. Indeed,

$$\frac{1-s_is_j}{2} = \frac{1-(2x_i-1)(2x_j-1)}{2} = x_i + x_j - 2x_ix_j. \tag{2.73}$$

Therefore, the weighted cut objective becomes

$$\mathrm{cut}(x) = \sum_{\{i,j\}\in E} \omega_{ij}\left(x_i + x_j - 2x_ix_j\right). \tag{2.74}$$

For balance, in the weighted case we want

$$c(V_1) = c(V_2) = \tfrac{1}{2}c(V), \qquad c(V) = \sum_{i=1}^{n} c_i. \tag{2.75}$$

Since $x_i = 1$ indicates membership in V_1, this condition is

$$\sum_{i=1}^{n} c_ix_i = \frac{1}{2}\sum_{i=1}^{n} c_i. \tag{2.76}$$

As in the spin formulation, we enforce balance via a quadratic penalty:

$$\mathrm{pen}(x) = \eta \left(\sum_{i=1}^{n} c_i x_i - \frac{1}{2} \sum_{i=1}^{n} c_i \right)^2, \qquad \eta > 0. \tag{2.77}$$

Putting the cut objective and the balance penalty together yields a QUBO model:

$$\min_{x \in \{0,1\}^n} \left[\underbrace{\sum_{\{i,j\} \in E} \omega_{ij} \left(x_i + x_j - 2 x_i x_j \right)}_{\text{linear + quadratic cut terms}} + \underbrace{\eta \left(\sum_{i=1}^{n} c_i x_i - \frac{1}{2} \sum_{i=1}^{n} c_i \right)^2}_{\text{quadratic balance penalty}} \right]. \tag{2.78}$$

Expanding the square in Eq. (2.77) shows that the full objective in Eq. (2.78) contains only constant, linear, and pairwise terms in the binary variables. For this reason, it is convenient to collect coefficients into a matrix representation,

$$f(x) = \sum_{i=1}^{n} h_i x_i \; + \sum_{1 \leqslant i < j \leqslant n} J_{ij} x_i x_j \; + \; \text{const}, \tag{2.79}$$

which is the standard QUBO form used by many solvers. This objective is a quadratic polynomial in binary variables and can be written compactly as

$$x^{\mathsf{T}} Q x + q^{\mathsf{T}} x + \text{const}, \tag{2.80}$$

for a suitable matrix Q and vector q (Kochenberger et al., 2014, p.7). For $k > 2$, a standard encoding introduces one-hot variables $x_{i,r} \in \{0, 1\}$ indicating whether vertex i is assigned to block $r \in \{1, \ldots, k\}$. The intended constraint is

$$\sum_{r=1}^{k} x_{i,r} = 1 \qquad \text{for all } i \in \{1, \ldots, n\}, \tag{2.81}$$

so each vertex belongs to exactly one block.

With this encoding, two vertices i and j are in the *same* block if there exists an r such that $x_{i,r} x_{j,r} = 1$. Hence the indicator for being in the same block is $\sum_{r=1}^{k} x_{i,r} x_{j,r}$, and the indicator for being in *different* blocks is $1 - \sum_{r=1}^{k} x_{i,r} x_{j,r}$. This gives the multiway cut objective

$$\mathrm{cut}(x) = \sum_{\{i,j\} \in E} \omega_{ij} \Big(1 - \sum_{r=1}^{k} x_{i,r} x_{j,r} \Big). \tag{2.82}$$

A balanced k-way partition targets equal total weight in each block,

$$\sum_{i=1}^{n} c_i x_{i,r} \approx \frac{c(V)}{k}, \qquad r = 1, \ldots, k, \tag{2.83}$$

which is enforced via a quadratic penalty. Likewise, the one-hot constraint Eq. (2.81) is enforced by a quadratic penalty. Collecting all terms yields a single unconstrained QUBO-style objective:

$$\min_{x \in \{0,1\}^{n \times k}} \Bigg[\underbrace{\sum_{\{i,j\} \in E} \omega_{ij} \Big(1 - \sum_{r=1}^{k} x_{i,r} x_{j,r}\Big)}_{\text{cut objective (penalizes edges crossing blocks)}} + \underbrace{\eta \sum_{r=1}^{k} \Big(\sum_{i=1}^{n} c_i x_{i,r} - \tfrac{c(V)}{k} \Big)^2}_{\text{balance penalty (equal block weights)}} + \underbrace{A \sum_{i=1}^{n} \Big(\sum_{r=1}^{k} x_{i,r} - 1 \Big)^2}_{\text{one-hot penalty (one block per vertex)}} \Bigg], \tag{2.84}$$

where $A > 0$ and $\eta > 0$ are penalty parameters. This formulation provides the standard interface for applying QA/QAOA-style solvers to k-way graph partitioning (Lucas, 2014).

2.5.3 The Adiabatic Theorem

Consider the evolution of a state been governed by the Schrödinger equation

$$i \frac{d}{dt} |\psi(t)\rangle = H(t) |\psi(t)\rangle, \tag{2.85}$$

and a smooth family of Hamiltonians $H(s)$ for $s \in [0, 1]$,

$$H(t) = \tilde{H}(t/T), \qquad t \in [0, T] \tag{2.86}$$

so that the total runtime T controls how slowly the Hamiltonian changes. Kato states in (Kato, 1950) that in the limit when the change $H(t)$ is made infinitely slow, starting from a stationary state $H(0)$, the system will pass through the corresponding stationary states of $H(t)$ for all t, this is what the adiabatic theorem for quantum mechanics states. This transformation will lead the state to reach the ground state and as the Ising formulation derived previously turns graph partitioning into a ground-state search problem, finding the ground state for the system is equivalent to finding the solution for the partitioning problem. In particular, the problem Hamiltonian defined in Eq. 2.71 assigns an energy to each spin configuration, and its ground state encodes the optimal partition under the chosen objective and penalties. Thus, in order to find the ground state for the Hamiltonian, it is necessary to apply an adiabatic transformation at the initial state

Hamiltonian to reach the final state as the ground state. For a Hermitian Hamiltonian H, the eigenvalue problem reads

$$H\,|\psi_\ell\rangle = E_\ell\,|\psi_\ell\rangle\,, \tag{2.87}$$

and let instantaneous eigenpairs satisfy

$$H(s)\,|\psi_\ell(s)\rangle = E_\ell(s)\,|\psi_\ell(s)\rangle\,. \tag{2.88}$$

with ordered eigenvalues

$$E_0(s) \leqslant E_1(s) \leqslant \cdots \leqslant E_{N-1}(s). \tag{2.89}$$

The ground state $|\psi_0\rangle$ corresponds to the minimum energy E_0. When H is constructed from a combinatorial cost function, minimizing the objective is equivalent to preparing the ground state. According to the adiabatic theorem (Kato, 1950), if the spectral gap between the two lowest energy levels, $E_1(s) - E_0(s)$, is strictly greater than zero for all dimensionless annealing parameters $s = t/T \in [0, 1]$, then the system will remain in the ground state as the total annealing time T approaches infinity

$$\lim_{T\to\infty} |\langle\psi_0(T) \mid \psi(T)\rangle|^2 = 1. \tag{2.90}$$

Regarding this energy difference, we define the minimum spectral gap between the ground and first excited levels throughout the evolution as

$$g_{\min} = \min_{s\in[0,1]} \big(E_1(s) - E_0(s)\big). \tag{2.91}$$

In algorithmic applications, the runtime required to remain close to the ground state is directly governed by this gap. Specifically, the standard quantitative adiabatic condition yields a scaling limit of the form

$$T \gg \frac{\max_s \left|\langle\psi_1(s)|\,\frac{dH(s)}{ds}\,|\psi_0(s)\rangle\right|}{g_{\min}^2}. \tag{2.92}$$

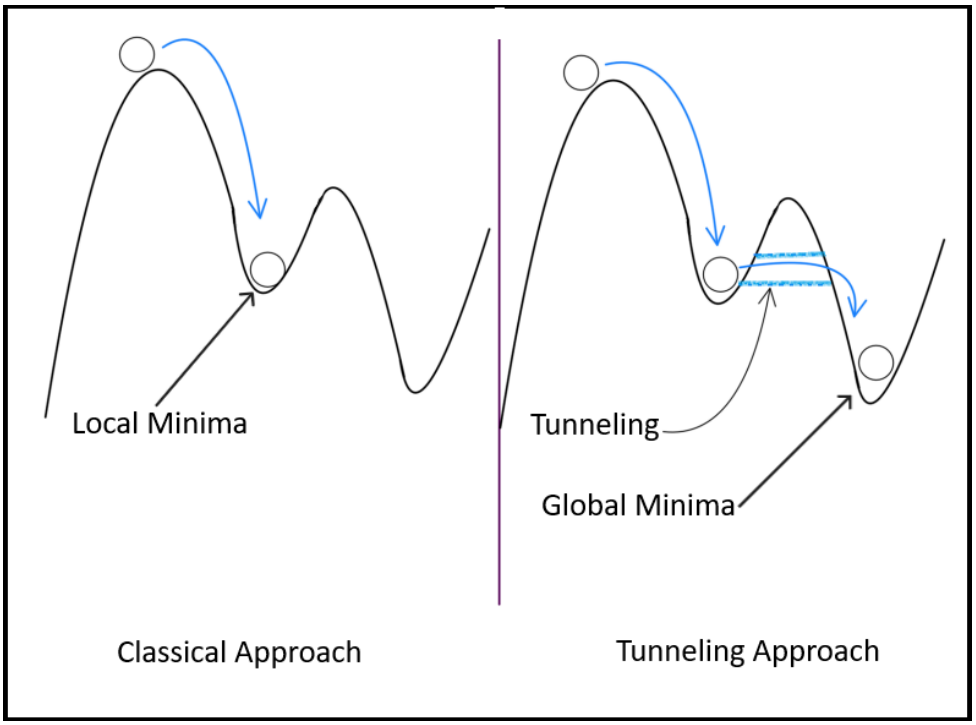
This highlights why adiabatic approaches can be highly efficient for some structured instances yet become impractical when $g_{\min}$ shrinks rapidly with problem size.

2.5.4 Quantum annealing

Quantum annealing (QA) focuses on finding the optimal solution to complex combinatorial optimization problems by leveraging the principles of quantum mechanics. To achieve these computational advantages, (Tripathi et al., 2025, p.13) introduces QA

as a method to bypass the energy barriers that typically trap classical methods in local minima. Instead of having to sequentially climb the energy landscape, QA allows the wave function of a quantum particle to pass directly through potential energy barriers as illustrated in Figure 7. This capability allows the system to explore a larger portion of the energy landscape in a shorter amount of time, enabling faster convergence to the global minimum, which represents the optimal solution for the combinatorial system.

Figure 7: Quantum tunneling overview



Source: Tripathi 2024

The underlying physics of QA is governed by a time-dependent Hamiltonian based on the adiabatic theorem. Let us consider an optimization problem encoded into a generalized Ising model denoted by the problem Hamiltonian H_P. This classical many-body Ising Hamiltonian includes linear biases and quadratic coupling weights, described in terms of the z components of the Pauli operator σ_j^z

$$H_P = \sum_i h_i \sigma_i^z + \sum_{i<j} J_{ij} \sigma_i^z \sigma_j^z. \tag{2.93}$$

The process introduces a driver Hamiltonian H_D, or $H_{initial}$, which does not commute with H_P and possesses a known, trivial ground state. A standard choice for H_D is the transverse field:

$$H_D = -\sum_j \sigma_j^x. \tag{2.94}$$

The total Hamiltonian of the QA process interpolates between these two states. This evolution is frequently represented using a normalized, dimensionless time parameter $s = t/T$ to transition from quantum exploration to classical exploitation

$$H(s) = (1 - s)H_D + sH_P. \tag{2.95}$$

The system is initially set at the ground state of the driver Hamiltonian at $s = 0$. If the change in $H(s)$ with respect to time is sufficiently small, the spin state evolves adiabatically. It stays in the instantaneous ground state $|\psi_0(s)\rangle$ and successfully arrives at the desired ground state of H_P at $s = 1$.

The success of QA is strictly determined by how slowly the Hamiltonian changes, which directly establishes its computational complexity. The spatial memory complexity on a quantum annealer scales linearly, requiring $\mathcal{O}(N)$ physical qubits to represent the variables of the problem, alongside any auxiliary physical qubits required to map the problem graph onto the fixed hardware topology. The theoretical time complexity is dictated by the adiabatic condition. The runtime T required to strictly remain in the ground state is governed by the minimum spectral gap $g_{\min}$ between the ground state $|\psi_0(s)\rangle$ and the first excited state $|\psi_1(s)\rangle$, explicitly formulated in Equation 2.92. When physical quantum hardware is restricted or unavailable, this physical evolution can be approximated on classical computers using heuristic techniques known as emulated quantum annealing.

2.5.4.1 Emulated Quantum Annealing

At this work, it is used the emulated version of the quantum annealing algorithm from the D-Wave library called Anneal. This library utilizes the approach from (Kirkpatrick et al., 1983) where the physical annealing process is numerically simulated using the Metropolis Monte Carlo algorithm. In this approach, the objective cost function of the combinatorial optimization problem is directly mapped to the energy E of a physical many-body system. The emulation proceeds in small steps by generating random rearrangements from one configuration to the next and evaluating the resulting change in the cost function, ΔE.

If a proposed move improves the overall solution ($\Delta E \leqslant 0$), it is automatically accepted. However, to prevent the algorithm from getting trapped in local optima, a common flaw of strict iterative improvement methods, moves that worsen the solution ($\Delta E > 0$) are also accepted probabilistically. The probability of accepting an uphill move is given by the Boltzmann factor $P(\Delta E) = \exp(-\Delta E/k_B T)$, where T acts as the effective temperature control parameter.

The simulated annealing process operates by first "melting" the system at a high effective temperature, allowing it to freely accept uphill moves and explore the global solution space. The temperature T is then gradually lowered according to a defined

cooling schedule until the system "freezes" and no further configuration changes occur. This mechanism acts as an adaptive divide and conquer strategy: gross features of the optimal state emerge at high temperatures, while fine-tuning occurs at lower temperatures. Crucially, empirical analysis has shown that to reach high-quality solutions, the average computational effort of this emulation scales highly efficiently, typically linearly as $\mathcal{O}(N)$ or as a small power of N.

2.5.5 Quantum Approximate Optimization Algorithm

As discussed in the previous section on quantum annealing, finding the exact ground state requires continuous, infinitely slow adiabatic evolution. The Quantum Approximate Optimization Algorithm (QAOA) serves as a algorithm directly inspired by this adiabatic process. Rather than strictly maintaining the adiabatic condition to guarantee a perfect solution, QAOA discretizes the continuous time evolution into a parameterized quantum circuit. This allows the algorithm to run significantly faster, trading absolute optimality for a high-quality approximate solution. It is, as stated in (Farhi et al., 2014, p.2), a quantum gate-model meta-heuristic which alternates between applying unitaries drawn from two families: a cost function based unitary family $U_p(\gamma) = e^{-i\gamma H_f}$ and a family of mixing unitaries $U_m(\beta) = e^{-i\beta H_\beta}$, for some fixed cost function based Hamiltonian H_f and some fixed mixing Hamiltonian H_β.

To formalize the QAOA for an unconstrained optimization problem, as in Equation 2.84, on n-bit strings, the algorithm is specified by two Hamiltonians: the phase Hamiltonian H_P encoding the cost function f and the mixing Hamiltonian H_M which is the transverse field Hamiltonian (Hadfield et al., 2019, p.4), along with $2p$ real parameters $\gamma_1, \ldots, \gamma_p$ and $\beta_1, \ldots, \beta_p$.

The phase Hamiltonian H_P encodes the cost function f to be optimized (Hadfield et al., 2019, p.4). It acts diagonally on n-qubit computational basis states such that

$$H_P|y\rangle = f(y)|y\rangle. \tag{2.96}$$

The standard mixing Hamiltonian H_M is the transverse field Hamiltonian

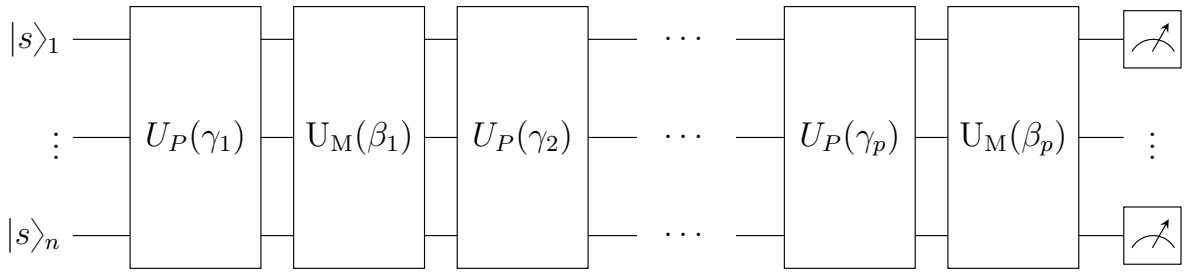
$$H_M = \sum_{j=1}^{n} X_j, \tag{2.97}$$

where X_j is the Pauli X operator acting on the j-th qubit. The Pauli X operator acts as a bit flip, meaning $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$. The initial state $|s\rangle$ is selected to be the equal superposition state of all possible solutions

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_x |x\rangle, \tag{2.98}$$

and this state is also the ground-state of $-H_M$. A parameterized quantum state is created by alternately applying the Hamiltonians H_P and H_M for p rounds as in figure 8.

Figure 8: Circuit overview for the QAOA



Source Hadfield 2019

The duration in round j is specified by the parameters γ_j and β_j, respectively, resulting in the state

$$|\beta, \gamma\rangle = e^{-i\gamma_1 H_P} e^{-i\beta_1 H_M} \dots e^{-i\gamma_j H_P} e^{-i\beta_j H_M} |s\rangle. \tag{2.99}$$

A computational basis measurement is performed on the state, which returns a candidate solution y with probability $|\langle y|\beta, \gamma\rangle|^2$. By repeating the state preparation and measurement, the expected value of the cost function over the returned solution samples can be evaluated as

$$\langle f \rangle = \langle \beta, \gamma | H_P | \beta, \gamma \rangle, \tag{2.100}$$

this value can be statistically estimated from the produced samples. The above steps are repeated iteratively, with updated sets of time parameters, as part of a classical optimization loop. This loop is used to optimize the algorithm parameters with respect to an objective such as $\langle f \rangle$ (Hadfield et al., 2019, p.5). The best problem solution found overall is returned. A critical factor for the success of QAOA is the selection or discovery of good values for the parameters $\gamma_1, \dots, \gamma_p$ and $\beta_1, \dots, \beta_p$, which are necessary to yield high-quality approximate solutions.

Regarding the computational cost, QAOA presents specific scaling characteristics that distinguish it from classical solvers. The spatial memory complexity on quantum hardware scales linearly as $\mathcal{O}(n)$ qubits, directly matching the number of variables in the target problem (Farhi et al., 2014, p.2). The quantum time complexity for executing a single shot of the circuit is determined by the number of algorithmic rounds p and the density of the problem graph. Applying the phase Hamiltonian requires executing discrete quantum gates for every quadratic interaction, resulting in a quantum gate time complexity of $\mathcal{O}(p \cdot |E|)$ for every shot, where $|E|$ (edges) is the number of non-zero coupled pairs in the QUBO formulation. The total time to solution then can be written as

$$\mathcal{O}(p \cdot |E| \cdot M \cdot K), \tag{2.101}$$

which represents this single-shot gate complexity $p \cdot |E|$ multiplied by the number of measurements M taken per classical iteration and the total number of classical parameter updates K (Cerezo et al., 2021, p.2) required to converge.

2.6 Quantum Annealing Hardware

At this thesis, were utilized both emulated and physical Annealers when executing the partitioning step within the quantum version of the pipeline. The key differences between physical and emulated Annealers rely mostly at the decoherence present at the physical hardware and error rates regarding the number of qubit's utilized.

The concrete experimental platforms used later in the thesis are documented with the simulation setup in Table 13. In summary, the quantum annealing experiments use the D-Wave Ocean/Leap software path, while the gate-based QAOA hardware path is accessed through a QLM backend.

2.6.1 Physical Annealers

At the core of physical quantum annealers, such as those developed by D-Wave Systems, are arrays of superconducting flux qubits (Johnson et al., 2011) and (King et al., 2022, p.1). These qubits are realized using loops of superconducting material, typically niobium or aluminum, interrupted by Josephson junctions. This design allows macroscopic persistent currents to flow in clockwise or counterclockwise directions, acting as synthetic spins to represent binary quantum states. To maintain these delicate quantum states and suppress environmental thermal noise, the processors must operate in extreme cryogenic environments, typically housed in dilution refrigerators cooled to approximately 15 millikelvin.

The primary advantage of this physical architecture is its potential to achieve significant computational speedups when solving large, highly complex optimization problems. Unlike classical simulated annealing, which relies on thermal fluctuations to iteratively climb over energy barriers, physical quantum annealers exploit the phenomenon of quantum tunneling to directly pass through tall, narrow barriers in rugged energy landscapes (Boixo et al., 2016, Crosson and Deng, 2014). This inherent quantum mechanical property dramatically increases the probability of escaping local minima, offering a faster and more efficient path to finding global optimal solutions in massive, real-world scenarios.

However, the practical realization of physical annealing is heavily constrained by decoherence and inherent hardware error rates (Gardas et al., 2018, p.5). Because the superconducting circuits are macroscopic, they are highly susceptible to environmental interactions, leading to rapid decoherence and extremely short coherence times, often

limited to the 5 to 50 nanosecond range. The impact of this environmental noise on the energy-relaxation time (T_1) of the system can be mathematically modeled using Fermi's golden rule

$$\frac{1}{T_1} = \sum_{\lambda} \frac{2|\langle e|\hat{D}_\lambda|g\rangle|^2}{\hbar^2} S_\lambda(\omega) \tag{2.102}$$

where $S_\lambda(\omega)$ represents the power spectral density of the specific noise source λ such as low-frequency $1/f$ flux noise, and $\hat{D}_\lambda$ is the transition dipole moment that couples the environment to the ground $|g\rangle$ and excited $|e\rangle$ states of the coupled system (Weber et al., 2017, p.9).

Furthermore, these devices suffer from Integrated Control Errors (ICE), which include low-frequency flux noise and finite precision limits in the digital-to-analog converters (DACs) used to program the qubits. Because of these analog imperfections, the Quantum Processing Unit (QPU) does not solve the exact, ideal Ising objective. Instead, it solves an altered energy function where the intended programmable biases h_i and coupling strengths $J_{i,j}$ are distorted by time- and state-dependent error components δh_i and $\delta J_{i,j}$:

$$E_{\text{altered}}(s) = \sum_{i=1}^{N} (h_i + \delta h_i) s_i + \sum_{i=1}^{N} \sum_{j>i}^{N} (J_{i,j} + \delta J_{i,j}) s_i s_j \tag{2.103}$$

These δh_i and $\delta J_{i,j}$ perturbations fluctuate based on background susceptibility, local flux noise, and hardware heterogeneity, shifting the energy landscape and directly degrading the probability of successfully localizing the true global minimum (D-Wave Systems Inc., 2024). Finally, physical annealers are restricted by sparse hardware connectivity topologies. This physical limitation requires complex minor embedding techniques that consume multiple physical qubits to represent a single logical variable, complicating the energy landscape and limiting the raw scale of problems that can be natively solved (Pelofske, 2025, p.3).

2.6.2 Emulated Annealers

Given the strict cryogenic infrastructure requirements, high operational costs, and limited availability associated with physical quantum hardware, this work also extensively evaluates optimization problems using emulated annealers. Emulated annealers are quantum-inspired systems that execute customized simulated annealing algorithms on classical CMOS hardware, such as modern CPUs, Graphic Processor Unit (GPUs), or specialized FPGAs (Aramon et al., 2019, p.1-2).

A major advantage of using emulated annealers is that they make algorithmic implementation and research significantly easier by operating stably at room temperature

and bypassing the physical wiring limits of superconducting chips. Because they utilize classical memory arrays to represent interaction weights, digital annealers offer fully connected architectures (Aramon et al., 2019, p.4). This allows researchers to directly map dense, large-scale combinatorial problems containing thousands of variables without the computational overhead or hardware loss associated with minor embedding.

Despite these practical research benefits, emulated annealers possess fundamental disadvantages, primarily because they do not exhibit real quantum behavior. Instead of genuine quantum tunneling, they rely on classical stochastic heuristics or deterministic differential equations to mimic quantum barrier-crossing effects (Kirkpatrick et al., 1983). Consequently, they may fail to capture the complex, multi-dimensional tunneling paths available to a physical quantum state, meaning they can still get trapped in local minima that a true quantum system would easily navigate. Furthermore, attempting to fully simulate exact quantum mechanical state vectors on classical hardware encounters an exponential "memory wall". Because representing a true quantum superposition of N qubits requires storing 2^N complex amplitudes, classical memory limits are quickly exhausted (Georgescu et al., 2014, p.2). This strictly limits exact quantum emulation to very small systems and forces large-scale emulators to rely purely on classical approximations of the Ising model rather than full quantum interference.

2.6.3 Hybrid Solutions

Despite the continuous advancements in pure quantum algorithms and hardware, current quantum processing units (QPUs) often face strict limitations when dealing with excessively complex or large-scale optimization problems. These hardware constraints primarily manifest as limited qubit counts, restricted connectivity, and vulnerability to noise (Tripathi et al., 2025, p.26). Because modern QPUs possess a sparse physical topology, mapping a densely connected problem requires minor embedding, a process that utilizes chains of multiple physical qubits to represent a single logical variable (Choi, 2011, p.2,3). This embedding overhead not only complicates the energy landscape but also drastically reduces the machine's effective capacity, limiting the raw scale of problems that can be natively solved (Pelofske, 2025, p.4).

To overcome these physical barriers, quantum-classical hybridization has emerged as a highly practical and scalable solution (Bass et al., 2021, p.1-2). Hybrid solvers are computational systems designed to maximize the complementary strengths of both classical and quantum resources. By integrating classical algorithms with quantum computing, these solvers leverage classical preprocessing to simplify or break down massive problem instances (Tripathi et al., 2025, p.26). Consequently, they can dispatch only the most computationally intensive subproblems to the QPU, where quantum tunneling is

utilized to efficiently explore complex regions of the solution space (Quinton et al., 2025, p.5).

In the context of the D-Wave ecosystem, this hybrid optimization is primarily managed through the Ocean software suite (D-Wave Systems Inc., 2023) and its Hybrid Solver Service (HSS) (D-Wave Systems Inc., 2020, p.1). A foundational approach within this suite is the Hybrid Workflow Solver, which processes problems across multiple resolution flows or branches. This architecture typically involves three core components: a Decomposer, which is an algorithm that breaks down the complete problem into smaller, solvable subproblems based on specific rules; a SubSampler, consisting of resolution algorithms (which can be either classical or quantum) designed to solve the generated subproblems; and a Composer, an algorithm that dictates how to integrate the subproblem results back into the complete collection of variables, often requiring a merging rule if multiple branches are present (Booth et al., 2017, p.1).

This decomposer-sampler-composer workflow directly parallels the multilevel scheme traditionally used for solving massive combinatorial problems, such as graph partitioning (Karypis and Kumar, 1998a). Within a hybrid quantum-classical framework, the classical machine acts as the Decomposer by executing the coarsening phase, systematically reducing the graph's degrees of freedom to approximate the original problem on a smaller, manageable scale. Once the problem is sufficiently condensed, the quantum machine steps in as the SubSampler to perform the initial partitioning. At this stage, the QPU processes the highly concentrated, reduced-size matrices, successfully bypassing the minor embedding bottlenecks of the original macroscopic graph and leveraging quantum mechanics to find optimal configurations. Following this, the classical machine takes over again as the Composer, handling the uncoarsening phase by projecting the quantum-derived sub-solutions back onto the original problem space and performing local refinements to construct the final global solution.

Ultimately, the primary motivation for adopting these hybrid solvers is scalability (Quinton et al., 2025, p.5). By allowing classical computers to manage the bulk of the problem's structure and data, hybrid solvers elegantly bypass the strict topological limitations of the QPU (Booth et al., 2017, p.8). This synergy enables the tackling of larger and more intricate combinatorial problems than would be natively possible on current standalone QPUs, particularly those involving strict operational constraints (Bass et al., 2021, p.17). While quantum machines represent a promising frontier for mathematical optimization, their current computational strength emerges most prominently when synergistically integrated with classical computers through these hybrid solver architectures.

3 STATE OF THE ART

This chapter positions the thesis within the current research landscape of domain decomposition, graph partitioning, sparse matrix reordering, and quantum optimization. The goal is not to repeat the mathematical background from Chapter 2, but to identify the most relevant state-of-the-art approaches, their limitations, and the gap addressed by the proposed hybrid quantum-classical workflow.

3.1 Domain Decomposition and Sparse Solvers

Domain decomposition methods are among the most established strategies for solving large systems arising from partial differential equations and finite element discretizations. Their central idea is to split the physical or algebraic problem into smaller subproblems, solve or precondition these subproblems locally, and exchange information through interfaces. Classical references describe this family of methods as a bridge between discretization, parallel computing, and iterative linear solvers (Smith et al., 1996, Toselli and Widlund, 2005). In engineering applications, the appeal of this approach is direct: large finite element models can exceed the memory and time limits of monolithic solvers, whereas decomposed models can exploit locality, parallelism, and more compact preconditioners.

The state of the art in domain decomposition is broad. Non-overlapping methods, overlapping Schwarz methods, FETI-type formulations, and related preconditioners have all been investigated for elliptic and structural problems (Champaney et al., 1997, p.254), (Gropp and Keyes, 1989, Su and Fulton, 1993). These methods are mature and mathematically well understood, but their effectiveness depends strongly on the quality of the decomposition. If subdomains are poorly balanced, if interfaces are unnecessarily large, or if the induced matrix ordering destroys locality, then the solver may suffer from excessive communication, weak preconditioners, or high memory usage. Therefore, even when the linear algebra method itself is classical, the partitioning and ordering steps remain critical.

Sparse direct and preconditioned iterative solvers also motivate matrix reordering. Incomplete factorizations, block preconditioners, and localized domain-decomposition preconditioners are sensitive to the distribution of nonzero entries around the diagonal. The classical theory of domain decomposition preconditioners shows that solver performance is tied to both algebraic conditioning and the geometry of the subproblems (Bramble, 1986, p.415, 428), (Toselli and Widlund, 2005) . Recent work on preconditioner-oriented matrix reordering reinforces this point by connecting structural concentration, incomplete factorization quality, and Krylov convergence. This thesis follows the same

direction: it does not claim that a permutation alone changes the spectrum of the original matrix, but investigates whether graph-based permutations can create structures that are more favorable for building preconditioners.

3.2 Graph Partitioning for Domain Decomposition

Graph partitioning is the dominant abstraction used to transform a mesh, sparse matrix, or finite element connectivity pattern into a decomposition problem. The matrix or mesh is represented as a weighted graph, where vertices represent unknowns or mesh nodes and edges represent coupling. A good partition should balance vertex weights while minimizing edge cuts. This formulation aligns naturally with domain decomposition: balanced parts avoid overloaded subdomains, while small cuts reduce interfaces and potential communication.

The most influential classical solution family is multilevel graph partitioning. In this approach, the graph is repeatedly coarsened, partitioned at a smaller scale, and then uncoarsened with local refinement. METIS is one of the best-known implementations of this paradigm and remains a standard baseline because of its robustness and speed on large sparse graphs (Karypis and Kumar, 1995, 1998a,b). The broader survey literature confirms that multilevel partitioning remains central for scientific computing, sparse matrices, and parallel workloads (Buluç et al., 2013, p.2,15). More recent packages such as KaHIP extend the design space with advanced local search and balance strategies (Sanders and Schulz, nd, p.4), but the core multilevel idea remains the main reference point.

Despite their maturity, classical partitioners have limitations that motivate further investigation. They are heuristic methods, so they do not generally guarantee global optimality. Their objective functions are also simplified proxies for downstream solver performance: minimizing an edge cut does not automatically minimize matrix bandwidth, improve incomplete factorization quality, or reduce Krylov iteration counts. In addition, different graph weights, coarsening rules, balance tolerances, and refinement objectives can lead to different matrix profiles. This creates a research opportunity for alternative partitioning backends that operate inside the same multilevel framework and can be compared under identical post-processing metrics.

3.3 Matrix Reordering and Preconditioning

Matrix reordering is a classical preprocessing step for sparse linear algebra. A symmetric permutation $A_p = PAP^T$ preserves the eigenvalues of A, but changes where nonzero entries appear in the matrix. This distinction is essential. Reordering cannot directly reduce the original condition number, but it can change the structural environment in

which a preconditioner or factorization is built. Bandwidth, average distance from the diagonal, and the fraction of nonzeros inside diagonal bands are therefore useful structural indicators.

The state-of-the-art interpretation of such metrics is nuanced. A lower maximum bandwidth can reduce the storage and operation count of banded or localized factorizations. A higher local concentration of nonzeros can help incomplete factorizations retain important entries while dropping remote or weak couplings. However, no single metric fully predicts solver behavior. A permutation can reduce the maximum bandwidth while increasing the average distance of nonzeros, or it can improve very local concentration while worsening larger-band concentration. This is why modern studies typically combine structural metrics with direct solver experiments when possible (Kumlien, 2024).

The present thesis contributes to this area by evaluating matrix reordering as the final output of a hybrid graph-partitioning pipeline. Instead of treating the partition only as a domain-decomposition object, the work evaluates its induced permutation on both stiffness and mass matrices. This is important for finite element workflows because the same degree-of-freedom ordering is usually applied consistently to several system matrices. The Results chapter therefore reports matrix-specific behavior for both K and M.

3.4 Quantum Optimization for Graph Partitioning

Quantum and quantum-inspired optimization have become active research directions for combinatorial problems. Many graph partitioning objectives can be encoded as QUBO or Ising models, which makes them compatible with quantum annealing, simulated annealing, and variational gate-based algorithms (Farhi et al., 2014, p.14). Quantum annealing was originally proposed as an optimization paradigm based on adiabatic evolution and tunneling through energy barriers (Albash and Lidar, 2018, p.44). Hardware systems such as D-Wave quantum annealers implement programmable Ising-like models using superconducting qubits (Johnson et al., 2011) (King et al., 2022, p.2).

QAOA offers a different route. It is a hybrid variational algorithm in which a parameterized quantum circuit alternates cost and mixer operations, while a classical optimizer updates the circuit parameters (Farhi et al., 2014, p.14). For graph partitioning, QAOA can use the same balanced-cut QUBO or Ising formulation as the annealing approach. Its attraction is conceptual flexibility: the same cost Hamiltonian can be evaluated by a simulator, a remote quantum processor, or future gate-based hardware with improved scale and connectivity.

Current quantum optimization hardware remains constrained. Quantum annealers have sparse hardware connectivity, finite precision, analog control errors, and embedding overheads (D-Wave Systems Inc., 2024) (Pelofske, 2025, p.4). Dense QUBO

models may require clique embeddings in which one logical variable consumes multiple physical qubits. Gate-based QPUs face limited qubit counts, noise, queue times, and sampling overhead. Emulated annealers and local QAOA simulations avoid hardware noise and scheduling problems, but they do not provide evidence of quantum advantage and can become expensive as the problem size increases (Aramon et al., 2019, p.1-2).

For these reasons, the state-of-the-art direction is not to send the original full-scale finite element graph directly to a QPU. Instead, hybrid approaches combine classical reduction with quantum or quantum-inspired solution of a smaller subproblem (Bass et al., 2021, p. 8), (Quinton et al., 2025, p.5). This thesis follows that logic. Coarsening reduces the graph to a hardware-feasible size, the quantum or classical backend solves the coarse partitioning problem, and uncoarsening projects the result back to the original matrix scale.

3.5 Platform Limitations and Research Gap

The D-Wave Ocean ecosystem and related hybrid services provide practical access to quantum annealing and quantum-inspired solvers (D-Wave Systems Inc., 2020, p.1,2,6) (D-Wave Systems Inc., 2023). These platforms make real hardware experimentation possible, but they also expose the practical limitations of current quantum computing: account-dependent solver availability, remote submission latency, queueing, embedding limits, noisy objectives, and occasional scheduler-side failures. QLM-based execution environments present similar issues for gate-based QAOA workflows, especially when many remote objective evaluations are required during classical optimization.

The research gap addressed by this thesis lies at the intersection of these fields. Classical multilevel partitioners are fast, robust, and difficult to outperform in general. Quantum methods are promising but constrained by hardware scale and reliability. Sparse solver performance depends not only on graph cuts, but also on the final matrix profile seen by preconditioners. A serious evaluation therefore requires a complete pipeline that connects graph construction, coarsening, backend partitioning, uncoarsening, matrix permutation, and structural profiling.

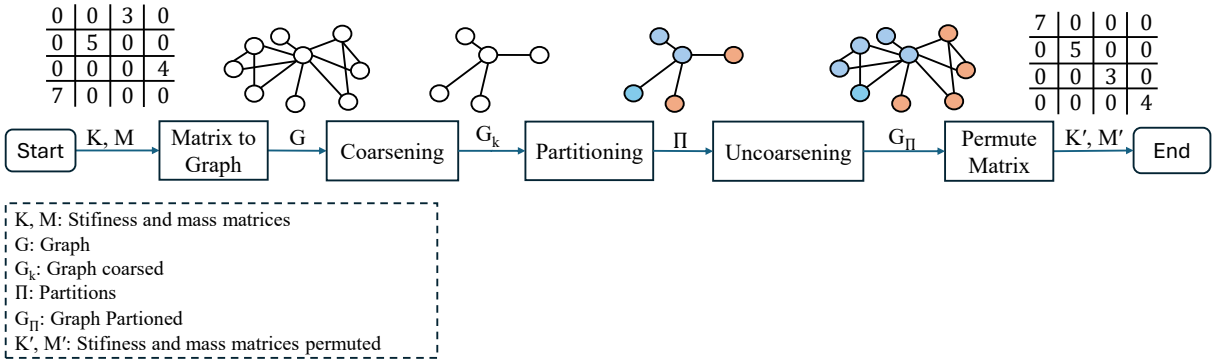
This thesis contributes such an integration study. It compares METIS, hardware quantum annealing, emulated quantum annealing, and emulated QAOA inside the same multilevel domain-decomposition workflow. The emphasis is not on claiming quantum advantage, but on establishing a reproducible environment in which quantum and quantum-inspired partitioning methods can be tested under realistic finite element matrix conditions and compared against a strong classical baseline.

4 PIPELINE ARCHITECTURE

This chapter provides a comprehensive overview of the pipeline architecture. Here will be depicted what are the possible outputs that the user can obtain while using the software, what is necessary to use it and an overview of each module responsibility. The complete pipeline, illustrating all modules integrated into the workflow, is depicted in Figure 9.

4.1 Domain decomposition pipeline

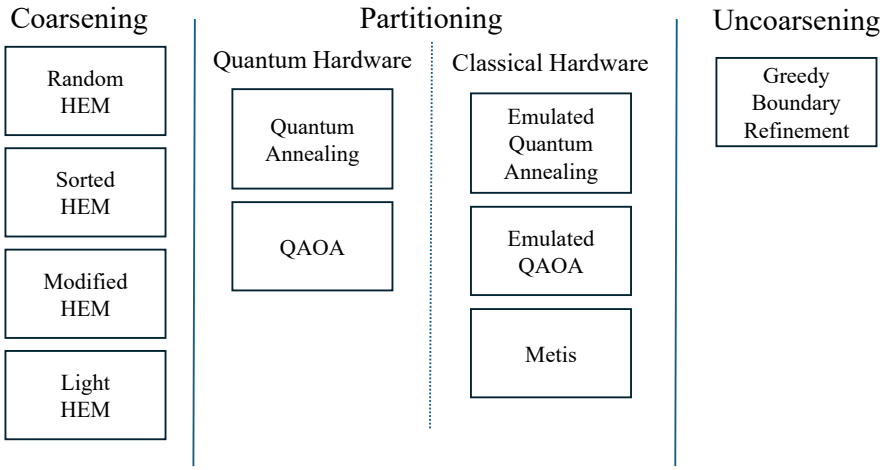
Figure 9: Domain decomposition architecture



Source: Author

The first step of the pipeline is the conversion from stiffness (K) and mass (M) matrices to the graph G and it acts just as a one to one connection between the mesh and the graph. Them, to start the multilevel partitioning scheme, the coarsening is applied at the graph G, reducing the number of nodes and edges of the graph and finishing with G_k. With the coarsed graph G_k, Π partitions will be created at the partitioning. Following with the partitions Π, the uncoarsening, as the last step of the multilevel scheme, will be performed so the graph G_k can be expanded to its original number of nodes and edges, but this time with the nodes partitioned into Π sub-graphs. And as the final step, the partitioned graph G_Π with its original number of nodes and edges will them be used to permute the lines of the initial K and M matrices, finishing with K' and M'. For the multilevel scheme (coarsening, partitioning, and uncoarsening), this work utilizes several different algorithms, as shown in Figure 10. Because the main focus of this research is investigating the partitioning step, the state-of-the-art algorithms were implemented for the coarsening and uncoarsening phases but not so deeply investigated. For the partitioning step itself, the user can choose from three different strategies in the *simulation_parameters.json* file before starting the simulation, each of these strategies will be detailed later in Chapter 6.

Figure 10: Multilevel Scheme Algorithms. Where HEM stands for Heavy Edge Matching and QAOA for Quantum Approximation Optimization Algorithm



Source: Author

4.2 Input Data and Initialization

Before the modules can operate, the pipeline requires specific entry data and configuration. The inputs for this initial step are the stiffness K and the mass M matrices and the configuration files. The matrices are the result from FEM and their construction do not belong to the scope of this work. At the pipeline developed at this thesis, they are inserted as a *.mtx* file and will be used as input to be mapped into a graph. And the configuration files are inserted in a format called *.json*.

Parameters and Configuration files To start the domain decomposition pipeline, it is necessary to insert the simulation configuration and parameters at *simulation_config.json* and *simulation_parameters.json* respectively. The configuration json file will have the main setup variables for the simulation such as the parameters file path, the matrices file path and what are the output files desired to be saved throughout the pipeline.

The parameter json file will then be responsible to store all the parameters necessary to run the multilevel scheme. The general parameters of the simulation are described at the table 2. And the available steps of the pipeline are coarsening, partitioning and uncoarsening and their specific parameters will be later depicted at chapters 5, 6 and 7 respectively.

Table 2: General simulation parameters

Description	Default Value
dof_per_node Integer defining the degrees of freedom per node, 3 for 3 dimensions problems.	3
seed Integer setting the random seed for reproducibility.	42

Source: Author

4.3 Output Data

At the configuration json file, it is possible to insert what are the files desired to be saved throughout the pipeline is been completed. These files are the results of each of the modules and the possibilities are described in table 3

Table 3: Output data

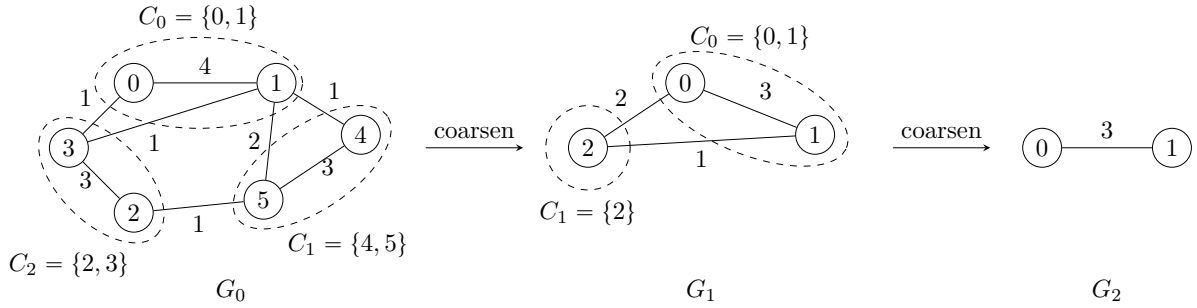
Description
coarsened_graph Final coarse graph exported as a Matrix Market file.
permutation Permutation of the matrices.
run_metrics Matrix profiling metrics, including bandwidth, Avg. Bandwidth, and diagonal-band concentration values.
run_simulation_parameters File storing the simulation parameters for that run
matrix_k_permuted Stiffness K matrix permuted with the permutation resulted from the multilevel scheme.
matrix_m_permuted Mass M matrix permuted with the permutation resulted from the multilevel scheme.

Source: Author

5 COARSENING

At this chapter will be discussed the implementation of the graph coarsening at section 5.1 and the parameters at section 5.2. The first step of the multilevel partitioning scheme is the coarsening. This step is responsible to reduce the dimension of the graph using matching strategies such as Heavy Edge Matching (HEM). At this step the initial graph will have the number of nodes and edges reduced in order to facilitate the partitioning step. At the figure 11 states an example of how the coarsening HEM algortihm matches the nodes iteratively until a desired number k of nodes.

Figure 11: Example of graph coarsening



Source: Author (2026)

At this work, the coarsening algorithm used is the HEM, this is the coarsening state of the art algorithm used by libraries such as METIS. This algorithm starts with a graph $G(V, E)$ and matches the nodes and edges so at the end the number of vertices v is equal to a parameter v' inserted by the user.

5.1 Coarsening process

As seen in 2.4.2, the finest graph G_0 for the coarsening process is built from a matrix. From G_0, an algorithm based on HEM is used to build clusters over the nodes in $G_i = (V(G_i), E(G_i))$ and build a coarser graph $G_{i+1} = (V(G_{i+1}), E(G_{i+1}))$. At this work, where implemented four strategies for the HEM algorithm, random, sorted, modified and light. The desired strategy to be used is inserted as a parameter in the *simulation_parameters.json* file at the coarsening step.

5.1.1 Random Heavy Edge Matching

The first implemented strategy is the random and it is the foundational concept of the HEM algorithm. The default behavior of this strategy is to iteratively pair up connected nodes that share the heaviest edge weight. In its most basic form, the

algorithm, as described in algorithm 1, randomly selects an unmatched node, searches for an available unmatched neighbor connected by the maximum edge weight, and groups them together into a single cluster. This process repeats until no further matches can be made. By prioritizing the merging of tightly coupled nodes, HEM effectively reduces the graph's degrees of freedom while preserving the most significant structural and numerical relationships of the original matrix.

Algorithm 1 Random Heavy-Edge Matching

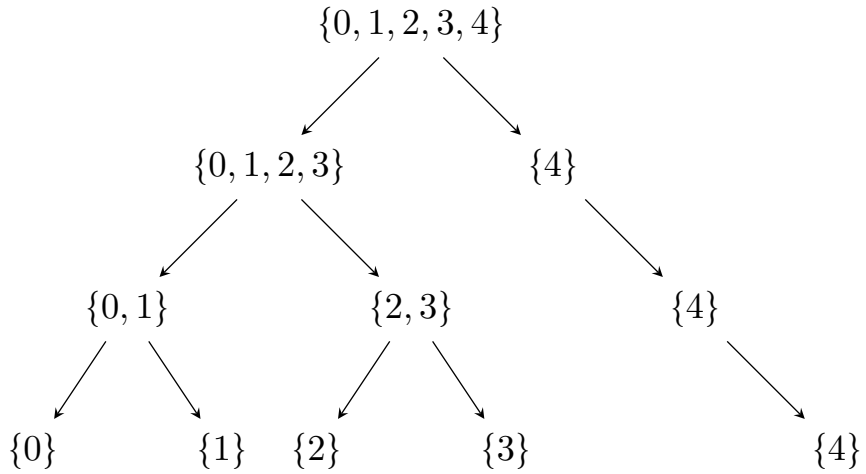
Input: Graph $G = (V, E)$
Output: Set of coarse clusters C for the next graph
$C \leftarrow \varnothing$
Mark all nodes in V as *unmatched*
while there are *unmatched* nodes in V **do**
  Pick a random *unmatched* node $u \in V$
  Find an *unmatched* neighbor v such that $w(u, v)$ is maximum
  if such v exists **then**
    $C \leftarrow C \cup \{\{u, v\}\}$
    Mark u and v as *matched*
  else
    $C \leftarrow C \cup \{\{u\}\}$
    Mark u as *matched*
  end if
end while
return C

However, a strictly random HEM approach can occasionally leave nodes isolated if all of their neighbors are matched before they are selected. This isolation forces single-node clusters to propagate through the levels, ultimately resulting in an unbalanced cluster tree as seen in figure 12.

5.1.2 Sorted Heavy Edge Matching

To counteract this, modified versions of the algortihm were employed at this thesis. The first modified version is the sorted, described in algorithm 2. This version addresses the isolation problem by pre-sorting the nodes before attempting to find matches. Known as Sorted Heavy-Edge Matching (SHEM), this strategy computes the weighted degree for every node in the graph, which is the sum of the weights of all its incident edges. The nodes are then sorted in descending order based on this metric. By iterating through this sorted list, the algorithm ensures that the most highly connected and structurally critical nodes, the "hubs" of the graph, are given priority to find a match. This systematic

Figure 12: Hierarchical tree structure of the sets



Source: Author (2026)

approach significantly reduces the likelihood of dense nodes being left isolated, leading to a much more balanced cluster tree.

5.1.3 Modified Heavy Edge Matching

The third implemented strategy is the modified Heavy Edge Matching (MHEM). While the previous strategies focus entirely on edge weights, MHEM introduces the concept of node weights into the heuristic. During the coarsening process, every time two nodes merge, their weights are summed to form the weight of the new supernode. If the algorithm blindly merges nodes connected by heavy edges, it can inadvertently create massively overweight supernodes, causing severe imbalances that the partitioning step will struggle to resolve. To prevent this, MHEM visits nodes in a random order but evaluates potential partners using a normalized scoring function. It divides the edge weight by the geometric mean of the two node weights. This mathematically penalizes the merging of already heavy nodes, actively favoring the grouping of nodes with similar, smaller weights.

Algorithm 2 Sorted Heavy-Edge Matching

Input: Graph $G = (V, E)$, edge weights $w(e)$
Output: Set of coarse clusters C for the next graph
$C \leftarrow \varnothing$
Compute weighted degree $d(u) = \sum_{v \in N(u)} w(u, v)$ for all $u \in V$
Sort nodes V in descending order of $d(u)$
Mark all nodes in V as *unmatched*
for each node u in sorted V **do**
  if u is *unmatched* **then**
    Find an *unmatched* neighbor v such that $w(u, v)$ is maximum
    if such v exists **then**
      $C \leftarrow C \cup \{\{u, v\}\}$
      Mark u and v as *matched*
    else
      $C \leftarrow C \cup \{\{u\}\}$
      Mark u as *matched*
    end if
  end if
end for
return C

Algorithm 3 Modified Heavy-Edge Matching

Input: Graph $G = (V, E)$, edge weights $w(e)$, node weights $vw(x)$
Output: Set of coarse clusters C for the next graph
$C \leftarrow \varnothing$
Mark all nodes in V as *unmatched*
while there are *unmatched* nodes in V **do**
  Pick a random *unmatched* node $u \in V$
  Find an *unmatched* neighbor v that maximizes the score: $S = \dfrac{w(u, v)}{\sqrt{vw(u) \cdot vw(v)}}$
  if such v exists **then**
    $C \leftarrow C \cup \{\{u, v\}\}$
    Mark u and v as *matched*
  else
    $C \leftarrow C \cup \{\{u\}\}$
    Mark u as *matched*
  end if
end while
return C

5.1.4 Light Heavy Edge Matching

The final strategy implemented is the light strategy, or Light-Edge Matching. This approach inverses the greedy logic of standard HEM. Instead of grouping nodes connected by the heaviest edges, it intentionally pairs nodes connected by the lightest edges. The algorithm visits nodes randomly and selects the unmatched neighbor that minimizes the edge weight. While this might seem counterintuitive for minimizing the edge-cut in dense graphs, it proves highly useful for specific topologies, such as very sparse graphs, where forcing the contraction of weak connections can preserve the dominant structural edges for the later refinement phases.

Algorithm 4 Light-Edge Matching

Input: Graph $G = (V, E)$, edge weights $w(e)$
Output: Set of coarse clusters C for the next graph
$C \leftarrow \varnothing$
Mark all nodes in V as *unmatched*
while there are *unmatched* nodes in V **do**
    Pick a random *unmatched* node $u \in V$
    Find an *unmatched* neighbor v such that $w(u, v)$ is minimum
    if such v exists **then**
        $C \leftarrow C \cup \{\{u, v\}\}$
        Mark u and v as *matched*
    else
        $C \leftarrow C \cup \{\{u\}\}$
        Mark u as *matched*
    end if
end while
return C

Beyond the core matching logic, the implemented coarsening process applies universal constraints to ensure the tractability of the resulting hierarchy. Across all strategies, a strict *max_node_weight* constraint is enforced. If merging a candidate node v with node u exceeds this threshold, the candidate is discarded. Furthermore, to mimic the state-of-the-art behavior seen in classical tools like METIS, the coarsening chain evaluates the structural reduction after every level. If the reduction ratio $\frac{|V(G_{i+1})|}{|V(G_i)|}$ exceeds a specified threshold, typically 0.85, or if the total number of nodes drops below a defined *coarsen_limit*, the process halts. This prevents the generation of excessive, inefficient layers where the graph fails to shrink meaningfully.

5.2 Coarsening Parameters

The coarsening step has parameters that can vary according to the use case and the desired result of the user. Parameters for the simulation are describe in table 4.

Table 4: Coarsening Parameters

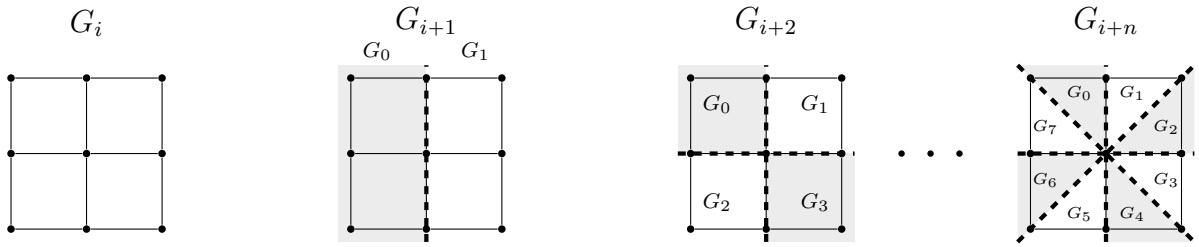
Description	**Default Value**
coarsen_inferior_limit Integer defining the threshold to stop coarsening when the graph has more than or equal to this many nodes.	-
trial_seed Seed for repeatability.	42
coarsen_superior_limit Integer defining the threshold to stop coarsening when the graph has fewer than or equal to this many nodes.	-
max_levels Integer specifying the maximum number of coarsening levels.	-
weight String indicating the edge weight attribute to use for coarsening.	"weight"
strategy String defining the coarsening strategy.	"sorted"
coarsen_ratio Floating-point number to stop the process if the reduction ratio exceeds this value.	-

Source: Author (2026)

6 PARTITIONING

At this chapter will be discussed how the partitioning step were implemented. This step happens after the coarsening and is the central and the only stage of the pipeline that quantum algorithms are utilized. This step will partition the coarse graph $G_i = (V(G_i), E(G_i))$ into k sub-graphs. Starting from the graph G_i, it is utilized the k-way partitioning in order to apply recursive bisections, as in figure 13, n times to reach the desired number k of sub-graphs.

Figure 13: Recursive k-way bisection

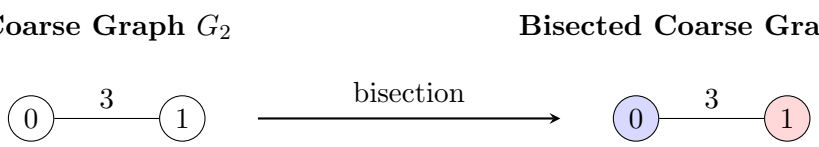


Source: Author (2026)

6.1 Partitioning Process

The overall goal of the partitioning stage is to assign each graph node $u \in V$ to a part label $\pi(u) \in \{0, \ldots, k-1\}$ such that the edge cut is small and the parts are balanced with respect to node weights at this thesis stored as *vweight*. In the multilevel pipeline, the recursive k-way bisection strategy is a consecutive bipartition of the graph as shown in figure 14 and is performed on a coarsened graph and then projected back to the fine level by uncoarsening.

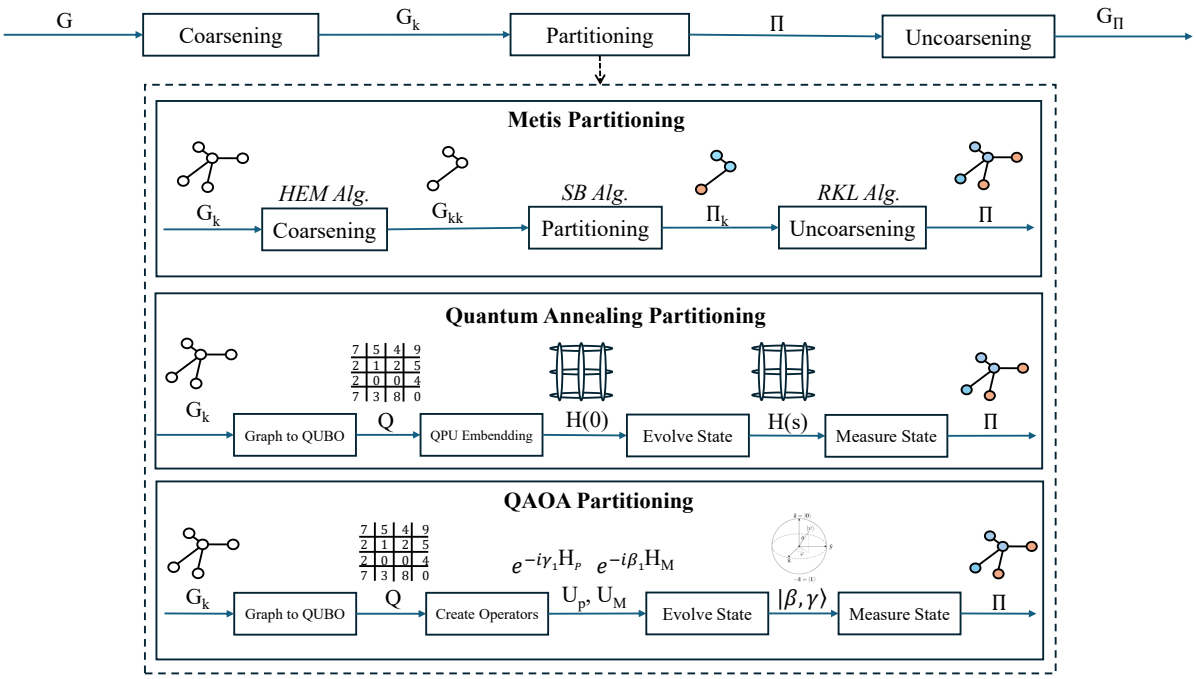
Figure 14: Single bisection of coarsed graph G_2.



Source: Author (2026)

At this work were implemented three different strategies for the partitioning step, see figure 15.

Figure 15: Partitioning Strategies. Where HEM Alg. stands for Heavy Edge Matching Algortihm, SB Alg. for Spectral Bissection Algorithm, RKL Alg. for Recursive Kernighan-Lin Algorithm, QUBO for Quadratic Unconstrained Binary Optimization and QPU for Quantum Processing Unit



Source: Author (2026)

6.1.1 METIS Partitioning

At this work pipeline, METIS is called through the `pymetis` wrapper using adjacency lists. Even though the wrapper hides the internals, METIS still executes the full multilevel logic internally, coarsen → initial partition → refine, for the graph it receives.

Algorithm 5 METIS Multilevel k-way Partitioning

Input: Graph $G = (V, E)$, desired parts k
Output: Partition labels $\pi : V \to \{0, \ldots, k-1\}$
Build a coarsening hierarchy $G_0 \to G_1 \to \cdots \to G_L$ by matching nodes
Compute an initial partition π_L on the coarsest graph G_L
for $\ell = L - 1$ down to 0 **do**
    Project $\pi_{\ell+1}$ to G_ℓ obtaining π_ℓ
    Refine π_ℓ locally to decrease cut while maintaining balance
end for
return π_0

6.1.2 Quantum Annealing

The QA-based method in this work performs *recursive bisection* using a balanced-cut QUBO formulation on subgraphs of the coarsest graph. For a subgraph $H = (V_H, E_H)$, define a binary assignment $x_u \in \{0, 1\}$ indicating which side of the cut a node belongs to. A standard quadratic formulation used here is:

$$\min_{x \in \{0,1\}^{|V_H|}} \quad \mathrm{cut}(x) \; + \; \lambda \left(\sum_{u \in V_H} c_u x_u - T \right)^2 ,$$

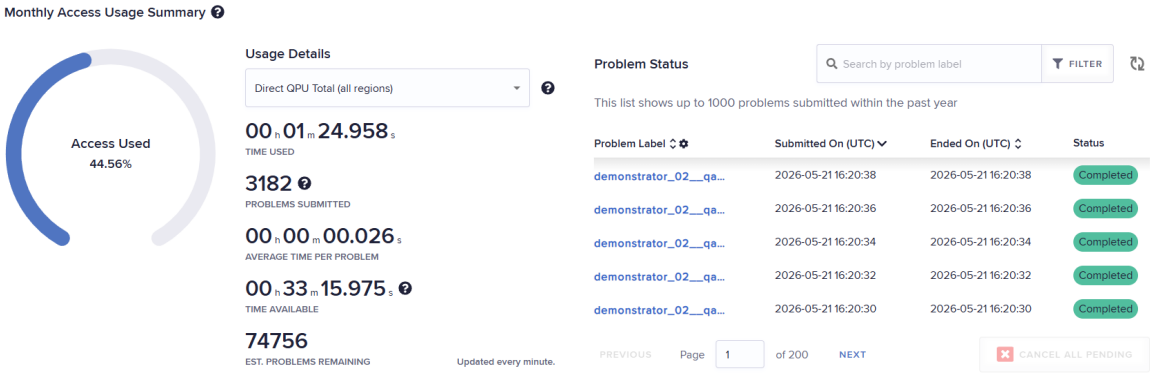
where $\mathrm{cut}(x)$ penalizes edges crossing the bipartition weighted by *weight*, c_u is a node weight, T is the target weight for one side and λ controls the strength of the balance penalty. This objective is assembled into a QUBO matrix Q such that

$$E(x) = \sum_{i \leqslant j} Q_{ij} x_i x_j .$$

The implementation supports two execution modes for this QUBO. When `simulated=true`, the code uses `dwave-neal`'s *SimulatedAnnealingSampler* to draw `num_reads` samples from the QUBO and selects the lowest-energy sample. When `simulated=false`, the same QUBO is submitted through the D-Wave Ocean stack to the configured quantum annealing processor. In that mode, the user may specify `qpu_solver_name`, `qpu_region`, and `qpu_problem_label`. Before submission, the implementation checks whether the dense balanced-cut QUBO can be embedded in the selected hardware capacity.

Image 16 shows how is the Dwave-leap platform for visualize the job submissions and manage the access tokens usage. It is possible to access important information such as total time used, number of submitted problems, estimations of the number of still available problems and so on. It is also possible to visualize all the jobs submitted and what are their current status of execution.

Figure 16: Dwave-Leap Platform



Source: Author (2026)

To obtain $k > 2$ parts, the method repeatedly selects the currently heaviest block by *vweight* and bisects it again, until k blocks are created. Balance tolerance is used to slightly perturb the target T per split within the tolerance band to diversify solutions. The full recursive procedure is run *qa_num_starts* times with different seeds. The best run is chosen via score $= \text{cut}(\pi) + \Lambda \cdot \text{violation}(\pi)$, where Λ is *qa_select_balance_lambda* and $\text{violation}(\pi)$ measures how much part weights deviate from the allowed balance band.

Algorithm 6 Recursive Quantum Annealing Partitioning

Input: Coarse graph G_c, parts k, penalty λ, tolerance τ
Output: Partition labels π
for $s = 1$ to `qa_num_starts` **do**
  Initialize blocks $\mathcal{B} \leftarrow \{V(G_c)\}$
  while $|\mathcal{B}| < k$ **do**
    Pick the heaviest block $B \in \mathcal{B}$
    Let $H \leftarrow G_c[B]$ be the induced subgraph
    Set target $T \leftarrow \frac{1}{2} \sum_{u \in H} c_u$ and perturb T within $\pm\tau$
    Build QUBO Q for $\text{cut}(x) + \lambda(\sum c_u x_u - T)^2$
    Sample x from Q using SA with `num_reads`; take best-energy sample
    Split B into $B_0 = \{u : x_u = 0\}$ and $B_1 = \{u : x_u = 1\}$; update $\mathcal{B}$
  end while
  Convert blocks in $\mathcal{B}$ to label map π_s
  Compute $\text{score}(\pi_s) = \text{cut}(\pi_s) + \Lambda \cdot \text{violation}(\pi_s)$
end for
return the π_s with minimal score

6.1.3 Quantum Approximation Optimization Algorithm

The QAOA-based method follows the same outer *recursive bisection* structure as the QA approach, but replaces the annealing QUBO solver by a small QAOA solve on each bisection subproblem. The same balanced-cut QUBO is built first. It is then transformed into an Ising Hamiltonian, up to a constant, using $x_i = (1 - z_i)/2$ with $z_i \in \{+1, -1\}$

$$E(z) = \sum_i h_i z_i + \sum_{i<j} J_{ij} z_i z_j + \text{const.}$$

This yields a cost Hamiltonian H_C composed of Pauli-Z and ZZ terms. The mixer Hamiltonian is chosen as $H_M = \sum_i X_i$. A p-layer QAOA circuit alternates cost and mixer layers (an example of this quantum circuit can be seen in figure 8)

$$U(\gamma, \beta) = \prod_{\ell=1}^{p} e^{-i\beta_\ell H_M} e^{-i\gamma_\ell H_C}.$$

Two execution modes are implemented. When `simulated=true`, the parameters (γ, β) are optimized locally with Adam on a PennyLane statevector simulator using the configured learning rate. After optimization, the circuit is sampled for `num_shots` shots, and the best sampled bitstring, lowest QUBO energy, is taken as the bisection result. When `simulated=false`, the QAOA circuit is built through the QLM/myQLM stack and submitted to the remote QPU named by `qlm_qpu_name`. In that case, the classical optimizer performs remote objective evaluations and a final sampling job on the QLM backend.

Because local QAOA simulation grows exponentially with the number of qubits and current remote QPUs have strict qubit limits, QAOA requires aggressive coarsening. If a bisection subproblem exceeds the available local or remote capacity, the method aborts and the user must coarsen more or choose another strategy.

Algorithm 7 Recursive QAOA Partitioning

Input: Coarse graph G_c, parts k, QAOA depth p
Output: Partition labels π
for $s = 1$ to `qa_num_starts` **do**
  Initialize blocks $\mathcal{B} \leftarrow \{V(G_c)\}$
  while $|\mathcal{B}| < k$ **do**
    Pick the heaviest block $B \in \mathcal{B}$
    Let $H \leftarrow G_c[B]$
    Build balanced-cut QUBO and map to Ising (h, J) defining H_C
    Prepare $|+\rangle^{\otimes n}$ and run p-layer QAOA with parameters (γ, β)
    Optimize (γ, β) for `qaoa_steps` with learning rate `qaoa_lr`
    Sample `qaoa_shots` bitstrings; select best by QUBO energy
    Split B according to the best bitstring; update $\mathcal{B}$
  end while
  Convert blocks to label map π_s and score it using cut + balance violation penalty
end for
return the π_s with minimal score

6.2 Partitioning Parameters

Due to the different available strategies for the partition, this stage has different sets of parameters for each possible used strategy. Firstly, there are the common parameters that all the strategies uses, they are called `partitioning_parameters.common` and has the structure as in the table 5. Additionally to the common parameters, the parameters necessary for running the recursive quantum annealing and recursive qaoa strategies are described in tables 6 and 7 respectively.

Table 5: Common partitioning parameters.

Description	Default Value
k_target Integer indicating the desired number of partitions.	2
balance_tolerance Allowed imbalance between partition weights.	0.1

Source: Author (2026)

Table 6: Parameters specific to the recursive Quantum Annealing strategy.

Description	**Default Value**
qubo_balance_lambda Balance constraint penalty weight λ in the bipartition QUBO.	0.1
num_reads Number of samples drawn by the simulated annealer per bipartition QUBO solve.	5
num_starts Number of full recursive runs.	5
balance_violation_lambda Penalty multiplier Λ for balance violation during best-of selection.	0.1
simulated If `true`, solve the QUBO with local simulated annealing; if `false`, submit it to the configured D-Wave QPU.	true
qpu_solver_name Optional D-Wave solver name used for hardware execution.	–
qpu_region Optional D-Wave Leap region used for hardware execution.	–
qpu_problem_label Optional label attached to submitted D-Wave QPU problems.	–

Source: Author (2026)

Table 7: Parameters specific to the recursive QAOA strategy.

Description	Default Value
qubo_balance_lambda Balance constraint penalty weight λ in the bipartition QUBO.	0.1
num_starts Number of full recursive runs.	12
balance_violation_lambda Penalty multiplier Λ for balance violation during best-of selection.	0.1
circuit_depth QAOA circuit depth.	12
num_steps Number of local Adam steps or remote objective-evaluation iterations for the QAOA parameters.	12
adam_learning_rate Learning rate for the local Adam optimizer when `simulated=true`.	0.1
num_shots Number of measurement shots for sampling bitstrings after optimization.	5
simulated If `true`, execute QAOA locally with PennyLane; if `false`, submit QAOA circuits to the configured QLM backend.	true
qlm_qpu_name QLM QPU name used when `simulated=false`.	–

Source: Author (2026)

6.3 Running the Hardware Backends

The software developed for this thesis can execute the same multilevel pipeline with local emulation or with remote quantum hardware, depending on the partitioning configuration. The single-run and batch entry points are shown in Listing 6.1. A detailed operational guide, including environment setup and backend inspection commands, is maintained in the thesis repository under *src/libs/multilevel_scheme/partitioning/qa/README.md*.

The concrete backend parameters used in the experimental campaign are summarized later in Table 13.

Listing 6.1: Single-run and batch commands used by the thesis software.

```
# Run one simulation.
./scripts/run demonstrators/demonstrator_01

# Prepare and run the default batch study in the foreground.
./scripts/run_batch
```

For D-Wave quantum annealing, the important configuration switch is `simulated=false` inside `partitioning_parameters.strategies.quantum_annealing`. Listing 6.2 gives the minimal structure used by the runtime. Authentication credentials and account-specific Leap configuration are not stored in the thesis files; they are handled by the local D-Wave Ocean setup.

Listing 6.2: Example of configuration pattern for running recursive quantum annealing on D-Wave hardware.

```
"partitioning_parameters": {
    "partitioning_strategy": "quantum_annealing",
    "common": {
        "k_target": 4,
        "balance_tolerance": 2.0
    },
    "strategies": {
       "quantum_annealing": {
                "qubo_balance_lambda": 2,
                "num_starts": 4,
                "num_reads": 100,
                "balance_violation_lambda": 5.0,
                "simulated": false,
                "qpu_region": "eu-central-1",
                "qpu_solver_name": "Advantage2_system2.1",
                "qpu_problem_label": "DD_ansys_v0_without_base"
        },
    }
}
```

For QAOA on a QLM backend, the partitioning strategy is changed to `quantum_approximation_optimizer` and `simulated=false`. Listing 6.3 shows the corresponding configuration pattern. In contrast to the local PennyLane mode, the learning rate is not the central control parameter for QLM hardware execution; the relevant values are the circuit depth, number of optimizer iterations, number of shots, and the QLM QPU name.

Listing 6.3: Example of configuration pattern for running recursive QAOA on a QLM backend.

```
"partitioning_parameters": {
    "partitioning_strategy": "quantum_approximation_optimizer",
    "common": {
        "k_target": 4,
        "balance_tolerance": 2.0
    },
    "strategies": {
        "quantum_approximation_optimizer": {
                "qubo_balance_lambda": 2,
                "num_starts": 1,
                "num_steps": 1,
                "num_shots": 8,
                "balance_violation_lambda": 5.0,
                "circuit_depth": 1,
                "simulated": false,
                "qlm_qpu_name": "qat.qpus:Jiqcer5QPU"
            }
    }
}
```

The QLM path may also be controlled through environment variables such as `HDD_QLM_JOB_TIMEOUT_SECONDS` and `HDD_QLM_STATUS_POLL_INTERVAL_SECONDS`. These options are useful because remote QAOA execution submits many objective-evaluation and sampling jobs. The batch runner therefore treats remote QLM cases asynchronously so that long hardware submissions do not block all other completed cases.

7 UNCOARSENING

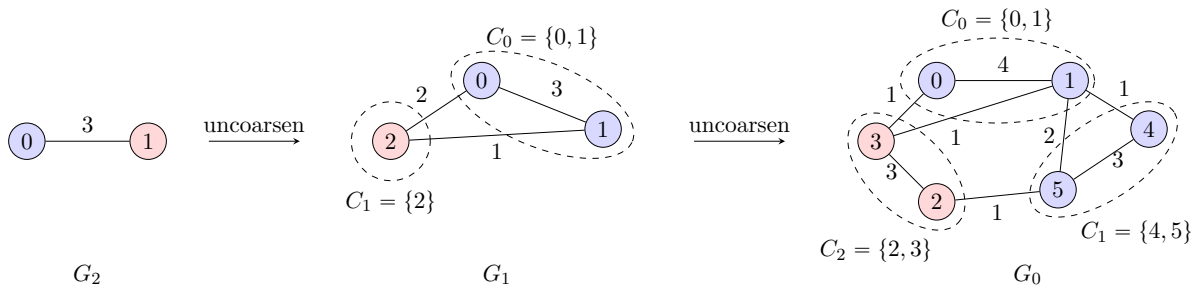
This chapter describes the uncoarsening phase of the multilevel graph partitioning pipeline. Here will be depicted the procedure, what is necessary to run this stage and what are the possible outputs.

7.1 Uncoarsening Process

After coarsening and partitioning, its obtained a partition assignment on the coarsest graph G_k. Uncoarsening lifts this partition back through the intermediate graphs until it reaches the original graph G_0, while applying local refinement at each level to improve the objective, meaning the edge cut and, when requested, the balance.

Figure 17 illustrates the key idea. A partition computed on a very small coarse graph, for example G_2, is projected back to a less coarse graph G_1, and then projected again to the original graph G_0. During each projection step, the part labels are preserved, so when a coarse node expands into multiple fine nodes, all those fine nodes inherit the same part label. This is exactly why, in the figure, the clusters C_0, C_1, C_2 remain recognizable while the graph gains nodes and edges. After this projection, the algorithm refines the assignment locally, so that the partition on the finer graph becomes better with respect to the chosen objective while still remaining feasible in terms of balance.

Figure 17: Example of graph uncoarsening with preserved partitions



Source: Author (2026)

In the Python implementation, the uncoarsening logic is implemented mainly through the function `uncoarsen_and_refine`. It receives the chain of graphs $\{G_0, G_1, \ldots, G_k\}$ from fine to coarse, the corresponding mapping dictionaries that describe how each node of G_ℓ was contracted into a node of $G_{\ell+1}$, and a partition map defined on the coarsest graph. The function then walks backwards through the levels, from the coarsest representation back toward the original graph. At each level ℓ, it constructs a partition on the current fine graph G_ℓ by assigning each fine node u the label of its coarse representative, using the stored contraction map. In mathematical form this is the

projection $\Pi_\ell(u) \leftarrow \Pi_{\ell+1}(\mathcal{M}_\ell(u))$, and it corresponds to the uncoarsening arrows shown in Figure 17, where the partitioned coarse clusters expand into their finer constituents.

Once the partition has been projected to G_ℓ, the code immediately refines it using `refine_kway_boundary_moves`.

The refinement procedure begins by computing the current total weight of each partition, using the node attribute `vweight` by default. From the total weight it defines an ideal target weight per partition, then it derives lower and upper bounds based on `balance_tolerance`. These bounds are treated as hard feasibility constraints, so a move that would push a partition below the lower bound or above the upper bound is rejected.

After the balance bounds are defined, the algorithm identifies boundary nodes, meaning nodes that have at least one neighbor in a different partition. Only boundary nodes are considered, because moving an interior node cannot reduce the cut locally without crossing a boundary. The boundary nodes are shuffled using a seeded random number generator, then the algorithm iterates through them. For each boundary node u, the algorithm looks at which partitions appear in its neighborhood. For each candidate partition b different from the current partition a, the algorithm computes how strongly u is connected to b compared to a, using the sum of edge weights from u into each partition. This quantity represents the cut gain, and it indicates whether moving u would reduce the cut. If the refinement objective is `cut`, the algorithm only accepts moves that strictly reduce the cut, so it behaves as a strict cut minimizer. If the refinement objective is `cut_balance`, the algorithm uses a combined score, it still prefers cut improvements, but it also subtracts a balance penalty term scaled by `refine_balance_lambda`. The balance penalty is computed as a change in squared deviations of the touched partitions from the ideal target weight, so the refinement encourages partitions that are both low cut and well balanced.

This process is repeated for multiple passes, up to `refine_max_passes_per_level`. The refinement stops early if a pass makes no moves, which indicates a local optimum for the chosen objective under the given balance constraint, and it can also stop within a pass if `refine_max_moves_per_pass` is set and the move cap is reached. The result of the pass becomes the partition for the current level, then uncoarsening continues to the next finer graph, until the algorithm finally returns a partition defined on G_0.

Algorithm 8 Uncoarsen and refine

Input: Graph levels $G_0, \ldots, G_k$, maps $\mathcal{M}_0, \ldots, \mathcal{M}_{k-1}$, coarse partition Π_k
Output: Partition Π_0 on G_0
$\Pi \leftarrow \Pi_k$
for $\ell = k - 1$ down to 0 **do**
  for each node $u \in V(G_\ell)$ **do**
    $c \leftarrow \mathcal{M}_\ell(u)$
    $\Pi_\ell(u) \leftarrow \Pi(c)$
  end for
  $\Pi \leftarrow$ RefineBoundaryMoves$(G_\ell, \Pi_\ell, k, \text{tolerance}, \lambda, \text{limits})$
end forreturn Π

Algorithm 9 Greedy boundary refinement

Input: Graph G, partition Π, number of parts k
Output: Refined partition Π
Compute part weights and target bounds from `balance_tolerance`
for pass = 1 to `refine_max_passes_per_level` **do**
  Build boundary node list, shuffle it with RNG
  moved $\leftarrow 0$
  for each boundary node u **do**
    Consider neighbor parts $b \neq \Pi(u)$
    Reject b if hard balance bounds would be violated
    Compute cut gain and balance penalty change
    Select best improving move, apply it, update weights
    moved $\leftarrow$ moved $+1$
    if `refine_max_moves_per_pass` is set and reached **then**
      break
    end if
  end for
  if moved $= 0$ **then**
    break
  end if
end forreturn Π

7.2 Uncoarsening parameters

The uncoarsening and refinement parameters are provided under `uncoarsening_parameters` in the simulation parameters file. The refinement routine is

always the boundary move heuristic described above, so there is no active `refine_method` switch in the implementation.

Table 8: Uncoarsening and refinement parameters.

Description	Default Value
refine_objective Refinement objective.	`"cut"`
refine_balance_lambda Balance penalty weight λ.	`1.0`
refine_max_passes_per_level Maximum refinement passes per uncoarsening level.	`5`
refine_max_moves_per_pass Maximum accepted moves per pass.	`None`
validate_node_weights Parsed configuration flag.	`None`

Source: Author (2026)

8 PERMUTATION AND PROFILING

This chapter describes the final matrix-processing stage of the proposed workflow. After the graph partition has been computed on the coarse graph and projected back to the original scale, the partition labels are converted into a permutation of the degrees of freedom. This permutation is applied to the stiffness matrix K and to the mass matrix M, producing reordered matrices that preserve the physical system but expose a different sparsity layout. Section 8.1 defines this operation, while Section 8.2 defines the structural metrics used to evaluate the resulting matrices.

The purpose of this chapter is to connect the graph-partitioning result to the solver objective discussed in Section 2.2.3.2. Matrix permutation is not a solver by itself, and it does not change the eigenvalues of the original operator. Its role is to reorganize the matrix so that later preconditioners can exploit locality, smaller effective bands, and clearer block structure.

8.1 Matrix Permutation

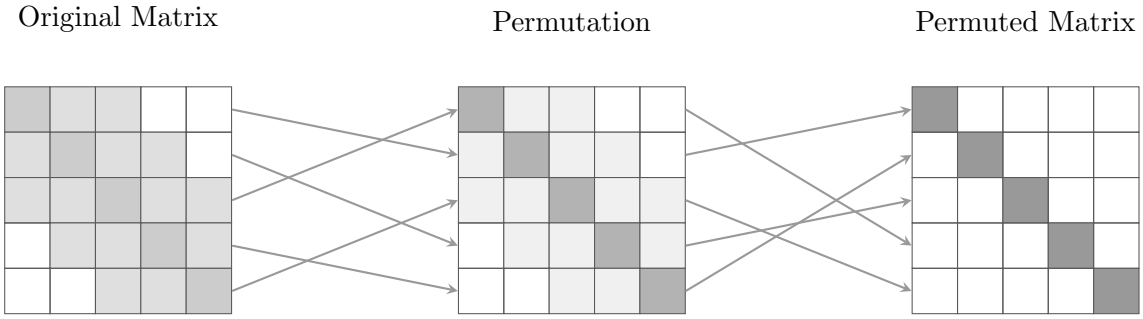
After the graph G_k has been uncoarsened back to its original size, it is necessary to apply matrix permutation to the original K and M matrices. The final partition assigns each node to one of the target subdomains. The implementation then orders nodes by partition label and expands this node ordering to all degrees of freedom. For a matrix A, the resulting symmetric permutation is

$$A_p = PAP^T, \tag{8.1}$$

where P is the permutation matrix and A_p is the reordered matrix. Since $P^T = P^{-1}$, this transformation is a similarity transformation. Therefore, A and A_p have the same eigenvalues, and the original condition number is preserved. The benefit is structural: the nonzero entries are moved to different row and column positions, ideally creating tighter blocks or stronger diagonal concentration that can later help the construction and application of preconditioners.

Following the permutation, the new structural characteristics of the matrices must be evaluated. This analytical process is referred to in this thesis as matrix profiling. The profiling stage does not compute the true runtime of a solver. Instead, it computes structural indicators that are known to affect the cost and effectiveness of many direct and preconditioned iterative methods.

Figure 18: Visual representation of matrix permutation



Source: Author (2026).

8.2 Matrix Profiling

For each nonzero entry $a_{ij} \neq 0$, the distance from the diagonal is $d_{ij} = |i - j|.$, the maximum bandwidth is then

$$B(A) = \max_{a_{ij} \neq 0} |i - j|, \tag{8.2}$$

and the Avg. Bandwidth metric is

$$\bar{B}(A) = \frac{1}{\text{nnz}(A)} \sum_{a_{ij} \neq 0} |i - j|. \tag{8.3}$$

To capture local concentration around the diagonal, the implementation also counts the number of entries within a fixed band b:

$$N_b(A) = \left| \{(i, j) : a_{ij} \neq 0 \ \wedge \ |i - j| \leqslant b\} \right|, \tag{8.4}$$

and its normalized fraction

$$F_b(A) = \frac{N_b(A)}{\text{nnz}(A)}. \tag{8.5}$$

Within the software framework developed for this thesis, these metrics are computed for the original and permuted versions of both K and M. The implementation can evaluate fixed bands, such as $b = 200$, and percentage-defined bands, where b is computed from the matrix dimension. The Results chapter reports percentage-defined bands at 0.1%, 0.4%, 0.7%, and 1% so that matrices of different sizes can be compared consistently.

Table 9: Matrix profiling metrics computed for original and permuted matrices.

Description	**Default Value**
n Matrix dimension n.	–
nnz Number of non-zero entries.	–
bandwidth Maximum distance of a non-zero entry from the diagonal.	–
avg_bandwidth Mean distance of non-zeros from the diagonal.	–
nnz_within Count of non-zeros within the selected fixed or percentage-defined band.	-
frac_within Fraction of non-zeros within the selected fixed or percentage-defined band.	-

Source: Author (2026)

The link between these structural metrics and solver cost must be interpreted carefully. A lower maximum bandwidth can reduce the effective band size used by banded or localized factorizations, which is relevant to the complexity model in Equation 2.21. A higher value of $F_b(A)$ indicates that more matrix entries are concentrated near the diagonal, which can make block or domain-decomposition preconditioners more local. However, because Equation 8.1 is spectrum preserving, the permutation alone does not reduce the original condition number $\kappa(A)$.

The practical target is instead the condition number of the preconditioned operator. As previously detailed in Section 2.2.3.2, the number of Krylov iterations depends on the effective condition number of the preconditioned system, as expressed in Equation 2.24. The role of the permutation is therefore to create a sparsity pattern from which a stronger or cheaper preconditioner can be built. The final results obtained from the simulations are presented in Chapter 9, where these metrics are evaluated for both demonstrators.

9 RESULTS

This chapter presents the experimental results obtained with the hybrid multilevel pipeline introduced in Chapter 4 and detailed in Chapters 5, 6, 7, and 8. The objective is to verify that the complete workflow produces valid permutations for the stiffness matrix K and the mass matrix M, and to evaluate whether the resulting matrix structures are more favorable for preconditioned linear solvers.

9.1 Metrics

The interpretation of the results follows the structural metrics defined in Chapter 8. The main quantities are the maximum bandwidth, average bandwidth, the fraction of nonzero entries located inside percentage-defined bands around the diagonal and the number of iterations on the worst case scenario. For a matrix $A \in \mathbb{R}^{n\times n}$ and a percentage value ρ, the band radius and reported locality fraction are

$$b_\rho(A) = \left\lceil \frac{\rho}{100} n \right\rceil, \qquad \mathrm{frac}_{\rho\%}(A) = \frac{|\{(i,j) : a_{ij} \neq 0 \ \wedge \ |i-j| \leqslant b_\rho(A)\}|}{\mathrm{nnz}(A)}. \tag{9.1}$$

The tables report $\rho \in \{0.1, 0.4, 0.7, 1.0\}$. These percentages make the metrics comparable across matrices of different sizes, while still measuring concentration near the diagonal.

The solver relevance of these metrics follows from the discussion in Section 2.2.3.2. A symmetric permutation can be written as

$$A_p = PAP^T, \tag{9.2}$$

where P is a permutation matrix. Since P is orthogonal, this transformation preserves the spectrum of the original matrix and therefore does not directly change $\kappa(A)$. The expected benefit is instead indirect: a reordered matrix with a more localized sparsity pattern can make incomplete, block, domain-decomposition, or banded preconditioners cheaper and more effective. In the terminology of Section 2.2.3.2, the purpose of the permutation is to improve the quality and cost of the preconditioning operator so that the effective condition number $\kappa(M^{-1}A)$ becomes more favorable. This relation is connected to Equation 2.24. In addition, when a banded or localized factorization is used, the factorization complexity discussed in Equation 2.21 depends strongly on the effective bandwidth parameter p.

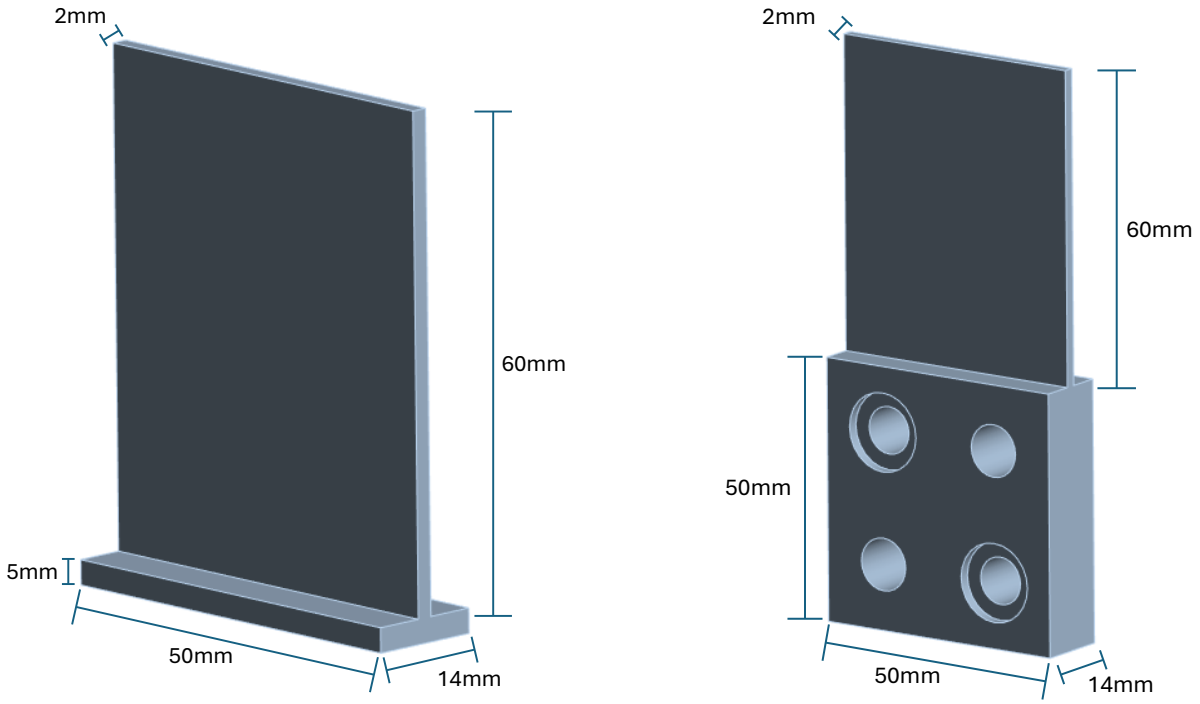
To keep the chapter compact, the colored matrix figures focus on the stiffness matrix K, from which the graph partition is computed. The tables still report both K

and M and compare all completed method families requested for the thesis: METIS, hardware quantum annealing, emulated quantum annealing, and emulated QAOA.

9.2 Demonstrators

The results focus on two ANSYS-derived geometries, image 19 used throughout the implementation and batch study. Demonstrator 01 corresponds to the blade configuration without base support, while Demonstrator 02 corresponds to the same blade model with a 5 mm base. This pair is useful because it preserves the same general physical context while changing the graph topology enough to alter the behavior of the multilevel partitioning workflow.

Figure 19: Demonstrator 01 on the left, and demonstrator 02 on the right



Source: Author (2026)

Table 10: Overview of the demonstrators

Demonstrator	**Graph Nodes**	**Graph Edges**	**Matrix Size n**
Demonstrator 01: blade model without base support	104145	1153409	312435
Demonstrator 02: blade model with 5 mm base	91080	2195175	273240

Source: Author (2026)

Table 10 indicates that the two demonstrators should not be expected to behave identically under permutation. Demonstrator 02 has fewer graph nodes, but substantially denser graph connectivity. This distinction is important because a denser graph can make it more difficult to reduce the maximum bandwidth, even when the reordered matrix becomes more locally concentrated around the diagonal.

9.3 Experimental Setup

All analyzed runs follow the workflow described in Chapter 4. The stiffness and mass matrices are loaded, the graph is constructed from the stiffness matrix, the graph is coarsened with modified heavy-edge matching, and the partitioning backend solves the coarse partitioning problem. The resulting partition is projected back through the hierarchy and refined by local boundary updates. Finally, the recovered partition ordering is applied to both K and M, and the quality of the reordered matrices is quantified using the profiling metrics from Chapter 8. Table 11 lists the shared numerical parameters that were kept fixed so that differences between rows can be attributed primarily to the selected backend and coarse-size preset.

Table 11: Common simulation parameters used across the analyzed batch cases

Description	**Value**
Degrees of freedom per node	3
Random seed	42
Target number of partitions k	4
Balance tolerance	2.0
Coarsening strategy	Modified heavy-edge matching
Coarsening ratio	0.99
Maximum coarsening levels	30
Edge attribute used during coarsening	`weight`
Refinement objective	`cut_balance`
Maximum refinement passes per level	50
Maximum refinement moves per pass	5000

Source: Author (2026)

The batch campaign compares the classical METIS baseline, quantum annealing on hardware, simulated quantum annealing, and emulated QAOA. To test how hierarchy depth affects the final permutation, three coarsening-scale presets were considered whenever the backend allowed them: *big* with coarse-size limits $(1500, 2500)$, *medium* with limits $(100, 175)$, and *small* with limits $(4, 5)$. As discussed in Chapter 5, smaller coarsening limits usually imply a deeper hierarchy and a more aggressively reduced coarse problem. Chapter 6 also shows that each backend constrains the feasible coarse size, which is why not every backend was evaluated on every preset.

Table 12 shows the complete batch design used for both demonstrator geometries. METIS was evaluated on all three presets because it is the classical reference method and can handle the largest coarse graphs directly. Hardware quantum annealing was restricted to the medium and small presets because the dense balanced-cut QUBO must satisfy D-Wave embedding and runtime constraints. Emulated quantum annealing was run on the full preset range because it is not limited by QPU embedding. QAOA hardware is shown separately because the remote QLM submissions were treated as platform context rather

than as completed numerical evidence in the final tables. Emulated QAOA was restricted to the small preset because local circuit simulation becomes expensive as the coarse graph grows.

Table 12: Batch cases selected for both demonstrator geometries

Method	**Big**	**Medium**	**Small**	**Description**
METIS	yes	yes	yes	Classical multilevel baseline evaluated across the complete coarse-size range.
QA hardware	N/A	yes	yes	D-Wave execution was limited to coarse graphs that are more compatible with dense-QUBO embedding and hardware runtime constraints.
QA emulated	yes	yes	yes	Local simulated annealing can evaluate the same QUBO family without QPU-capacity restrictions.
QAOA hardware	N/A	N/A	N/A	QLM hardware execution is discussed as platform context, but completed numerical result folders were not available for the final comparison tables.
QAOA emulated	N/A	N/A	yes	Local QAOA emulation was kept on the smallest coarse graph because circuit simulation and repeated objective evaluations scale poorly with problem size.

Source: Author (2026)

Table 13 summarizes the concrete quantum-platform parameters used in the campaign. The complete operational guide for configuring these backends is maintained with the thesis source code in the repository. The same repository contains the scripts and parameter files used to reproduce the simulations.

Table 13: Quantum backend and platform parameters used in the simulation campaign

Backend	**Configuration**
QA hardware	D-Wave Ocean/Leap execution with `qpu_solver_name=Advantage_system6.4`, `simulated=false`, `num_starts=50`, `num_reads=100`, `qubo_balance_lambda=2.0`, and `balance_violation_lambda=5.0`.
QA emulated	Local simulated annealing through the D-Wave Ocean-compatible software path with `simulated=true`, `num_starts=50`, `num_reads=20`, `qubo_balance_lambda=2.0`, and `balance_violation_lambda=5.0`.
QAOA emulated	Local PennyLane-based QAOA emulation with `circuit_depth=5`, `num_steps=6`, `num_shots=20`, `adam_learning_rate=0.1`, `num_starts=20`, and `simulated=true`.
QAOA hardware	Remote QLM execution targeting `qlm_qpu_name=qat.qpus:Jiqcer5QPU`. Runtime logs identify this target as a JuSpark/IQMBackend v2 5-qubit QPU.

Source: Author (2026)

9.4 Results for Demonstrator 01

Demonstrator 01 provides a K-only visual sequence for the representative multilevel run. The results obtained will be shown divided by the partition method used. Outputs for each method include the partition-colored initial graph, coarse graph, uncoarsened graph, original and permuted matrix and band curve concentration.

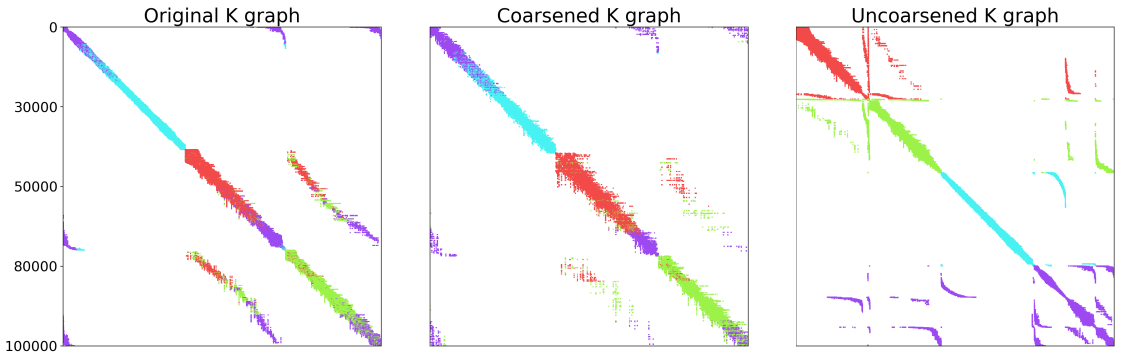
9.4.1 METIS Partitioning

Using the METIS as the partitioning method, were performed three simulations with big, medium and small size for the coarsen graph. Each of the results for all the selected sizes for this method are described and its results discussed. These results obtained for this method will be used as a baseline to evaluate the performance of the quantum-based partitioning methods.

Figure 20 shows the evolution of the K-graph using the METIS strategy for the problem with Big size. The initial graph image corresponds to the graph obtained from the FEM matrix before coarsening. The coarse image visualizes the reduced graph on which the partitioning problem is solved. The uncoarsened image shows the final partition after projection and boundary refinement, which is the direct continuation of Chapter 7. This three graph view is only a example of how the graph is processed during the multilevel

partitioning pipeline. For evaluating the results only the original K matrix and its final permutation will be used.

Figure 20: Partition-colored K-graph views for Demonstrator 01 big coarsen size: initial graph, coarsened graph, and uncoarsened graph.



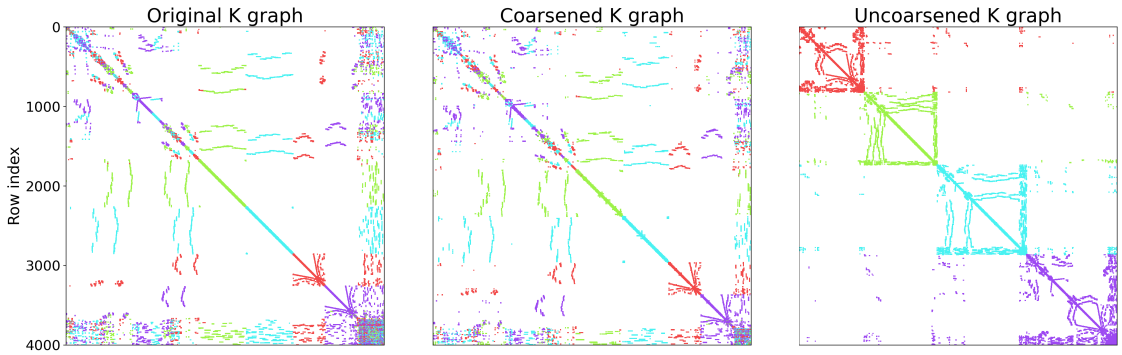
Source: Author (2026)

Only K-stage figures are shown because the graph partition is built from the stiffness matrix and because showing the same sequence for M would duplicate the visual evidence without changing the partitioning decision. The stage-wise images confirm the main multilevel assumption of the thesis: the partition is not created directly on the largest graph. Instead, it is produced on a structurally meaningful coarse representation and then expanded back to the original scale. The final matrix ordering therefore depends jointly on coarsening quality, coarse-level partitioning quality, and uncoarsening refinement.

Figure 22, 38 and 39 show what the result of the permutation using the METIS method in a big, medium and small size for the coarsen graph. Its possible to see that after the permutation, the nodes could be divided into four different blocks with clear distinction between them. These results will be compared with the other methods using the same sizes for the coarsen graph for this demonstrator.

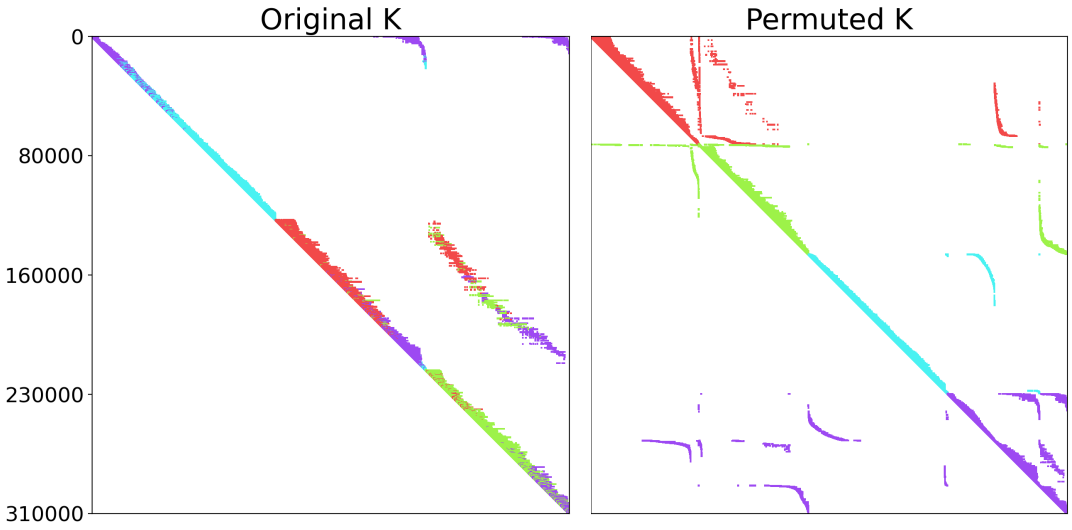
It is worth noting that, by looking only at the graph in Figure 20 after the uncoarsening step, the blocks created by the permutation may not be readily visible. This phenomenon occurs due to the boundary conditions applied to the original matrix, which, as a collateral effect, tend to concentrate the nodes along the diagonal. In Figure 21, where no boundary conditions are applied and the original matrix is therefore less sparse, it is possible to observe the block structure before and after permutation along the diagonal. However, since this scenario lacks boundary conditions, it does not represent the physics of the real geometry and is used here solely for visual comparison purposes.

Figure 21: Partition-colored K-graph views for Demonstrator 03 big coarsen size: initial graph, coarsened graph, and uncoarsened graph.



Source: Author (2026)

Figure 22: Demonstrator 01 K-matrix METIS big coarsen size



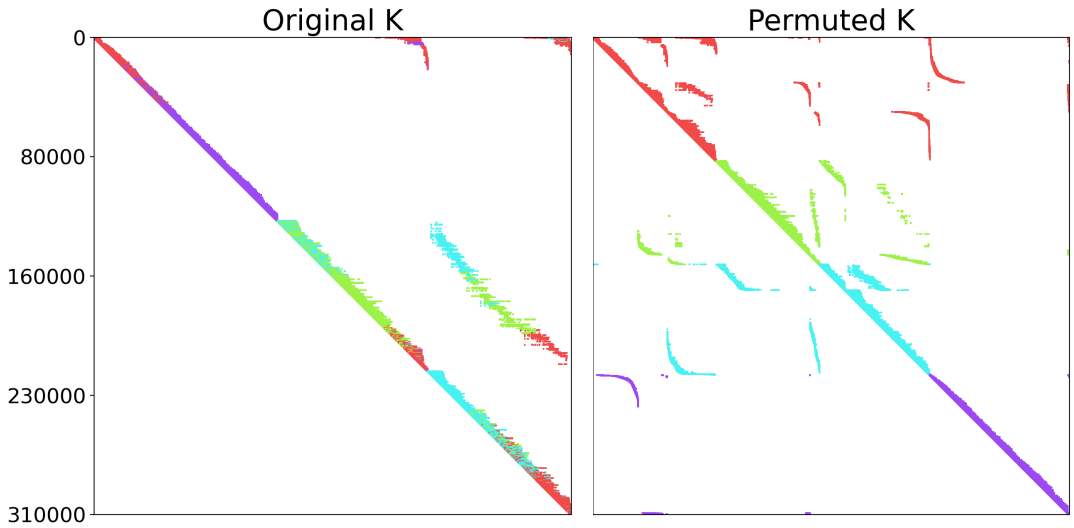
Source: Author (2026)

9.4.2 Quantum Annealing Hardware Partitioning

Using the Quantum Annealing Hardware as the partitioning method, were performed only two simulations with coarsen graph sizes medium and small. The exclusion of the big coarsen size was due to quantum hardware limitation. Figures 23 and 43 shows the result obtained for the simulations using medium and small coarsen size graph respectively. And figures 46 and 44 shows the curves for the fraction of non zero values within band.

Figure 46 shows that for the medium preset, neither method achieves a strong maximum-bandwidth reduction. QA hardware medium reaches a K bandwidth of 311 540 and an average bandwidth of 6 661.11, while METIS medium reaches 310 781 and 6 404.33.

Figure 23: Demonstrator 01 K-matrix Quantum Annealing Hardware medium coarsen size



Source: Author (2026)

In the narrowest 0.1 % band, QA hardware medium is slightly higher (0.3404 versus 0.3386), and it remains stronger at 0.4 % (0.7237 versus 0.7071) and 0.7 % (0.8633 versus 0.8529). Thus, the hardware run does not improve the maximum bandwidth or average bandwidth over METIS, but it produces a broader mid-band concentration improvement.

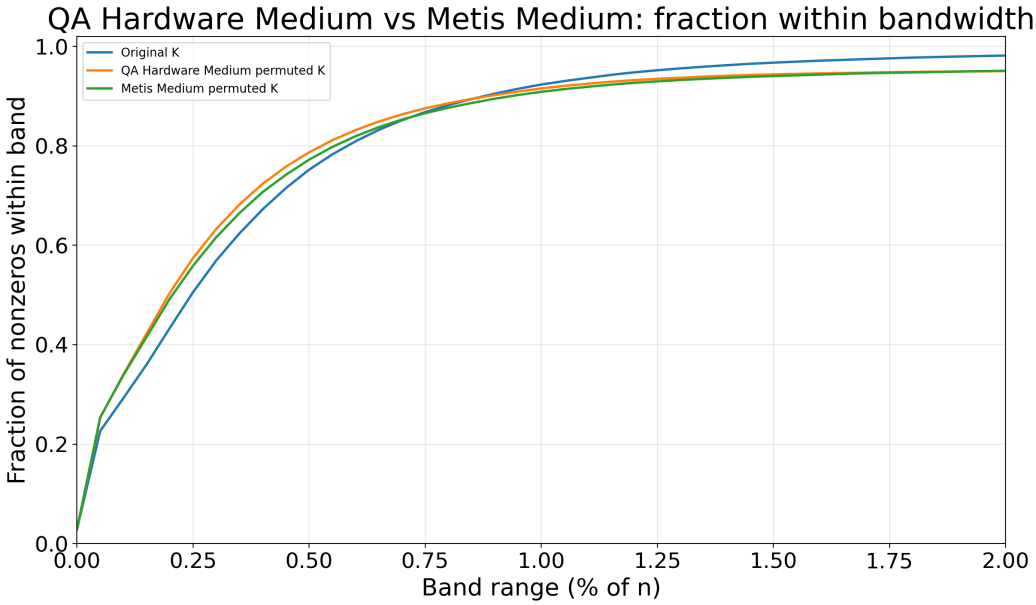
Figure 44 shows that for the small preset, METIS produces the more favorable local-concentration profile. Its K matrix reaches 0.3848 at 0.1% and 0.7795 at 0.4%, whereas QA hardware small reaches 0.3630 and 0.7500. METIS also provides a stronger maximum-bandwidth reduction, from the original 309635 to 258956, while QA hardware small reaches 306515. The visual evidence is therefore consistent with the quantitative curve: in this small-scale comparison, METIS creates clearer local block concentration.

9.4.3 Quantum Annealing Emulated Partitioning

Figures 47, 49 and 25 shows the results of the permutation with big, medium and small coarsen graph respectively.

Figure 48 shows that for the big preset, both methods reduce the maximum bandwidth by a similar amount: METIS big reaches 251021 and QA emulated big reaches 250511. The main difference appears in average bandwidth, where QA emulated big is much lower, 3770.20 compared with 7413.11 for METIS big. METIS big remains stronger in the 0.1% and 0.4% local bands, but QA emulated big provides a better profile-distance average. This result shows a relevant trade-off: one method creates stronger immediate diagonal concentration, while the other reduces the average nonzero distance more effectively.

Figure 24: K-only fraction-within-band comparison for Demonstrator 01: QA hardware medium versus METIS medium.

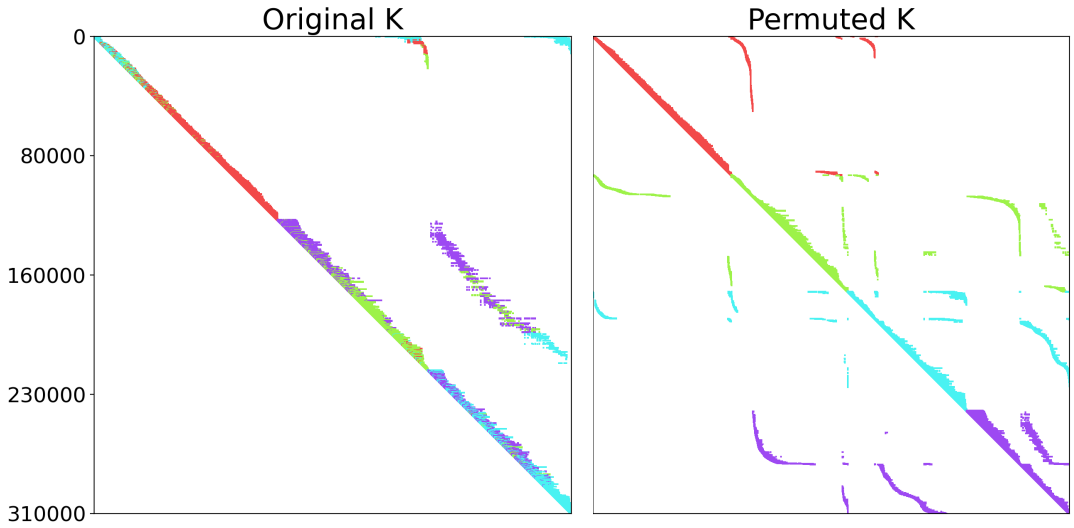


Source: Author (2026)

Figure 50 also shows that the quantum annealing emulated method using a medium coarsen size performed worse than the metis method.

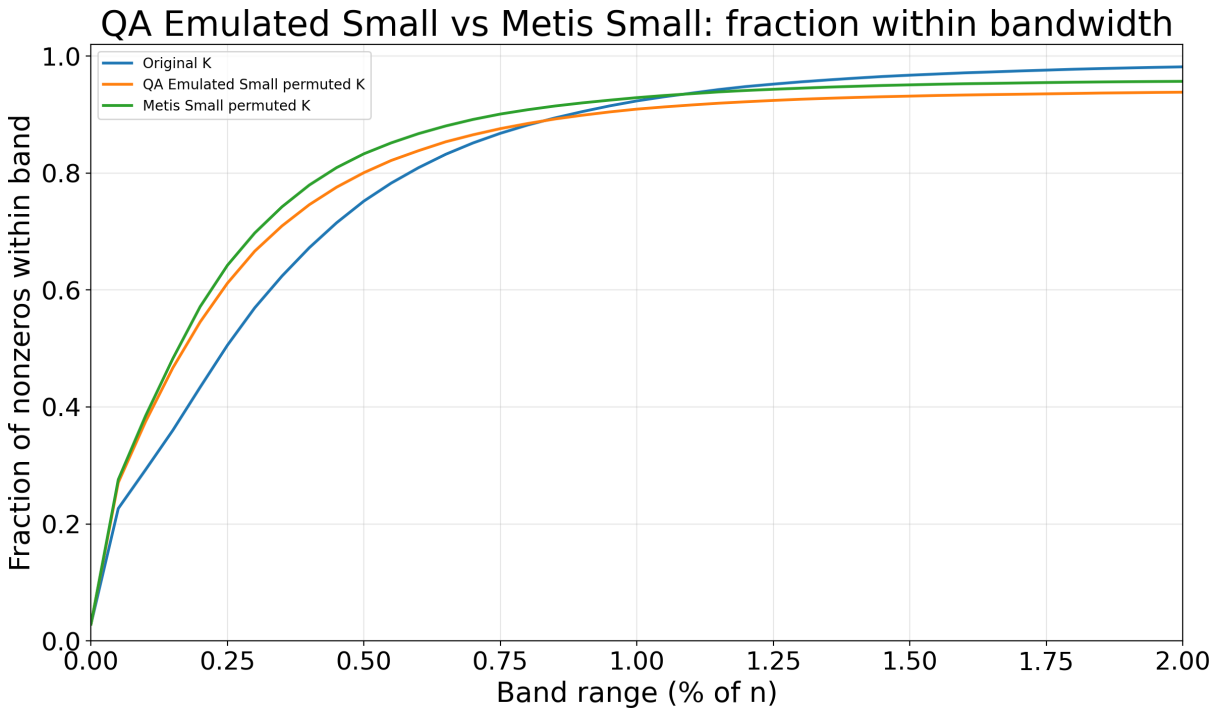
And in figure 26 is shown the small METIS versus small emulated-annealing comparison where illustrates why several metrics are required. QA emulated small provides the strongest K bandwidth reduction in Demonstrator 01, reaching 186398, whereas METIS small reaches 258956. However, METIS small is stronger in the local bands, with 0.3848 at 0.1% and 0.7795 at 0.4%, compared with 0.3749 and 0.7459 for QA emulated small. The matrix views and curves therefore show a genuine trade-off between shrinking the farthest profile entries and maximizing near-diagonal concentration.

Figure 25: Demonstrator 01 K-matrix Quantum Annealing Emulated small coarsen size



Source: Author (2026)

Figure 26: K-only fraction-within-band comparison for Demonstrator 01: QA hardware small versus METIS small.

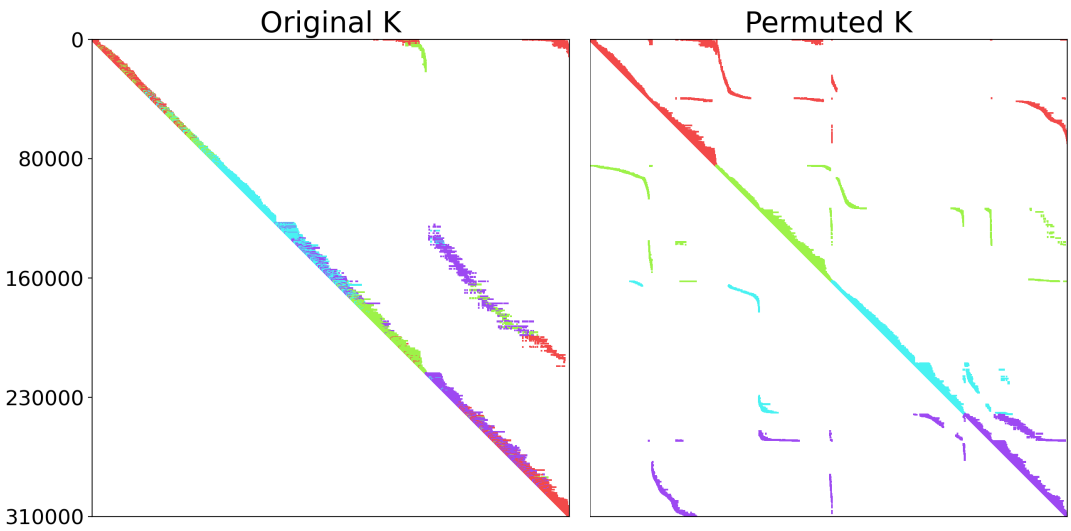


Source: Author (2026)

9.4.4 QAOA Emulated Partitioning

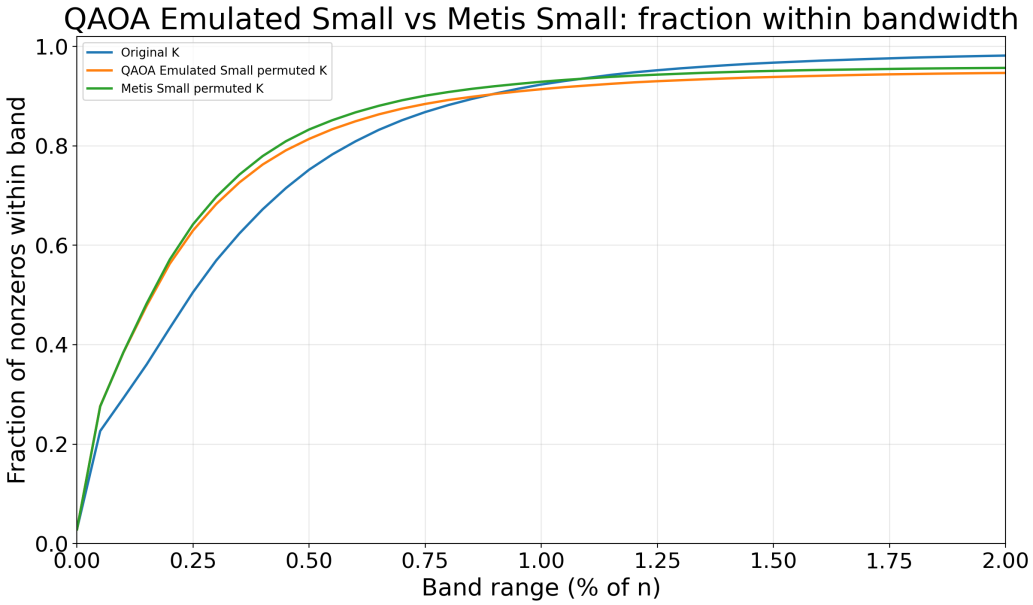
Figure 27 show the original and permuted matrix result using this method for small size.

Figure 27: Demonstrator 01 K-matrix QAOA Emulated small coarsen size



Source: Author (2026)

Figure 28: K-only fraction-within-band comparison for Demonstrator 01: QAOA emulated small versus METIS small.



Source: Author (2026)

9.4.5 Numerical comparison across methods

Table 14: K-matrix numerical summary for Demonstrator 01

Method	**Size**	**Bandwidth**	**Avg. Bandwidth**	$\text{frac}_{0.1\%}$	$\text{frac}_{0.4\%}$	$\text{frac}_{0.7\%}$	$\text{frac}_{1.0\%}$
Original	N/A	309635	2695.46	0.2926	0.6724	0.8515	0.9234
METIS	big	251021	7413.11	0.3318	0.7056	0.8483	0.9019
METIS	medium	310781	6404.33	0.3386	0.7071	0.8529	0.9087
METIS	small	258956	4994.66	0.3848	0.7795	0.8917	0.9289
QA hardware	medium	311540	6661.11	0.3404	0.7237	0.8633	0.9155
QA hardware	small	306515	5680.84	0.3630	0.7500	0.8756	0.9201
QA emulated	big	250511	3770.20	0.3070	0.6815	0.8484	0.9137
QA emulated	medium	283400	5610.28	0.3341	0.6998	0.8482	0.9047
QA emulated	small	186398	6822.64	0.3749	0.7459	0.8655	0.9093
QAOA emulated	small	312230	6706.74	0.3842	0.7625	0.8748	0.9140

Source: Author (2026)

Table 15: M-matrix numerical summary for Demonstrator 01

Method	**Size**	**Bandwidth**	**Avg. Bandwidth**	$\text{frac}_{0.1\%}$	$\text{frac}_{0.4\%}$	$\text{frac}_{0.7\%}$	$\text{frac}_{1.0\%}$
Original	N/A	309633	2633.04	0.3110	0.6797	0.8551	0.9253
METIS	big	251019	7214.44	0.3490	0.7124	0.8521	0.9045
METIS	medium	310779	6228.05	0.3556	0.7138	0.8567	0.9112
METIS	small	258954	4859.02	0.4005	0.7845	0.8944	0.9308
QA hardware	medium	311538	6488.80	0.3573	0.7299	0.8667	0.9177
QA hardware	small	306513	5533.29	0.3793	0.7557	0.8787	0.9221
QA emulated	big	250509	3667.84	0.3249	0.6887	0.8521	0.9160
QA emulated	medium	283398	5470.51	0.3513	0.7067	0.8520	0.9072
QA emulated	small	186396	6644.11	0.3909	0.7518	0.8688	0.9116
QAOA emulated	small	312228	6531.55	0.3999	0.7679	0.8779	0.9162

Source: Author (2026)

Table 16: K-matrix nB^2 metrics for Demonstrator 01

Method	**Size**	**NNZ Diff**	**Max Abs Err**	$\text{Iter}_{\text{original}}$	$\text{Iter}_{\text{permuted}}$	$\text{Iter}_{\text{reduction}}$
METIS	big	0	0.0	3.00e+16	1.97e+16	34.3%
METIS	medium	0	0.0	3.00e+16	3.02e+16	-0.7%
METIS	small	0	0.0	3.00e+16	2.10e+16	30.1%
QA hardware	medium	0	0.0	3.00e+16	3.03e+16	-1.2%
QA hardware	small	0	0.0	3.00e+16	2.94e+16	2.0%
QA emulated	big	0	0.0	3.00e+16	1.96e+16	34.5%
QA emulated	medium	0	0.0	3.00e+16	2.51e+16	16.2%
QA emulated	small	0	0.0	3.00e+16	1.09e+16	63.8%
QAOA emulated	small	0	0.0	3.00e+16	3.05e+16	-1.7%

Source: Author (2026)

Tables 14 and 15 show that the strongest successful bandwidth reduction for Demonstrator 01 is obtained by the QA emulated small case. The K bandwidth is reduced from 309635 to 186398, and the M bandwidth is reduced from 309633 to 186396. This corresponds to a reduction of approximately 39.8% for both matrices. However, the same method is not best in average bandwidth. The QA emulated big case gives the lowest average bandwidth among the reordered cases for both K and M, with 3770.20 and 3667.84, respectively.

The METIS family provides the strongest percentage-band locality for this demonstrator. In the small METIS case, the K matrix reaches 0.3848 at 0.1%, 0.7795 at 0.4%, 0.8917 at 0.7%, and 0.9289 at 1.0%. The same pattern appears for M, where the small METIS case reaches 0.4005, 0.7845, 0.8944, and 0.9308. This indicates that the classical multilevel baseline remains very strong when the evaluation focuses on local diagonal concentration rather than only on the furthest nonzero.

The hardware quantum annealing cases behave competitively but not uniformly. The medium hardware run does not reduce the maximum bandwidth (311 540 for K and 311 538 for M, close to the original 309 635/309 633), but it improves the 0.4 % and 0.7 % locality values over the original matrices. The small hardware run improves the very local 0.1 % and 0.4 % bands more strongly, and it achieves a slightly lower bandwidth (306 515 for K and 306 513 for M). This demonstrates that hardware annealing can produce useful reorderings, but its result quality depends strongly on the selected coarse graph size.

The emulated QAOA small case is especially strong at the narrowest local band: it reaches 0.3842 for K and 0.3999 for M at 0.1%, close to the best METIS values. Nevertheless, it does not reduce the maximum bandwidth for Demonstrator 01. The method therefore concentrates many entries very close to the diagonal while leaving some long-range entries in the profile. This confirms that one backend can be best for maximum-band shrinkage, another can be best for average bandwidth, and another can be competitive in local concentration. No single metric is sufficient to characterize the quality of a permutation.

9.5 Results for Demonstrator 02

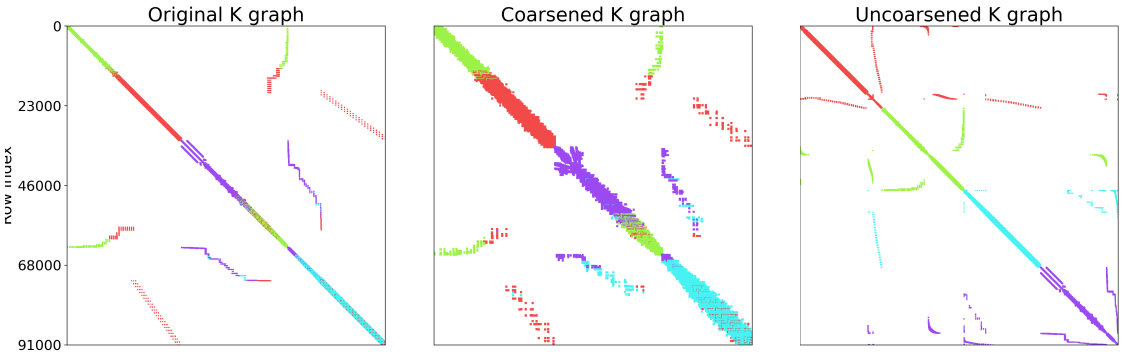
Demonstrator 02 has a denser graph structure because of the base support. The same methods and coarsen sizes where also chosen for this demonstrator. Therefor the same analysis as in the demonstrator 01 will be performed for this second demonstrator.

9.5.1 METIS Partitioning

Figure 29 shows the K-graph evolution for the representative Demonstrator 02 METIS big run. The coarsened graph is visibly different from the corresponding Demonstrator 01

image because the base-supported geometry produces a denser connectivity pattern before coarsening. This denser graph topology affects the coarse partition and later appears again in the matrix-profile metrics.

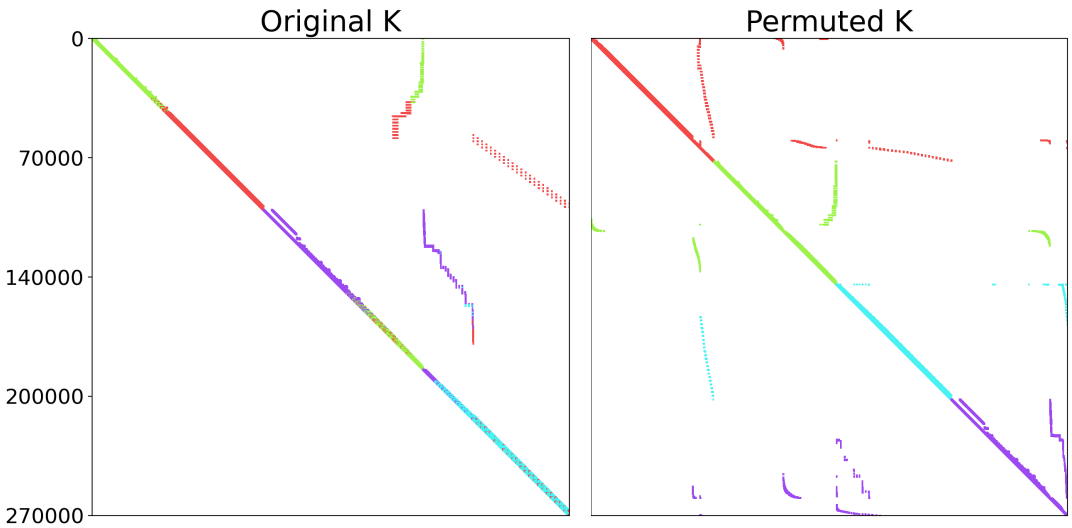
Figure 29: Partition-colored K-graph views for Demonstrator 02: initial graph, coarsened graph, and uncoarsened graph.



Source: Author (2026)

As in Demonstrator 01, only the K-stage sequence is shown. This is sufficient for the visual explanation because the graph partition is computed from K and the recovered permutation is then applied consistently to both K and M.

Figure 30: Demonstrator 02 K-matrix METIS big coarsen size



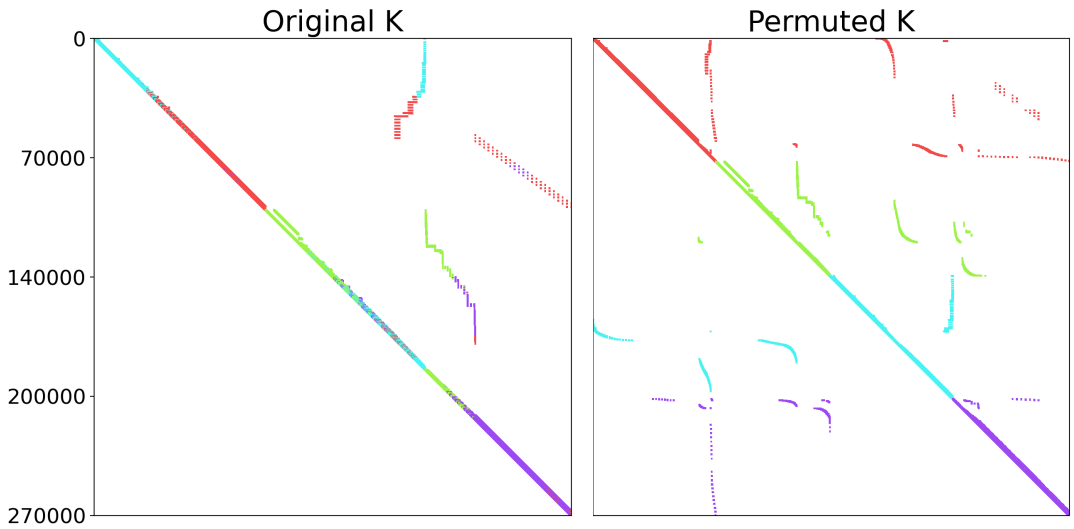
Source: Author (2026)

9.5.2 Quantum Annealing Hardware

Medium and small permutation results for this method are shown in figures 45 and 31 respectively.

The medium preset provides the most informative hardware comparison for the base-supported geometry. QA hardware medium reaches a bandwidth of 250 784, which is larger than both the original, 189 587, and METIS medium, 215 378. Its average bandwidth is 5 715.28, also above the METIS medium value of 5 422.92. METIS medium is higher in the 0.1 % band, 0.3499 versus 0.3457, and slightly higher at 0.4 %, 0.4743 versus 0.4723, but QA hardware medium is stronger at 1.0 %, 0.9133 versus 0.9106. This suggests that hardware annealing on the medium coarse graph of the denser demonstrator does not achieve bandwidth reduction, although it provides a marginal wide-band concentration benefit.

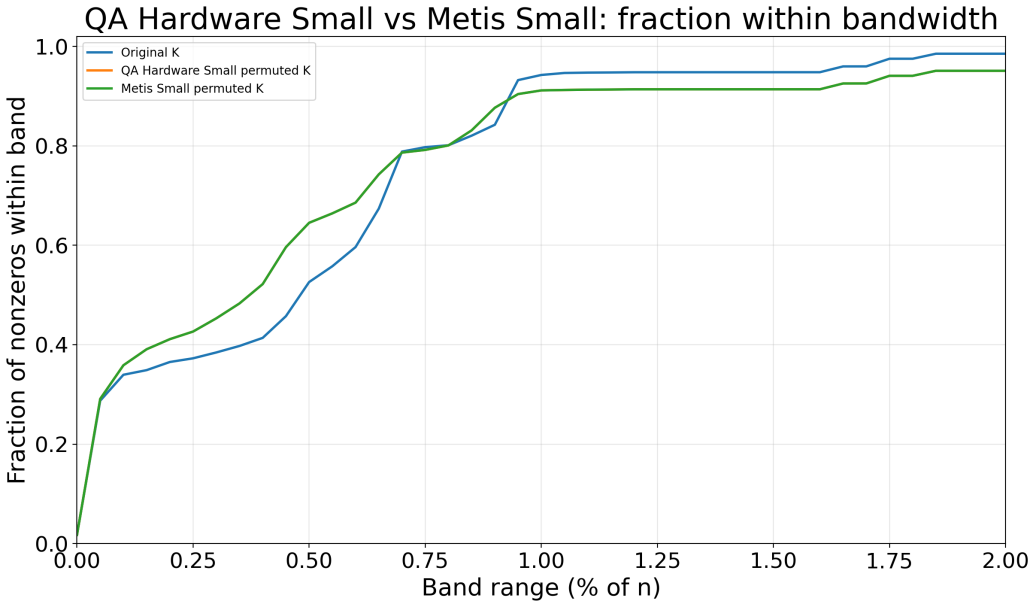
Figure 31: Demonstrator 02 K-matrix Quantum Annealing Hardware small coarsen size



Source: Author (2026)

For Demonstrator 02, the small METIS and small QA hardware runs produce the same local K concentration values in the reported bands: 0.3586 at 0.1% and 0.5219 at 0.4%. Their visual matrix structures are therefore nearly indistinguishable in terms of local block concentration. Neither method improves the original maximum bandwidth of 189587; METIS small reaches 201641 and QA hardware small reaches 212735. This comparison is useful because it shows convergence to a similar very small coarse-level partition rather than identifying a clear winner.

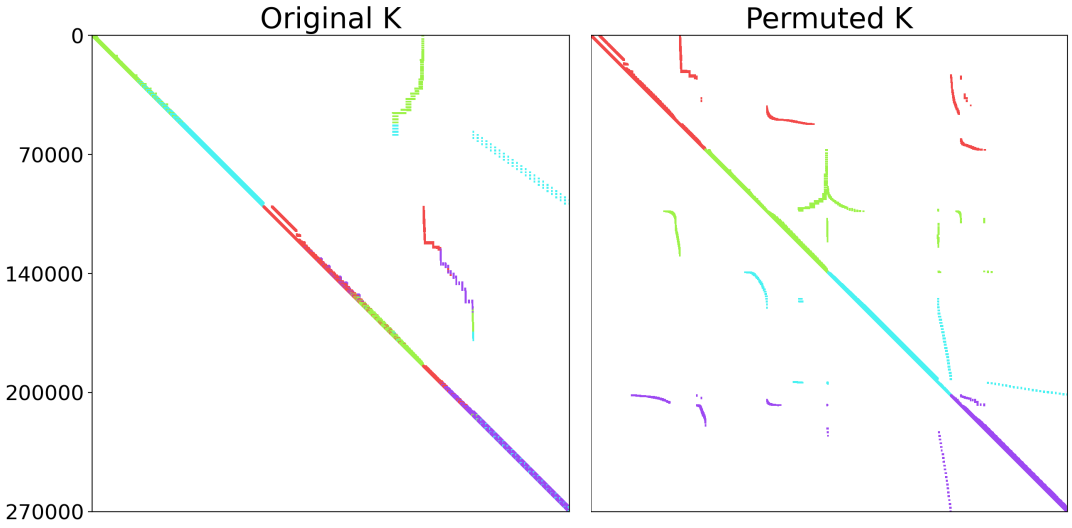
Figure 32: K-only fraction-within-band comparison for Demonstrator 02: QA hardware small versus METIS small.



Source: Author (2026)

9.5.3 Quantum Annealing Emulated Partitioning

Figure 33: Demonstrator 02 K-matrix Quantum Annealing Emulated medium coarsen size

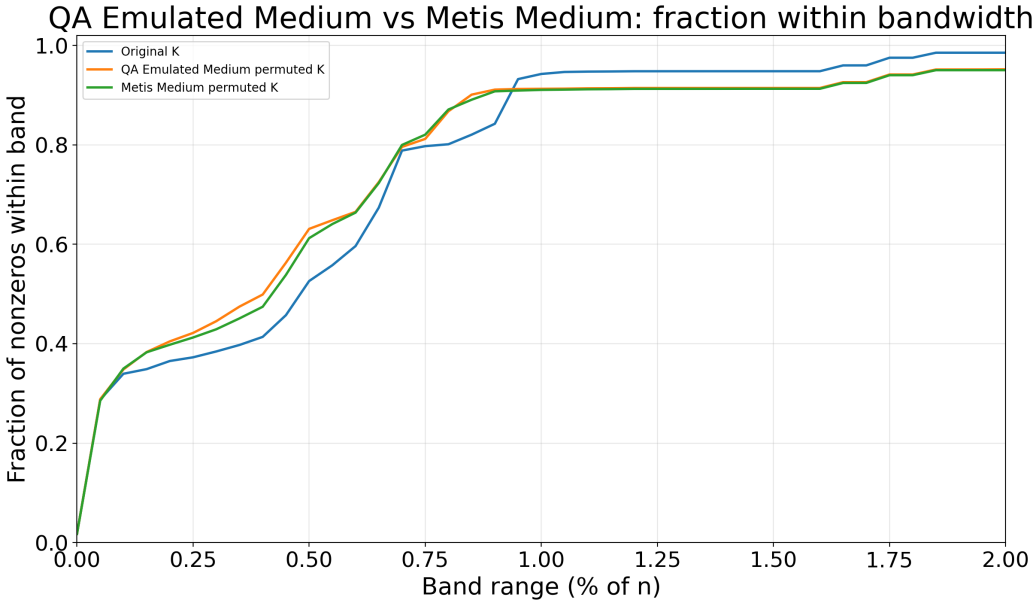


Source: Author (2026)

The medium preset for the Quantum Annealing Emulated method presents a very similar performance comparing with the METIS for the same size. With the fraction 0.1% of

0.3482 for the QA Emulated and 0.3499 for the metis. And for the 1.0% fraction the QA Emulated had 0.9127 and metis 0.9106.

Figure 34: K-only fraction-within-band comparison for Demonstrator 02: QA emulated medium versus METIS medium.

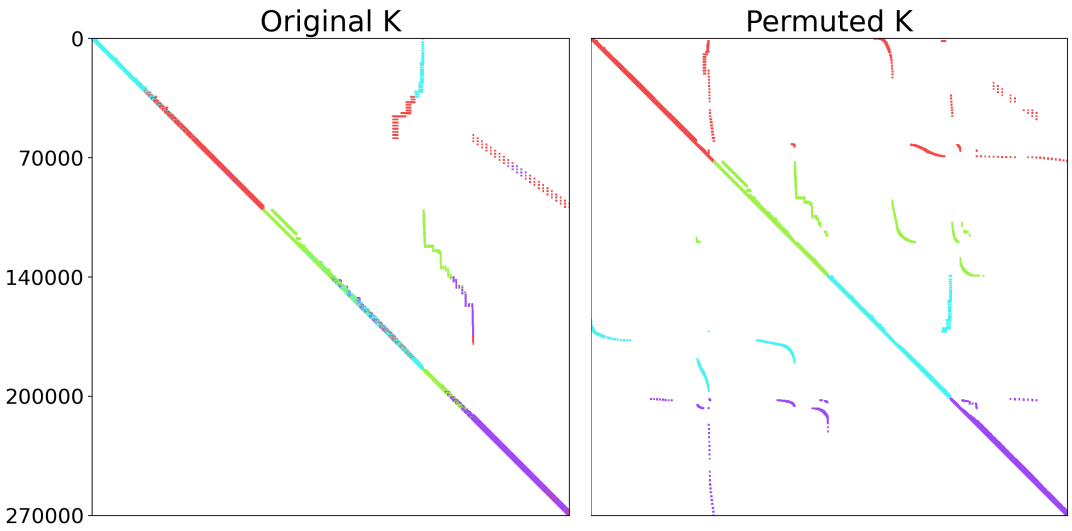


Source: Author (2026)

The small METIS versus small emulated-annealing comparison again produces identical narrow-band locality values in the reported columns, with 0.3586 at 0.1% and 0.5219 at 0.4% for both methods. The bandwidth values differ, with METIS small at 201641 and QA emulated small at 212735, but both remain above the original bandwidth. This reinforces the interpretation that Demonstrator 02 is dominated by geometric density: several backends recover the same small coarse partition, yet none of these small runs produces the maximum-bandwidth improvement observed for Demonstrator 01.

9.5.4 QAOA Emulated Partitioning

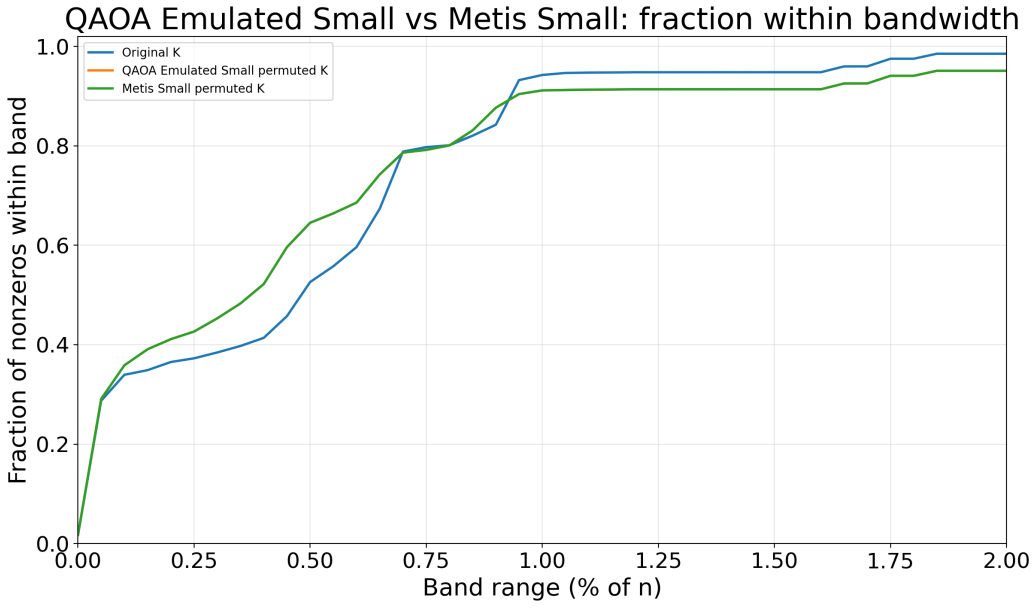
Figure 35: Demonstrator 02 K-matrix QAOA Emulated small coarsen size



Source: Author (2026)

The QAOA Emulated with the small preset has almost the same results for the matrix metrics as the Quantum Annealing emulated for the same size. With also the curve fraction band with the identical same values as the metis with the same preset.

Figure 36: K-only fraction-within-band comparison for Demonstrator 02: QAOA emulated small versus METIS small.



Source: Author (2026)

9.5.5 Numerical comparison across methods

Table 17: K-matrix numerical summary for Demonstrator 02

Method	**Size**	**Bandwidth**	**Avg. Bandwidth**	$\text{frac}_{0.1\%}$	$\text{frac}_{0.4\%}$	$\text{frac}_{0.7\%}$	$\text{frac}_{1.0\%}$
Original	N/A	189587	2938.74	0.3395	0.4137	0.7885	0.9426
METIS	big	214238	6331.55	0.3414	0.4545	0.7978	0.9106
METIS	medium	215378	5422.92	0.3499	0.4743	0.7998	0.9106
METIS	small	201641	6241.72	0.3586	0.5219	0.7864	0.9115
QA hardware	medium	250784	5715.28	0.3457	0.4723	0.7964	0.9133
QA hardware	small	212735	5648.64	0.3586	0.5219	0.7864	0.9115
QA emulated	big	244265	5812.89	0.3445	0.4688	0.8039	0.9174
QA emulated	medium	188486	4661.49	0.3482	0.4989	0.7959	0.9127
QA emulated	small	212735	5648.64	0.3586	0.5219	0.7864	0.9115
QAOA emulated	small	212735	5648.64	0.3586	0.5219	0.7864	0.9115

Source: Author (2026)

Table 18: M-matrix numerical summary for Demonstrator 02

Method	**Size**	**Bandwidth**	**Avg. Bandwidth**	$\text{frac}_{0.1\%}$	$\text{frac}_{0.4\%}$	$\text{frac}_{0.7\%}$	$\text{frac}_{1.0\%}$
Original	N/A	189585	2667.45	0.3984	0.4710	0.8139	0.9491
METIS	big	214236	5979.42	0.3991	0.5061	0.8192	0.9175
METIS	medium	215376	5092.02	0.4079	0.5243	0.8208	0.9176
METIS	small	201639	5851.24	0.4166	0.5674	0.8094	0.9187
QA hardware	medium	250782	5416.56	0.4032	0.5225	0.8182	0.9201
QA hardware	small	212733	5321.51	0.4166	0.5674	0.8094	0.9187
QA emulated	big	244263	5431.39	0.4030	0.5194	0.8250	0.9243
QA emulated	medium	188484	4295.40	0.4061	0.5462	0.8182	0.9201
QA emulated	small	212733	5321.51	0.4166	0.5674	0.8094	0.9187
QAOA emulated	small	212733	5321.51	0.4166	0.5674	0.8094	0.9187

Source: Author (2026)

Table 19: K-matrix metrics for Demonstrator 02

Method	**Size**	**NNZ Diff**	**Max Abs Err**	$\text{Iter}_{\text{original}}$	$\text{Iter}_{\text{permuted}}$	$\text{Iter}_{\text{reduction}}$
METIS	big	0	0.0	9.82e+15	1.25e+16	-27.7%
METIS	medium	0	0.0	9.82e+15	1.27e+16	-29.1%
METIS	small	0	0.0	9.82e+15	1.11e+16	-13.1%
QA hardware	medium	0	0.0	9.82e+15	1.72e+16	-75.0%
QA hardware	small	0	0.0	9.82e+15	1.24e+16	-25.9%
QA emulated	big	0	0.0	9.82e+15	1.63e+16	-66.0%
QA emulated	medium	0	0.0	9.82e+15	9.71e+15	1.2%
QA emulated	small	0	0.0	9.82e+15	1.24e+16	-25.9%
QAOA emulated	small	0	0.0	9.82e+15	1.24e+16	-25.9%

Source: Author (2026)

Tables 17 and 18 show behavior that differs from Demonstrator 01. The original bandwidth is already much smaller, at 189587 for K and 189585 for M. The best successful

bandwidth result is obtained by the QA emulated medium case, which reaches 188486 for K and 188484 for M. This is a marginal improvement of about 0.58%, far smaller than the reduction observed in Demonstrator 01. The same QA emulated medium run also gives the best average bandwidth among the reordered cases for both matrices.

For METIS, the small case gives the strongest local concentration at the 0.1% and 0.4% bands, reaching 0.3586 and 0.5219 for K and 0.4166 and 0.5674 for M. The same values are also obtained by the small QA hardware, QA emulated, and QAOA emulated cases in this campaign, showing that several backends converge to the same very small coarse-level partition for the base-supported demonstrator. However, at the wider 0.7% and 1.0% bands, QA emulated big is strongest among the reordered cases, with 0.8039 and 0.9174 for K and 0.8250 and 0.9243 for M.

The hardware annealing medium case is also relevant. It does not beat the original maximum bandwidth, and its bandwidth (250 784) is larger than those of the small hardware (212 735) and small METIS (201 641) runs. Nevertheless, it improves the 0.4 % local concentration over the original (0.4723 versus 0.4137), indicating that the D-Wave hardware path produces a structurally meaningful partition even when bandwidth reduction is not achieved.

The 1.0% columns also show an important limitation of interpreting locality metrics independently. The original Demonstrator 02 matrices already have high 1.0% concentration, and most reorderings reduce this value while improving the very narrow 0.1% and 0.4% bands. Therefore, the base-supported geometry benefits more from local redistribution very close to the diagonal than from a uniform improvement across all band widths. The maximum bandwidth alone underestimates some structural benefits, but the percentage-band metrics also show that improvements can be scale-dependent.

9.6 Cross-Demonstrator Comparison

The comparison between both demonstrators shows that the multilevel pipeline reacts strongly to geometry-dependent graph structure. Demonstrator 01 has more graph nodes but fewer graph edges, and it offers considerably more freedom to reduce the maximum bandwidth. Demonstrator 02 has fewer nodes but almost twice as many graph edges, and this denser connectivity limits the achievable maximum-band reduction.

For Demonstrator 01, the best successful run using the QA Emulated method with coarsen size small reduced the maximum bandwidth by approximately 39.8% for both K and M and the number of iterations had a reduction of 63.8%. This indicates that the blade geometry without base support offers significant room for structural improvement through graph-based permutation. The results also showed that the method with the best maximum-bandwidth reduction was not always the method with the best fixed-band concentration, confirming that permutation quality cannot be reduced to a single metric.

For Demonstrator 02, the denser base-supported geometry led to a different behavior. Using the QA Emulated with coarsen size medium, the K and M bandwidth could only be reduced marginally by 0.6 % and 1.2% reduction on the number of iterations, but several methods improved the local concentration of nonzero around the diagonal. This demonstrates that geometry and graph density strongly influence the effectiveness of the same multilevel strategy.

The K and M matrices follow similar trends within each demonstrator, but the mass matrix tends to have higher local concentration in the narrow percentage bands. For Demonstrator 01, the best M value at 0.1% is 0.4005 compared with 0.3848 for K, and at 0.4% it is 0.7845 compared with 0.7795 for K. For Demonstrator 02, the same behavior is stronger: the original M value at 0.1% is 0.3984 compared with 0.3395 for K, and the best reordered M value at 0.4% is 0.5674 compared with 0.5219 for K. This suggests that the mass matrix structure is more locally concentrated under the same ordering, even though the partition is computed from the stiffness graph.

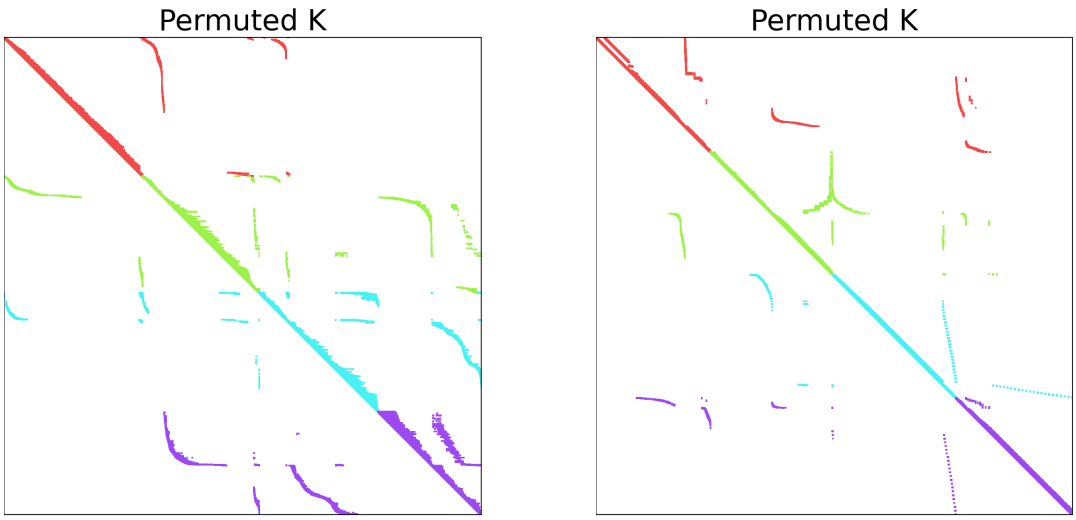
The results also show that smaller coarse graphs do not automatically imply better final permutations. In Demonstrator 01, the small QA emulated case gives the best maximum-bandwidth reduction, the big QA emulated case gives the best average bandwidth, and the small METIS case gives the best percentage-band concentration. In Demonstrator 02, the best bandwidth and average bandwidth are obtained at the medium QA emulated level, while several small cases tie for the best 0.1% and 0.4% concentration. Coarse size should therefore be interpreted as a control parameter whose effect depends on both the geometry and the backend.

Across the method families, METIS remains the strongest baseline for local concentration in several narrow bands, especially in the small setting. Hardware quantum annealing produced valid and competitive partitions. In both demonstrators the small hardware cases achieve lower maximum bandwidths than the medium hardware cases, indicating that deeper coarsening helps the hardware backend produce tighter orderings. Emulated quantum annealing produced the best maximum-bandwidth results in both demonstrators, although not always the best local concentration. Emulated QAOA was competitive in very narrow bands but did not dominate the maximum-bandwidth metric. This confirms that the quantum and quantum-inspired backends should be evaluated as part of a full structural trade-off, not only by a single scalar score.

Using maximum-bandwidth reduction as the primary selection criterion, Figure 37 shows the best K-matrix original-versus-permuted comparisons from the campaign. Under this criterion, Demonstrator 01 selects QA emulated small, which reduces the K bandwidth from 309635 to 186398. Demonstrator 02 selects QA emulated medium, which gives the best reordered K bandwidth from 189 587 to 188 486 and average bandwidth among the completed cases, although the improvement over the original bandwidth is only marginal, approximately 0.6 %. The figure therefore summarizes the

main cross-demonstrator conclusion: emulated quantum annealing is the most successful family for bandwidth reduction, while METIS remains highly competitive when the objective is local diagonal concentration.

Figure 37: Best-method K original-versus-permuted comparisons by maximum-bandwidth criterion: Demonstrator 01 with QA emulated small on the left and Demonstrator 02 with QA emulated medium on the right.



Source: Author (2026)

From the solver perspective, the main conclusion is structural rather than spectral. Equation 9.2 preserves the original eigenvalues, so the tables do not prove a direct reduction of $\kappa(A)$. They instead show that the matrices can be reorganized into patterns that are more suitable for constructing localized preconditioners. If the reordered matrix allows a preconditioner to use a smaller effective bandwidth p, Equation 2.21 indicates that the factorization cost can decrease significantly. If that preconditioner also improves $\kappa(M^{-1}A)$, Equation 2.24 indicates a reduction in the expected number of Krylov iterations. The present results therefore support the thesis hypothesis by showing the structural precondition for such improvements, while direct solver timings and explicit condition-number measurements remain a necessary future validation step.

9.7 Experimental Limitations

The hardware QAOA cases were not included in the numerical comparison tables because the original comparison batch did not produce completed result folders for those submissions. The logs indicate backend or scheduler-side submission problems rather than failures in graph construction, coarsening, recursive bisection, uncoarsening, or matrix

permutation. These cases are therefore treated as infrastructure limitations of remote quantum experimentation.

The results are also structural metrics, not direct solver measurements. Bandwidth, average bandwidth, Krylov iteration count and percentage-band locality fractions are useful indicators of matrix locality, but they do not replace direct measurements of preconditioner setup time, memory consumption, condition number of the preconditioned operator. Nevertheless, the batch campaign provides a consistent and informative evaluation of the proposed workflow. Across both demonstrators, the method preserved the nonzero-pattern cardinality after permutation, produced valid reorderings, and revealed meaningful geometry-dependent trade-offs between maximum-band reduction and local diagonal concentration.

10 CONCLUSION

This thesis investigated the use of hybrid quantum-classical graph partitioning for domain decomposition and matrix permutation in finite element models. The central motivation was that the cost of solving large sparse systems is not determined only by the number of unknowns, but also by the matrix structure seen by direct solvers, incomplete factorizations, block preconditioners, and Krylov methods. By reorganizing the degrees of freedom according to graph partitions, the matrix can be transformed into a form that is more suitable for localized preconditioning.

10.1 Main Contributions

The first contribution of this work is the implementation of an end-to-end multilevel domain-decomposition pipeline. The workflow converts FEM matrices into graph representations, applies heavy-edge-based coarsening, partitions the coarse graph, uncoarsens and refines the partition, and finally applies the recovered ordering to both the stiffness matrix K and the mass matrix M. This connects the theoretical graph-partitioning discussion to concrete matrix outputs.

The second contribution is the integration of several partitioning strategies into the same workflow. Classical METIS, simulated quantum annealing, hardware quantum annealing, local QAOA emulation, and remote QAOA execution were treated through a common interface. This made it possible to compare the effect of backend choice under the same coarsening and profiling methodology.

The third contribution is the development of a result-analysis layer based on matrix profiling. The thesis used bandwidth, Avg. Bandwidth, Krylov iteration count and fraction of nonzeros within percentage-defined diagonal bands to evaluate the effect of permutations. These metrics are structural proxies for preconditioner friendliness and were used consistently across the two demonstrators.

Finally, the thesis produced partition-colored matrix visualizations that connect the graph-level workflow to the final matrix layout. These figures make the effect of coarsening, coarse partitioning, uncoarsening, and matrix permutation visible in a way that complements the numerical tables.

10.2 Main Findings

The experiments showed that valid matrix reorderings can be produced consistently for both demonstrators. The permutation preserves the matrix entries and the nonzero-pattern cardinality, while changing the location of nonzero entries in the matrix profile.

This confirms that the implemented workflow behaves as a valid symmetric permutation process.

Table 16 and 19 shows at the NNZ Diff (Difference on the number of non-zeros) that after applying the permutation, the permuted matrices was kept with the same number of nonzero and in Max Abs Err (Maximum absolute error) shows that all these nonzero was kept with the same value, preserving the whole matrices eigenvalues. Column Iter % Reduction shows the difference on the number of iterations that the worst case scenario solver would have according to equation 2.24. This results highlights that the matrix permutation were able while maintaining the physicality of the matrix, to enhance the performance of the matrix preconditioning.

These findings support the thesis hypothesis that graph partitioning can produce matrix structures that are more suitable for preconditioned solvers. However, the results should be interpreted as structural evidence rather than final solver-performance proof. Because a symmetric permutation $A_p = PAP^T$ preserves the eigenvalues of A, the permutation alone does not reduce the original condition number. The expected benefit appears when the reordered structure enables a better or cheaper preconditioner, thereby improving the effective condition number $\kappa(M^{-1}A)$ discussed in Section 2.2.3.2 and Equation 2.24.

10.3 Limitations

The first limitation is that the thesis evaluates structural matrix metrics, not full solver performance. Bandwidth and diagonal concentration are relevant for preconditioner construction, but they do not directly measure preconditioner setup time, memory consumption, Krylov iteration count, or final wall-clock performance.

The second limitation is that quantum hardware access remains infrastructure-dependent. The quantum annealing hardware runs completed for the tested cases, but remote QAOA hardware submissions were affected by backend or scheduler-side issues. These failures reflect the practical difficulty of using remote quantum resources in automated numerical workflows.

The third limitation is that only two demonstrators were analyzed. Although they are useful because they share the same physical context and differ by the base support, a broader set of geometries is required before the observed trends can be generalized.

10.4 Future Work

Future studies should first connect the structural metrics to actual solver performance. The next experimental step is to build preconditioners after permutation and measure the resulting setup time, memory cost, factorization cost, and Krylov iteration count.

The coarsening and refinement stages should also be studied more deeply. Different coarsening limits, edge-weight definitions, balance tolerances, and refinement objectives may produce substantially different matrix profiles. Automated hyperparameter search could be used to identify configurations that optimize bandwidth, local concentration, or direct solver performance.

The quantum backends require further investigation as well. Future work should evaluate the scaling limits of hardware quantum annealing and QAOA, quantify remote submission cost, compare queue time with solution quality, and determine which coarse graph sizes are practically meaningful for available devices. Persisting intermediate partition artifacts would also make post-hoc visualization and debugging easier without requiring a full rerun.

Finally, the demonstrator set should be expanded. Additional geometries, mesh densities, material models, and boundary conditions would make it possible to determine when hybrid graph partitioning is most beneficial and when classical methods already provide sufficient ordering quality.

10.5 Final Remarks

The work developed in this thesis shows that hybrid multilevel graph partitioning is a viable preprocessing strategy for restructuring finite element matrices. The strongest result is not that every geometry obtains a large maximum-bandwidth reduction, but that the workflow exposes controllable structural changes that can be analyzed, compared, and connected to preconditioning theory. This establishes a foundation for future solver-level studies in which the matrix permutations produced here are used directly inside preconditioned linear-solver pipelines.

BIBLIOGRAPHY

Albash, T. and Lidar, D. A. (2018). Adiabatic quantum computation. *Reviews of Modern Physics*, 90:015002.

Aramon, M., Rosenberg, G., Valiante, E., Miyazawa, T., Tamura, H., and Katzgraber, H. G. (2019). Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. *Frontiers in Physics*, 7:48.

Arrazola, P. J., Özel, T., Umbrello, D., Davies, M., and Jawahir, I. S. (2013). Recent advances in modelling of metal machining processes. *CIRP Annals*, 62(2):695–718.

Bass, G., Henderson, M. P., Heath, J., and Dulny, J. (2021). Optimizing the optimizer: decomposition techniques for quantum annealing. *Quantum Machine Intelligence*, 3(1):1–14.

Boixo, S., Smelyanskiy, V. N., Shabani, A., et al. (2016). Computational multiqubit tunnelling in programmable quantum annealers. *Nature Communications*, 7:10327.

Booth, M., Reinhardt, S. P., and Roy, A. (2017). Partitioning optimization problems for hybrid classical/quantum execution. Technical report, D-Wave Systems Inc.

Bramble, J. H. (1986). The construction of preconditioners for elliptic problems by substructuring. In Hennart, J.-P., editor, *Numerical Analysis*, pages 158–166, Berlin, Heidelberg. Springer Berlin Heidelberg.

Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., and Schulz, C. (2013). Recent advances in graph partitioning. *CoRR*, abs/1311.3144.

Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, K., McClean, J. R., Mitarai, K., Yuan, X., Cincio, L., and Coles, P. J. (2021). Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644.

Champaney, L., Cognard, J. Y., Dureisseix, D., and Ladevèze, P. (1997). Large scale applications on parallel computers of a mixed domain decomposition method. *Computational Mechanics*, 19(3):253–263.

Choi, V. (2011). Minor-embedding in adiabatic quantum computation: Ii. minor-universal graph design. *Quantum Information Processing*, 10(3):343–353.

Cimino, C., Negri, E., and Fumagalli, L. (2019). Review of digital twin applications in manufacturing. *Computers in Industry*, 113:103130.

Cook, R. D., Malkus, D. S., Plesha, M. E., and Witt, R. J. (2002). Concepts and applications of finite element analysis. In *Concepts and Applications of Finite Element Analysis*, New York. Wiley.

Coppersmith, D. (1994). An approximate fourier transform useful in quantum factoring. IBM Research Report (also available as arXiv:quant-ph/0201067).

Crosson, E. and Deng, M. (2014). Tunneling through high energy barriers in simulated quantum annealing. *arXiv preprint arXiv:1410.8484*.

D-Wave Systems Inc. (2020). D-wave hybrid solver service: An overview. White paper, D-Wave Systems Inc.

D-Wave Systems Inc. (2023). *Ocean Software Suite*. `https://docs.ocean.dwavesys.com/en/stable/`.

D-Wave Systems Inc. (2024). *Errors and Error Correction — D-Wave Quantum Computing Products documentation*. Available at D-Wave Documentation: `https://docs.dwavequantum.com/en/latest/quantum_research/errors.html`.

Danz, S., Berta, M., Schröder, S., Kienast, P., Wilhelm, F. K., and Ciani, A. (2025). Calculating response functions of coupled oscillators using quantum phase estimation. *Physical Review Research*, 7(2).

Fadiji, T., Coetzee, C. J., Berry, T. M., Ambaw, A., and Opara, U. L. (2018). The efficacy of finite element analysis (fea) as a design tool for food packaging: A review. *Biosystems Engineering*, 174:20–40.

Farhi, E., Goldstone, J., and Gutmann, S. (2014). A quantum approximate optimization algorithm.

Fuhrmann, J. (2024). Iterative methods for linear systems. Advanced Topics from Scientific Computing, Notebook 08, TU Berlin Winter 2024/25.

Gardas, B., Dziarmaga, J., Zurek, W. H., and Zwolak, M. (2018). Defects in quantum computers. *Scientific Reports*, 8(1):4539.

Garey, M., Johnson, D., and Stockmeyer, L. (1976). Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237–267.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Georgescu, I. M., Ashhab, S., and Nori, F. (2014). Quantum simulation. *Reviews of Modern Physics*, 86(1):153–185.

Gilyén, A., Su, Y., Low, G. H., and Wiebe, N. (2018). Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics.

Golub, G. H. and Van Loan, C. F. (2013). *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, 4th edition.

Griffiths, R. B. and Niu, C.-S. (1995). Semiclassical fourier transform for quantum computation. *Physical Review Letters*, 76(17):3228–3231.

Gropp, W. D. and Keyes, D. E. (1989). Domain decomposition on parallel computers. *IMPACT of Computing in Science and Engineering*, 1(4):421–439.

Hadfield, S., Wang, Z., O'Gorman, B., Rieffel, E. G., Venturelli, D., and Biswas, R. (2019). From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34.

Hendrickson, B. and Kolda, T. G. (1999). Graph partitioning models for parallel computing. *Parallel Computing*, 26(12):1519–1534. Graph Partitioning and Parallel Computing.

Herrera-Revilla, I., Contreras, I., and Herrera, G. S. (2019). The divide-and-conquer framework: A suitable setting for the ddm of the future.

Johnson, M. W., Amin, M. H., Gildert, S., Lanting, T., Hamze, F., Dickson, N., Harris, R., Berkley, A. J., Johansson, J., Bunyk, P., et al. (2011). Quantum annealing with manufactured spins. *Nature*, 473:194–198.

Jordan, H. F. (1987). Interpreting parallel processor performance measurements. *SIAM Journal on Scientific and Statistical Computing*, 8(2):s220–s226.

Kadowaki, T. and Nishimori, H. (1998). Quantum annealing in the transverse ising model. *Physical Review E*, 58(5):5355–5363.

Karypis, G. and Kumar, V. (1995). Metis – unstructured graph partitioning and sparse matrix ordering system, version 2.0. *Technical report*.

Karypis, G. and Kumar, V. (1998a). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392.

Karypis, G. and Kumar, V. (1998b). Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129.

Kato, T. (1950). On the adiabatic theorem of quantum mechanics. *Journal of the Physical Society of Japan*, 5(6):435–439.

King, A. D. et al. (2022). Coherent quantum annealing in a programmable 2,000 qubit ising chain. *Nature Physics*, 18(11):1324–1328.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science (New York, N.Y.)*, 220:671–80.

Kitaev, A. Y. (1995). Quantum measurements and the abelian stabilizer problem. *arXiv preprint quant-ph/9511026*.

Klocke, F. and Straube, A. M. (2004). Virtual process engineering: An approach to integrate vr, fem, and simulation tools in the manufacturing chain. *Mécanique & Industries*, 5(2):199–205.

Kochenberger, G., Hao, J.-K., Glover, F., Lewis, M., Lü, Z., Wang, H., and Wang, Y. (2014). The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization*, 28(1):58–81.

Kumlien, W. (2024). Enhancing rate of convergence in gmres using incomplete lu factorizations. Degree project, Lund University, Faculty of Science.

Low, G. H. and Chuang, I. L. (2019). Hamiltonian simulation by qubitization. *Quantum*, 3:163.

Lucas, A. (2014). Ising formulations of many np problems. *Frontiers in Physics*, 2:5.

Negri, E., Fumagalli, L., and Macchi, M. (2017). A review of the roles of digital twin in cps-based production systems. *Procedia Manufacturing*, 11:939–948.

Nielsen, M. A. and Chuang, I. L. (2010). *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press.

Nikishkov, G. P. (2004). Introduction to the finite element method. Lecture Notes, University of Aizu. Available at: niki@u-aizu.ac.jp.

Pelofske, E. (2025). Comparing three generations of d-wave quantum annealers for minor embedded combinatorial optimization problems. *Quantum Science and Technology*, 10(2):025025.

Quinton, F. A., Myhr, P. A. S., Barani, M., del Granado, P. C., and Zhang, H. (2025). Quantum annealing applications, challenges and limitations for optimisation problems compared to classical solvers. *Scientific Reports*, 15:12733.

Sanders, P. and Schulz, C. (n.d.). KaHIP – Karlsruhe High Quality Partitioning. `http://algo2.iti.kit.edu/documents/kahip/index.html`. Accessed: 2026-02-05.

Smith, B. F., Bjørstad, P. E., and Gropp, W. D. (1996). *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press.

Srirekha, A. and Bashetty, K. (2010). Infinite to finite: An overview of finite element analysis. *Indian Journal of Dental Research*, 21(3):425–432.

Su, P. S. and Fulton, R. E. (1993). A parallel domain decomposition finite element method for massively parallel computers. *Computing Systems in Engineering*, 4(4-6):489–494.

Toselli, A. and Widlund, O. (2005). *Domain Decomposition Methods: Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer.

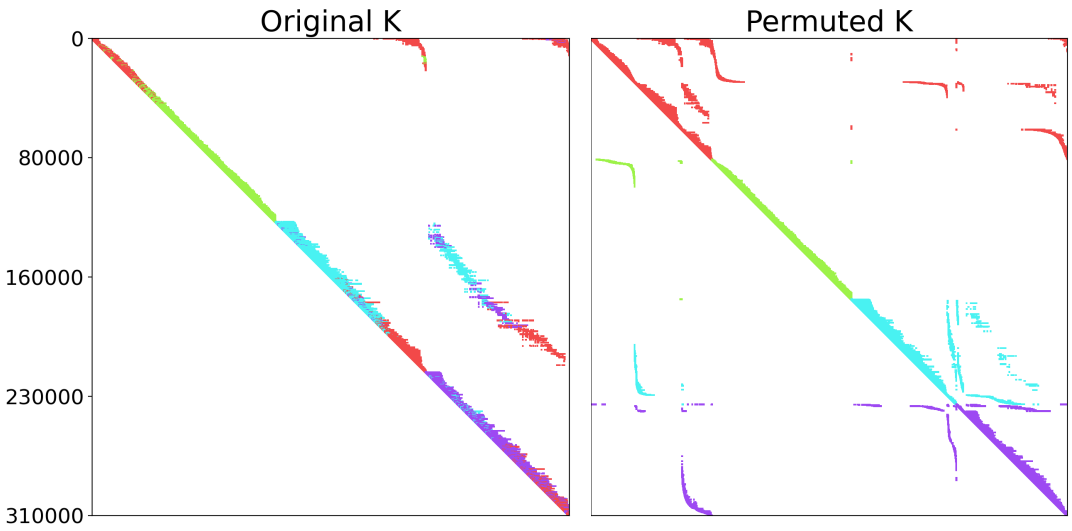
Tripathi, R., Tomar, S., and Kumar, S. (2025). A comprehensive survey on quantum annealing: Applications, challenges, and future research directions.

Ushijima-Mwesigwa, H., Negre, C. F. A., and Mniszewski, S. M. (2017). Graph partitioning using quantum annealing on the d-wave system. In *Proceedings of the Second International Workshop on Post-Moore's Era Supercomputing (PMES'17)*, pages 22–29. ACM.

Weber, S. J., Samach, G. O., Hover, D., Gustavsson, S., Kim, D. K., Melville, A., Rosenberg, D., Sears, A. P., Yan, F., Yoder, J. L., et al. (2017). Coherent coupled qubits for quantum annealing. *Physical Review Applied*, 8(1):014004.

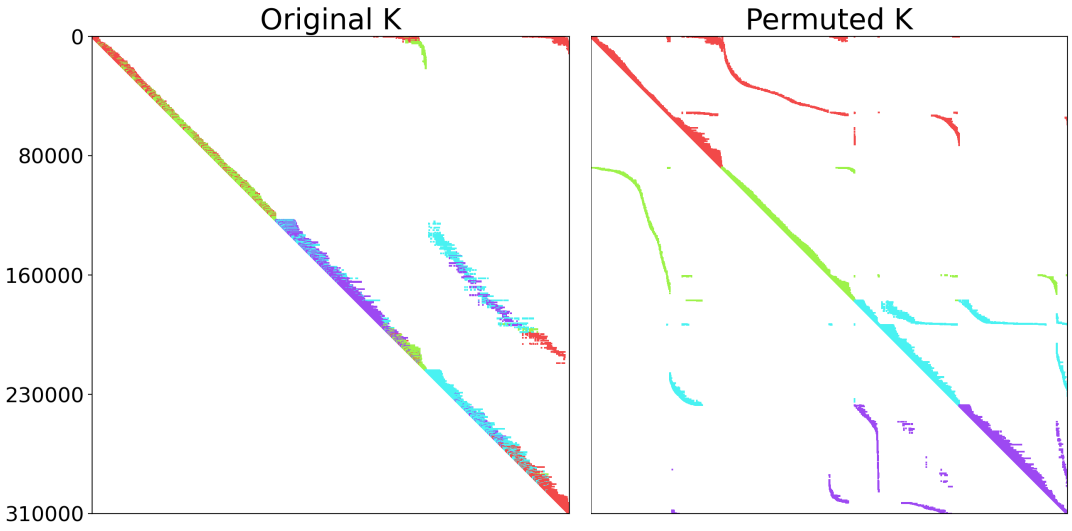
A APPENDIX

Figure 38: Demonstrator 01 K-matrix METIS medium coarsen size



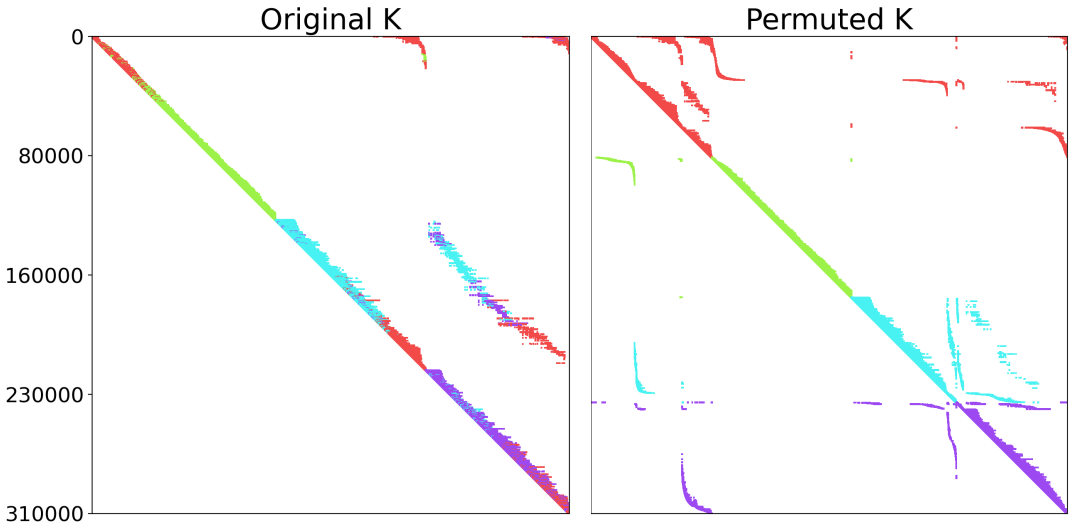
Source: Author (2026)

Figure 39: Demonstrator 01 K-matrix METIS small coarsen size



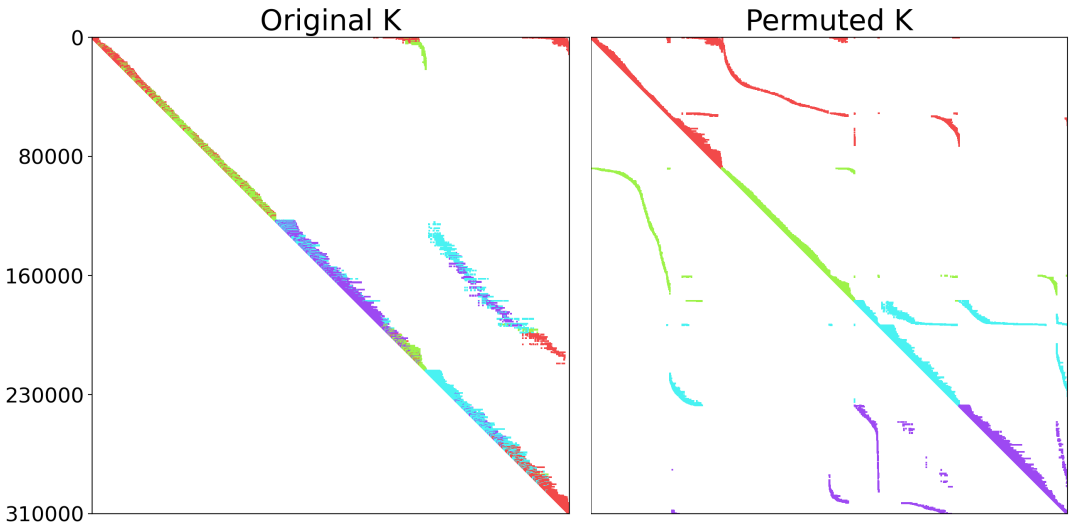
Source: Author (2026)

Figure 40: Demonstrator 02 K-matrix METIS medium coarsen size



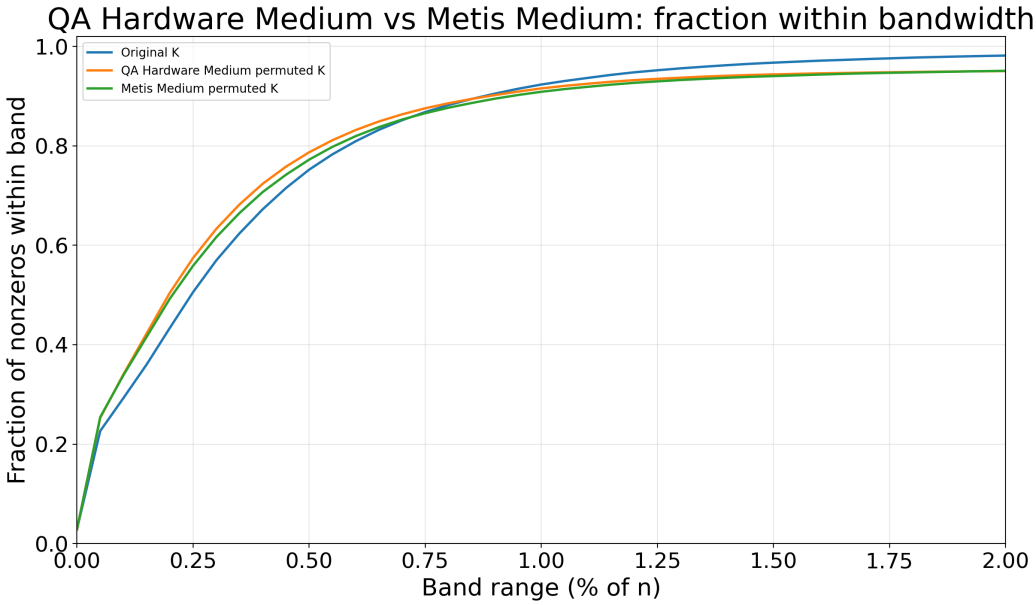
Source: Author (2026)

Figure 41: Demonstrator 02 K-matrix METIS small coarsen size



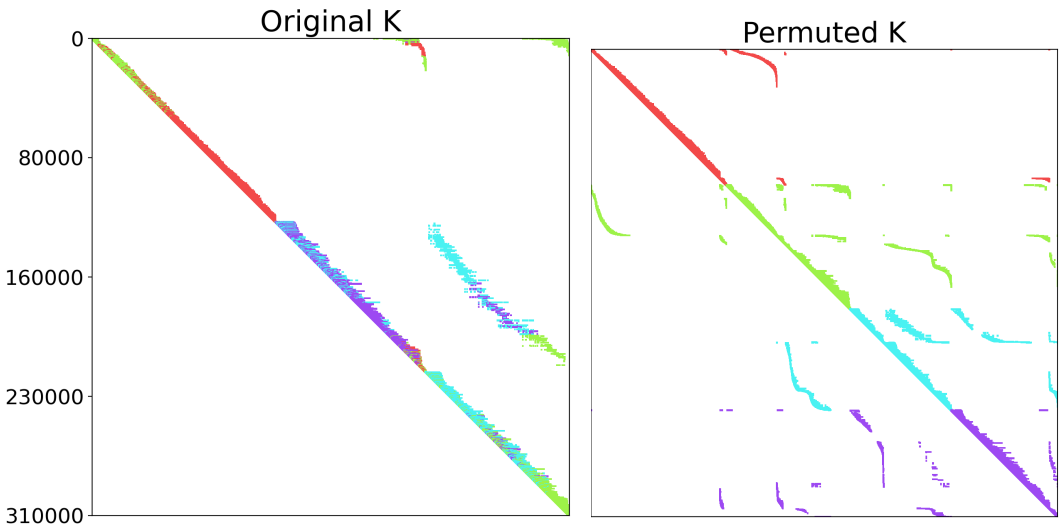
Source: Author (2026)

Figure 42: K-only fraction-within-band comparison for Demonstrator 01: QA hardware medium versus METIS medium.



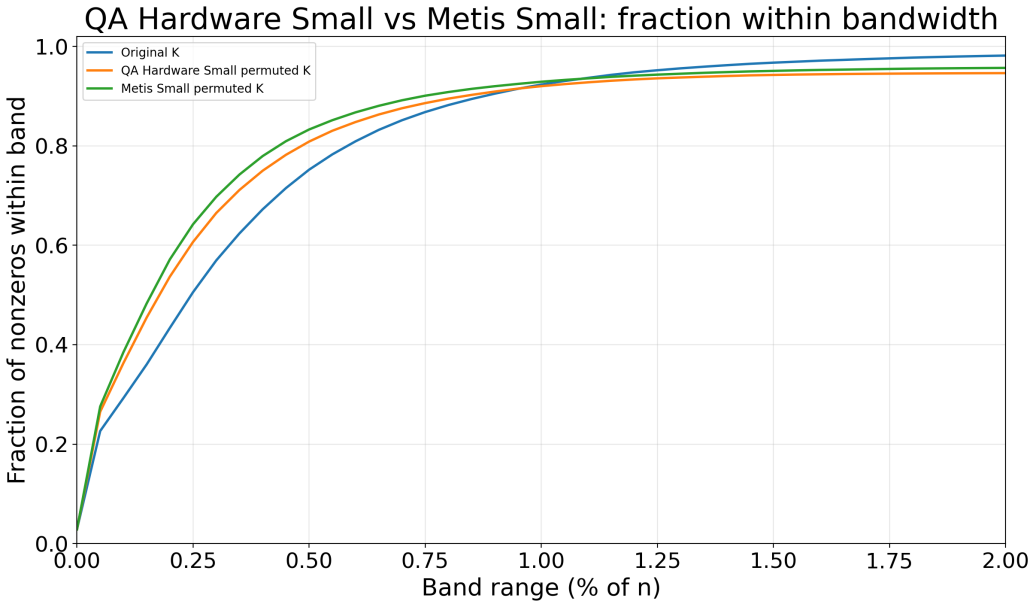
Source: Author (2026)

Figure 43: Demonstrator 01 K-matrix Quantum Annealing Hardware small coarsen size



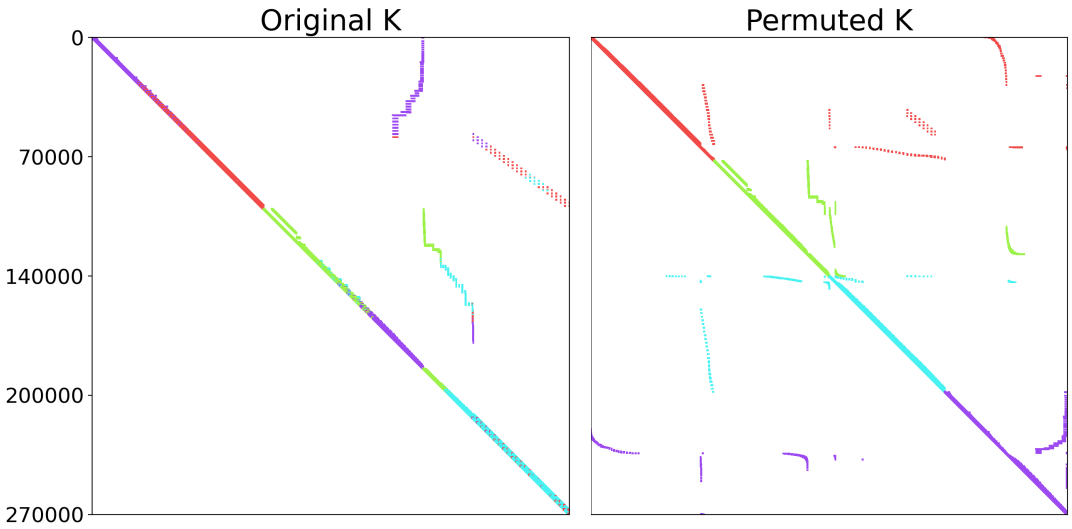
Source: Author (2026)

Figure 44: K-only fraction-within-band comparison for Demonstrator 01: QA hardware small versus METIS small.



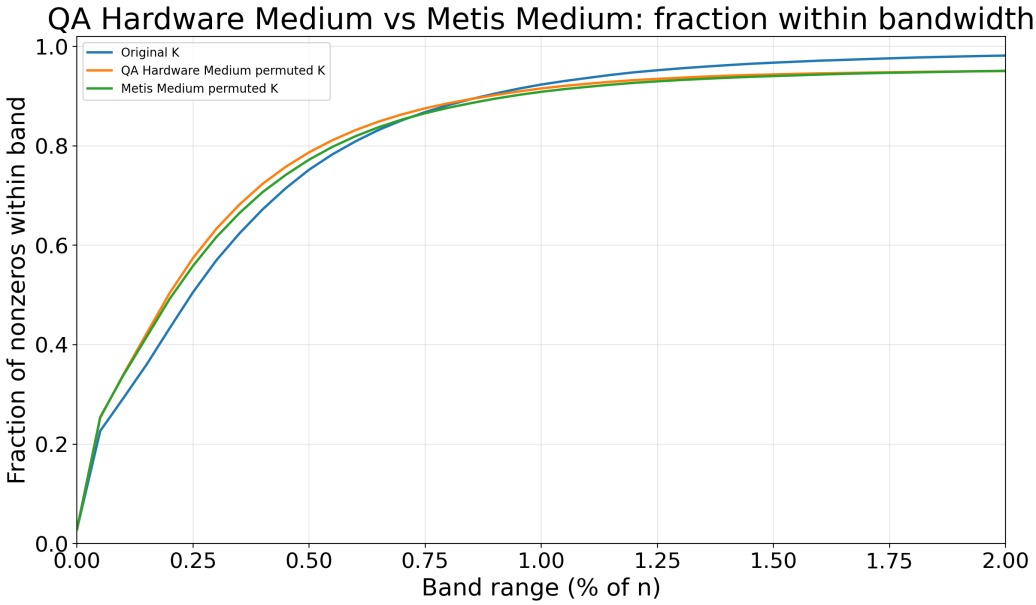
Source: Author (2026)

Figure 45: Demonstrator 02 K-matrix Quantum Annealing Hardware medium coarsen size



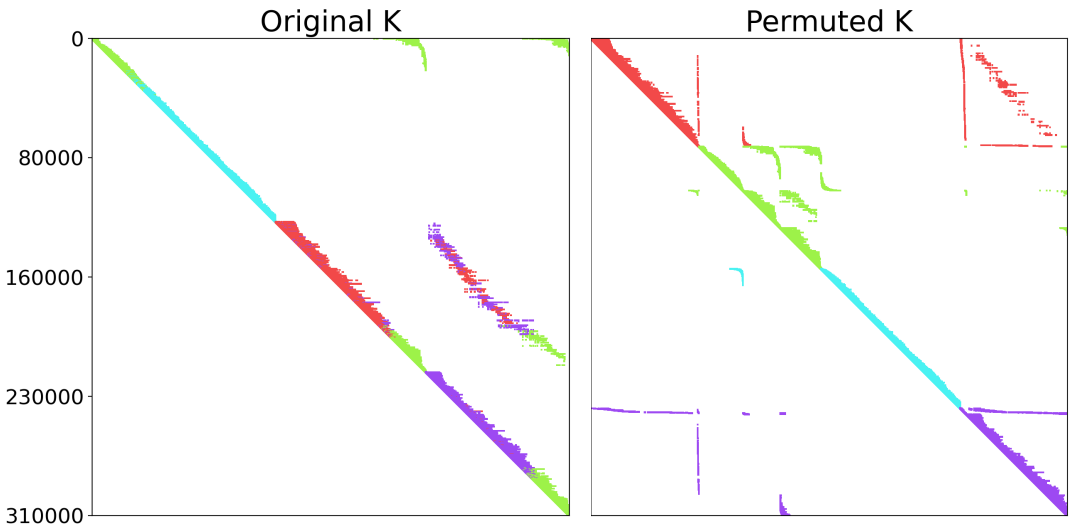
Source: Author (2026)

Figure 46: K-only fraction-within-band comparison for Demonstrator 01: QA hardware medium versus METIS medium.



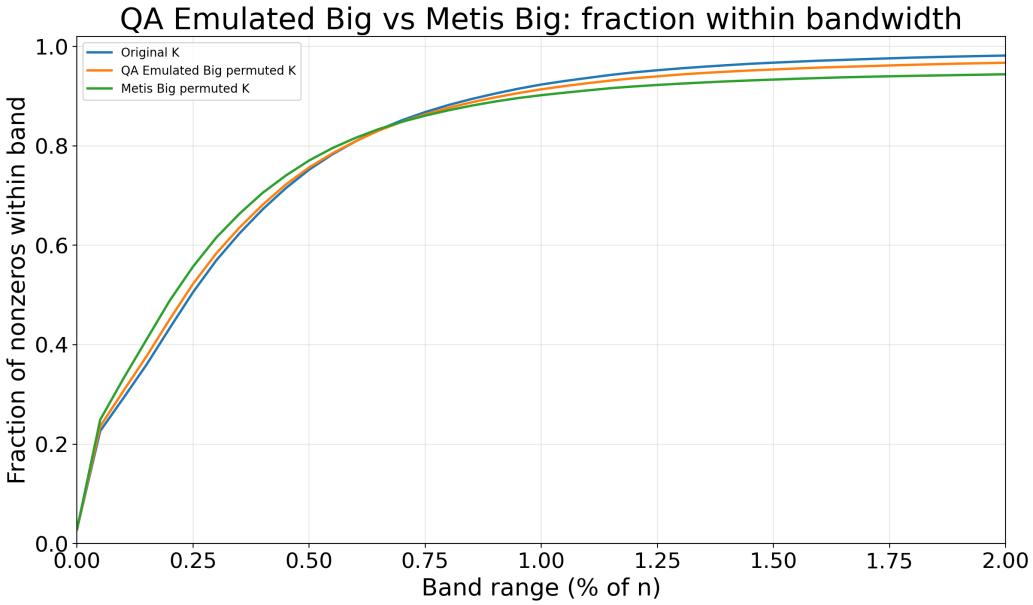
Source: Author (2026)

Figure 47: Demonstrator 01 K-matrix QA Emulated big coarsen size



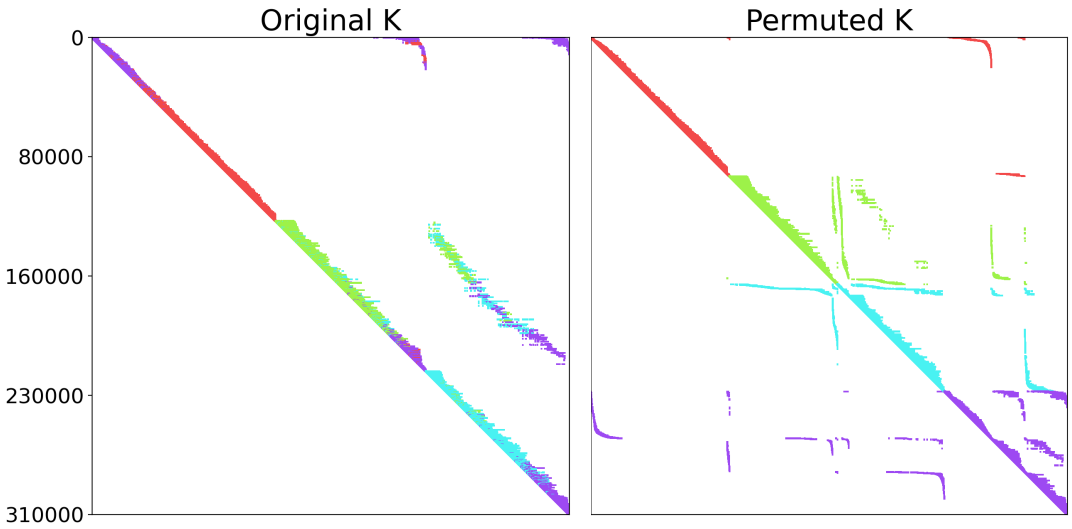
Source: Author (2026)

Figure 48: K-only fraction-within-band comparison for Demonstrator 01: QA Emulated Big versus METIS Big.



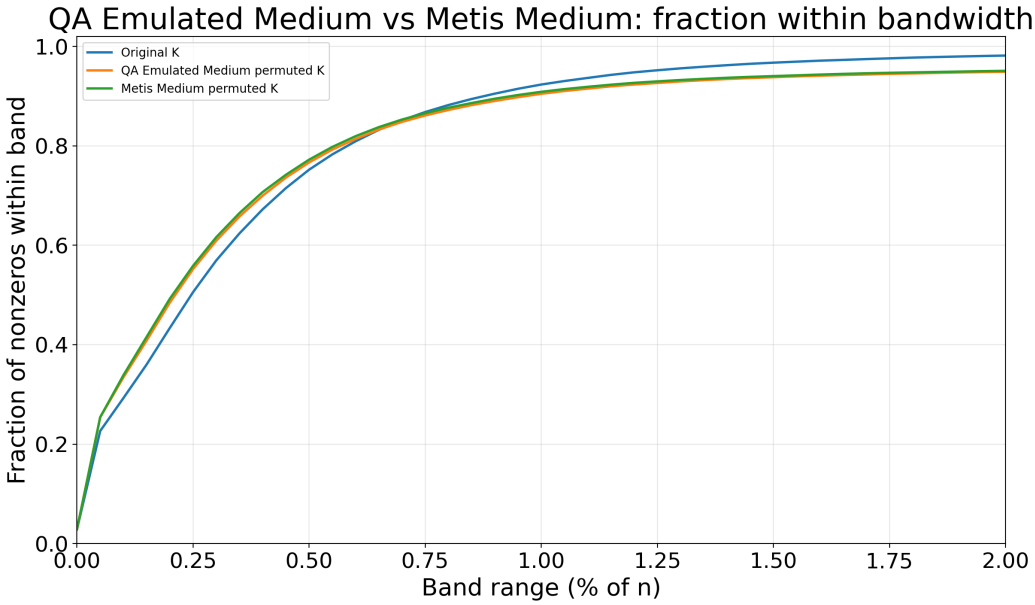
Source: Author (2026)

Figure 49: Demonstrator 01 K-matrix Quantum Annealing Emulated medium coarsen size



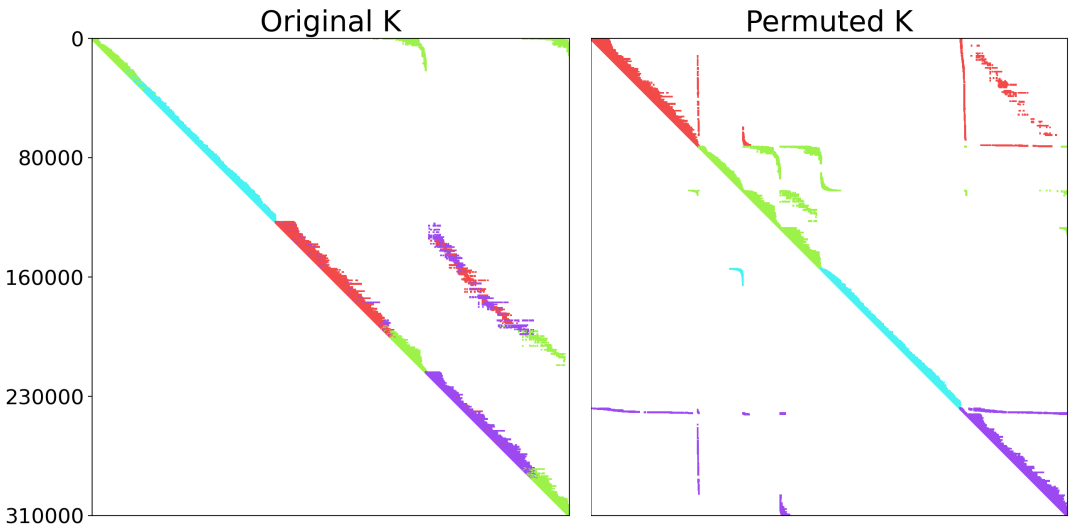
Source: Author (2026)

Figure 50: K-only fraction-within-band comparison for Demonstrator 01: QA hardware medium versus METIS medium.



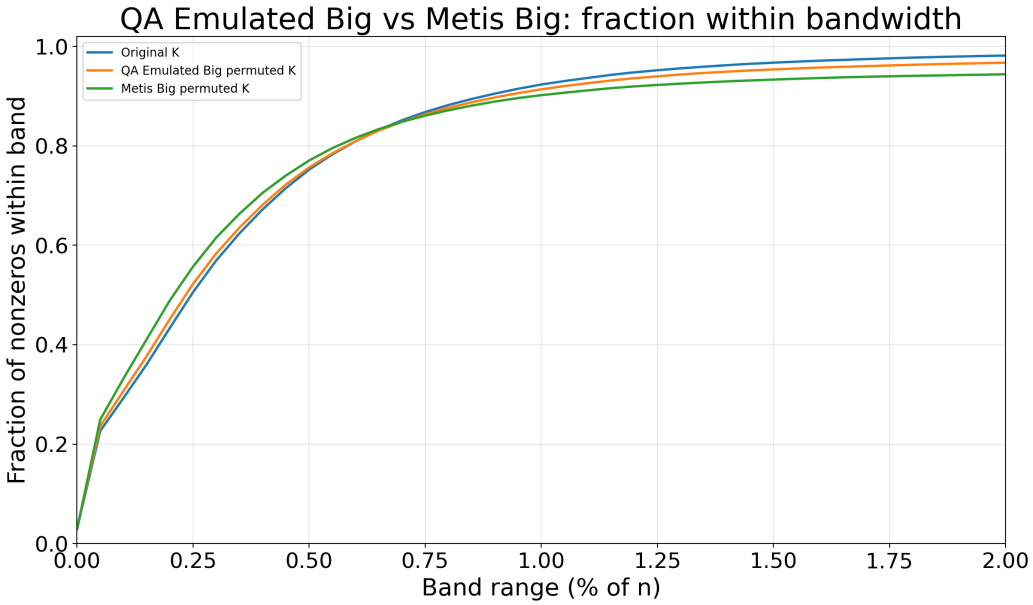
Source: Author (2026)

Figure 51: Demonstrator 02 K-matrix Quantum Annealing Emulated big coarsen size



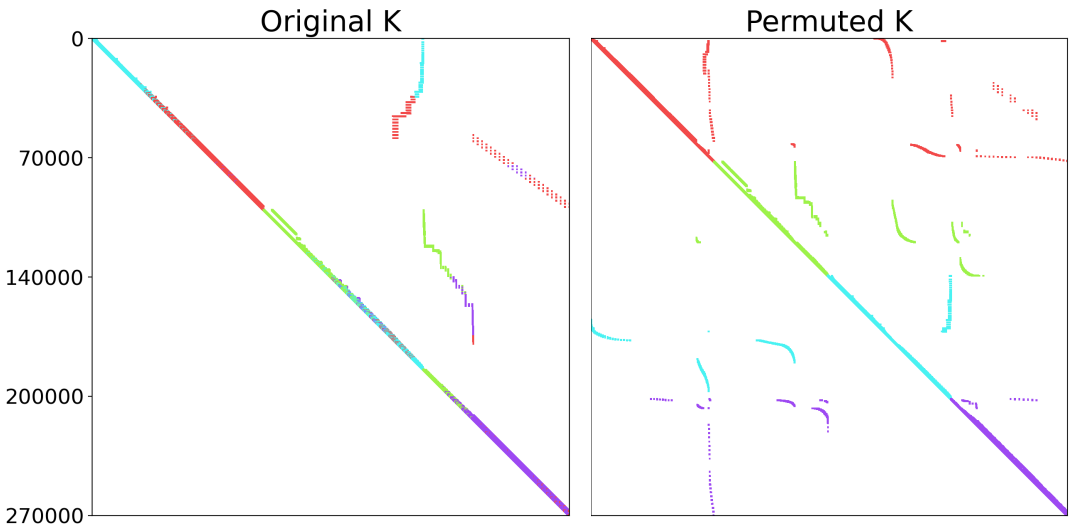
Source: Author (2026)

Figure 52: K-only fraction-within-band comparison for Demonstrator 02: QA emulated big versus METIS big.



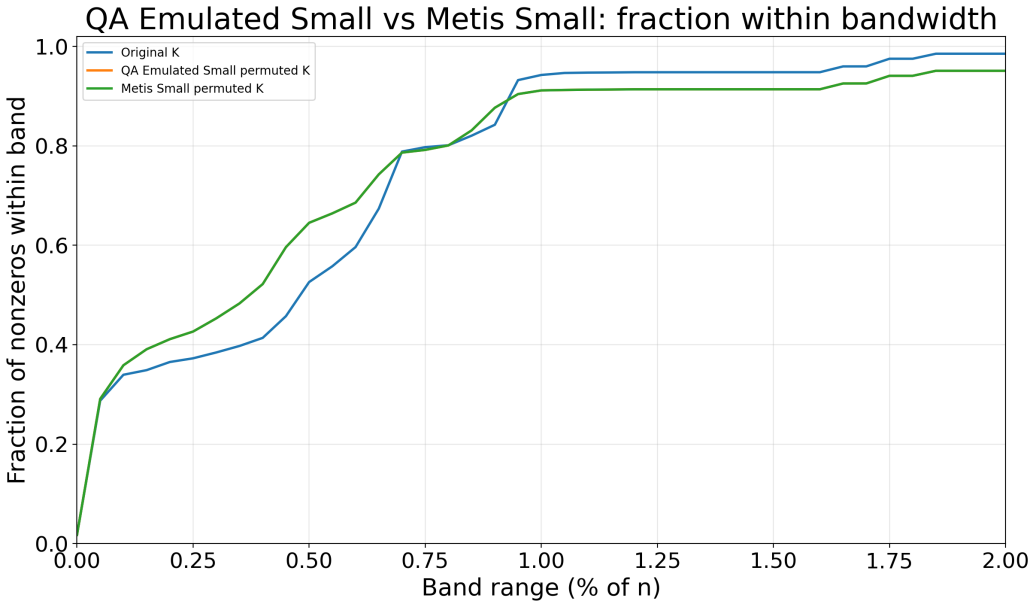
Source: Author (2026)

Figure 53: Demonstrator 02 K-matrix Quantum Annealing Emulated small coarsen size



Source: Author (2026)

Figure 54: K-only fraction-within-band comparison for Demonstrator 02: QA hardware small versus METIS small.



Source: Author (2026)