Parallel Finite Element Processing Using Gaussian Belief Propagation Inference on Probabilistic Graphical Models

Yousef Malek El-Kurdi

Doctor of Philosophy
Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

December 2014

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

To my father and my mother

To my wife and our children

ABSTRACT

The Finite Element Method (FEM) is one of the most popular numerical methods to obtain approximate solutions to Partial Differential Equations (PDEs). Due to its wide applicability, robustness and accuracy of its solution, the FEM takes a prominent role in the design and analysis of many engineering applications. However, the FEM is also considered a computationally intensive method when used for complex designs requiring accurate modeling. As a result, high fidelity FEM simulations create strong demand for scalable High Performance Computing (HPC) systems. The FEM's conventional approaches are based on global sparse matrix operations that severely limit the parallel scalability of costly HPC systems.

In this work we look into Belief Propagation (BP) inference algorithms to address the FEM's computational bottleneck. BP is a message passing algorithm on probabilistic Graphical Models (GMs) used to efficiently compute marginal distributions. BP algorithms can exploit the underlying problem's structure of variable interactions to expose more parallelism from its computations. A particular instance of BP algorithms is BP on Gaussian models with pairwise interaction (PW-GaBP). However, PW-GaBP did not provide the sought numerical efficiency for FEM problems in general due to its large number of iterations. Nonetheless, our analysis of using PW-GaBP to solve FEM problems reveals great insights on how to improve the numerical efficiency of BP style algorithms while maintaining its parallelism features.

Instead, we developed a novel probabilistic Gaussian GM for the FEM based on Factor Graphs (FEM-FG) by reformulating the FEM problem as a distributed variational inference problem. This development facilitates the use of computational inference algorithms such as BP to solve the FEM problem by decoupling it into local systems of low ranks. The resulting FEM Gaussian Belief Propagation (FGaBP) algorithm solves the FEM in parallel,

element-by-element, eliminating the need for large sparse matrix operations. The FGaBP algorithm demonstrates significant parallel scalability; nonetheless, its number if iterations scales linearly with the number of unknowns making it slow for larger FEM problems.

We remedy this by introducing a multigrid adaptation to FGaBP resulting in the fast FMGaBP algorithm. The FMGaBP algorithm processes the FEM in a fully distributed and parallel manner, with stencil-like element-by-element operations, while eliminating all sparse algebraic operations. To our knowledge, this is the first multigrid formulation for continuous domain Gaussian BP algorithms that is derived directly from a variational formulation of the FEM. In comparison with state-of-the-art parallel implementations of the Multigrid Preconditioned Conjugate Gradient (MG-PCG) solver, the FMGaBP algorithm demonstrates considerable speedups for both 2D and 3D FEM problems.

ABRÉGÉ

La méthode des éléments finis (MEF) est une des approches les plus populaires pour résoudre numériquement des équations aux dérivées partielles. La MEF s'applique à une grande variétés de situations et fournit des solutions robustes et précises, ce qui lui donne un rôle de premier plan dans la conception et l'analyse de plusieurs problème d'ingénérie. Par contre, la MEF requiert une forte puissance de calcul lorsqu'elle est utilisée pour résoudre des problèmes de conception complexes qui nécessitent une modélisation précise. Pour cette raison, les simulations par MEF à haute fidélité génèrent une demande importante pour des systèmes de calcul haute performance. Les approches conventionnelles utilisées pour la MEF sont basées sur des opérations globales effectuées sur des matrices creuses, ce qui limite sévèrement les possibilités de parallélisation.

La présente thèse s'intéresse aux algorithmes d'inférences par Belief Propagation (BP) dans le but de réduire la puissance de calcul nécessaire aux simulations par MEF. L'algorithme BP est basé sur des transport de messages à l'intérieur de réseaux statistiques servant à calculer efficacement des distributions marginales. Un algorithme BP est à même d'exploiter la structure des interactions entre variables pour exposer une plus grande part d'opérations parallèles. L'algorithme BP sur réseaux gaussiens avec interactions par paires, ou BP on Gaussian models with pairwise interaction (PW-GaBP), est un type d'algorithme BP bien connu. Malheureusement, nous avons constaté qu'en général, cet algorithme n'était pas à même de fournir une efficacité numérique suffisante pour se prêter aux simulations par MEF, à cause de son nombre d'itérations trop important. Malgré tout, notre analyse de l'algorithme PW-GaBP utilisé dans le cadre de la MEF fournit de bonnes pistes sur la façon d'améliorer l'efficacité numérique des algorithmes de style BP, tout en conservant ses propriétés parallèles.

Pour palier aux défauts de l'algorithme PW-GaBP, nous avons développé une nouvelle structure de réseau statistique gaussien pour la MEF à l'aide de Factor Graphs, en reformulant le problème de la MEF en tant que problème d'inférence variationnelle distribuée. Cette reformulation facilite l'utilisation d'algorithmes d'inférence tel que BP pour résoudre des problèmes de MEF en les séparant en plusieurs systèmes locaux à bas rang. Nous obtenons ainsi un algorithme appelé FEM Gaussian Belief Propagation (FGaBP) qui peut résoudre un système de MEF en parallèle, élément-par-élément, éliminant ainsi la nécessité d'effectuer des opérations complexes sur des matrices creuses. L'algorithme FGaBP peut être parallélisé de façon importante, mais il demande néanmoins un nombre d'itérations qui croît linéairement en nombre d'inconnus, ce qui le rend assez lent sur des problèmes MEF de grande taille.

Nous améliorons ce dernier point en proposant une adaptation multi-grille de l'algorithme FGaBP pour obtenir un algorithme plus rapide baptisé FMGaBP. L'algorithme FMGaBP traite le système MEF de façon parfaitement distribuée et parallèle. Les opérations se font élément-par-élément, tout en éliminant toutes les opérations sur matrices creuses. Au meilleur de notre connaissance, il s'agit de la première formulation multi-grille pour des algorithmes BP gaussiens dérivée directement d'une formulation variationnelle du système MEF. Par rapport aux implémentations actuelles de solveurs à gradient conjugué multi-grilles, l'algorithme FMGaBP offre des gains substantiels de vitesse pour les calculs de MEF, aussi bien dans le cas des problèmes 2D que pour ceux en 3D.

ACKNOWLEDGMENTS

I would like to extend my thanks to my advisors, Professor Dennis Giannacopoulos and Professor Warren Gross, for their guidance throughout my research life at McGill University. The long discussions we had were all insightful, inspiring, and joyful. I am also thankful for the financial support they provided. My special thanks to my PhD committee supervisors: Professor David Lowther, Professor Hannah Michalska, and Professor Louis Collins. Their insightful comments on my work are greatly appreciated. Many thanks to the McGill Faculty of Engineering for offering the MEDA scholarship which provided the needed financial support to start this PhD research.

And last but not least, my deepest appreciation goes to my wife Alaa, for her endless love and patience, to my children Malek and Maryam for being their adorable selves, and most of all, to my parents whose unconditional love was a great source of inspiration throughout challenging times.

Contents

List of Figures

List of Tables

List of Acronyms

AMD	Approximate Minimum Degree.
AMG	Algebraic Multigrid.
API	Application Programming Interface.
AU-FGaBP	Approximate Update Finite Element Gaussian Belief Propagation.
$b(\cdot)$	Belief distribution of either a variable node or a factor node.
BiCG	Biconjugate Gradient.
BLAS	Basic Linear Algebra.
BP	Belief Propagation.
BVP	Boundary Value Problem.
CA	Communication Avoiding.
CC	Compressed Coordinates.
CG	Conjugate Gradient.
CPS	Color-Parallel Message Schedule.
CSC	Compressed Sparse Column.
CSR	Compressed Sparse Row.
CUDA	Compute Unified Device Architecture.

DP-FP	Double-Precision Floating-Point.
D-PCG	Diagonally-Preconditioned Conjugate Gradient.
DRAM	Dynamic Random-Access Memory.
DR-GaBP	Dynamically Relaxed Gaussian Belief Propagation.
EM	Element Merging.
EMR	Edge Message Relaxation.
FN	Factor Node.
FDM	Finite Difference Method.
FEM	Finite Element Method.
FEM-FG	Finite Element Based Factor Graph.
FG	Factor Graph.
FGaBP	Finite Element Gaussian Belief Propagation.
FMGaBP	Finite Element Multigrid Gaussian Belief Propagation.
FPGA	Field Programmable Gate Arrays.
GB	Gigabyte = 1024^3 bytes of memory.
GB/s	Gigabyte per second of memory bandwidth.
GM	Graphical Model.
GMG	Geometric Multigrid.
GMRES	Generalized Minimal Residual.
GPU	Graphics Processing Unit.
GS	Gauss-Seidel.
HPC	High Performance Computing.

HUS	Hybrid Update schedule.
IC-PGC	Incomplete Cholesky-Preconditioned Conjugate Gradient.
JD	Jagged Diagonal.
KB	Kilobyte = 1024 bytes of memory.
KSM	Krylov-Subspace Method.
LBP	Loopy Belief Propagation.
LRM	Link Ratio Metric.
LSC	L-shaped corner of the Square Coaxial Conductor.
MB	Megabye = 1024^2 bytes of memory.
MD	Minimum Degree.
MG-PCG	Multigrid-Preconditioned Conjugate Gradient.
MPI	Message Passing Interface.
MRF	Markov Random Field.
MRR	Merge Reduction Ratio.
ND	Nested Dissection.
N_e	Number of edges in a graphical model.
$\mathcal{N}(\cdot)$	Neighborhood set of a particular node, which is the set of all adjacent nodes.
$\mathcal{N}(a) \setminus i$	Node a neighborhood set minus node i.
N_f	Number of factor nodes in a factor graph model.
NMR	Nodal Message Relaxation.
nnz	Number of non-zeros in a sparse matrix.
N_v	Number of variable nodes or vertices in a graphical model.

OOP	Object Oriented Programming.
OpenCL	Open Computing Language.
PCG	Preconditioned Conjugate Gradient.
PDE	Partial Differential Equation.
PPS	Partition-Parallel Message Schedule.
PUS	Parallel Update Schedule.
PW-GaBP	Gaussian Belief Propagation on Pairwise Graphical Models.
PWGM	Pairwise Graphical Model.
RCM	Reverse Cuthill-McKee.
R-GaBP	Relaxed Gaussian Belief Propagation.
SDD	Strictly Diagonally Dominant.
SM	Streaming Multiprocessor.
SMVM	Sparse Matrix Vector Multiplication.
SOR	Successive Over-Relaxation.
SP	Scalar Processor Core.
SPD	Symmetric Positive Definite.
SU	Speedup in the execution time.
SRM	Storage Ratio Metric.
SSC	Shielded Strip Conductor.
SUS	Sequential Update Schedule.
VN	Variable Node.
WDD	Weakly Diagonally Dominant.

CHAPTER 1

Introduction

1.1 Motivation

Mathematical models such as systems of Partial Differential Equations (PDEs) are used to describe the behavior of physical phenomena such as electromagnetism, fluid dynamics and material structural properties. A prominent example of PDEs is Maxwell's equations describing the propagation of Electromagnetic fields and waves in free space and material. The introduction of Maxwell's equations by James Clerk Maxwell in the second half of the nineteenth century [1] resulted in fundamental impacts on theoretical physics. Maxwell's equations are also the foundations behind many of the phenomenal advancements made by engineers in modern age information and communications technologies; such technologies are now pervasive in our day-to-day lives.

To unleash the full predictive power of Maxwell's equations in the analysis of electromagnetic fields, engineers must solve the set of Maxwell's PDEs. In many practical applications though, closed-form solutions of Maxwell's equations, as well as other PDEs in general, can not be obtained. This is due to the arbitrary geometries of the devices in question and the

material involved, which can also exhibit varying physical properties. As a result, approximate solutions to PDEs are sought in practice. One way of obtaining such solutions is by using numerical techniques such as the Finite Difference Method (FDM), the Finite Volume Method (FVM), and the Finite Element Method (FEM). The FEM is one of the most popular numerical techniques used by engineers today mainly due to its robust applicability and the accuracy of its solution.

The FEM is a rigorous numerical method that obtains an approximate solution to a system of PDEs by partitioning the continuous domain into discrete non-overlapping and well defined geometrical shapes. The discrete shapes are then used to interpolate the approximate solution on the full domain. The FEM became an indispensable part of the design and analysis processes of many engineering applications such as automotive, aerospace, biomedical, electromagnetic, material, power, and structural engineering. In addition, the FEM is a vital tool used by engineers in many R&D teams to accurately model complex technologies and simulate its designs, validate its system-wide specifications and optimize its performance. FEM modeling and simulation software helps eliminate many of the costly manufacturing procedures, that would have been otherwise needed for product prototyping and field testing, resulting in significantly shorter product design cycles and costs. As a result, FEM analysis are involved in many cutting-edge technologies that are ubiquitous in our day-to-day lives such as consumer electronics, mobile devices, circuits, wireless networks, optics, X-rays and medical imaging; as well as, major industrial technologies such as airplains, motors, turbines, actuators, transformers, sensors, coils and microwaves.

However, the FEM is also considered a computationally intensive method when used for complex designs requiring accurate modeling. FEM models can easily scale to solving for millions or billions of variable parameters. As a result, high fidelity FEM simulations create strong demand for scalable High Performance Computing (HPC) hardware. Traditionally,

CPU manufacturers such as Intel® and AMD®, used to scale the computational throughput of their CPU architectures by manufacturing CPUs that run at higher frequencies (frequency scaling). This allowed FEM users to accelerate their simulations by simply upgrading their workstations with faster CPUs. This also created a trend within the FEM software industry to rely more on inherently sequential solvers such as the Conjugate Gradient (CG) algorithm to better utilize the single-CPU workstations. While demands for HPC were met by high-cost CPU-clusters and supercomputers, the inherent inefficiencies in the FEM software due to sequential algorithms were not obvious.

A clear trend emerged in recent years showing that, CPU and HPC manufacturers such as Intel®, AMD®, IBM®, and HP® are scaling the computational throughput of their computing hardware by introducing more CPU cores in the same chip, referred to as multicore or manycore chips, which in effect, results in scaling the capacity of computation by means of parallelism as opposed to frequency scaling. This trend is enforced as a result of physical and power limitations at the micro-chip level that prevented further frequency scaling. A clear sign of this trend is the recent launch of the Xeon Phi™ [2] coprocessor by Intel® containing up to 61 cores in one silicon chip.

The HPC manycore parallel trend has introduced difficult challenges to both FEM software vendors and FEM software clients. FEM vendors, on one hand, need to parallelize their FEM software in order to better utilize manycore architectures; while clients on the other hand, demand high fidelity accurate FEM simulations for their increasingly complex designs. Such demands create strong constraints for the clients' workflow environment by efficiently utilizing their HPC platforms, reducing simulation time, increasing man-hour productivity, and reducing product design cycles.

Parallelizing FEM software is a very difficult task. The conventional FEM software relies on performing global and highly irregular (sparse) algebraic operations that severely limit its

parallel performance. For example, implementations of Krylov-Subspace Methods (KSMs), such as the CG solver [3], require global sparse operations which yield poor performance of about 10-20% of the peak CPU computational throughput [4]. This performance bottleneck is even more aggravated when high accuracy FEM analysis scales to large numbers of unknowns, in the order of millions or more, which prevents clients from productively utilizing their HPC platforms.

Most of the existing FEM software libraries attempt to improve performance by means of introducing sophisticated programming techniques mainly aimed at accelerating the underlying sparse algebraic operations. In contrast, our research aims to attack the root-cause of the problem by reformulating the conventional FEM in order to completely eliminate the global sparse operations, and perform instead localized dense computations in an embarrassingly parallel manner.

To address this challenge, we have investigated Belief Propagation (BP) inference algorithms. BP is a message passing algorithm based on probabilistic Graphical Models (GMs) to efficiently compute marginal distributions. We first recast the deterministic FEM problem as a variational inference problem on a probabilistic graphical model. This reformulation inherently eliminates the sparsity bottleneck of the FEM computation by decoupling the FEM into isolated local systems of very low ranks referred to as factor node and variable node belief distributions. These local belief systems update their states by communicating informative messages in a concurrent manner, which iteratively achieves a global solution as a stationary point of the local systems' consensus on the information flow. The concerted play of message communication between localized systems along the edges of the graphical model gives rise to the generalized Finite Element Gaussian Belief Propagation (FGaBP) algorithm. The FGaBP is characterized by a high parallel efficiency due to its distributed and localized computation in addition to its flexible memory communication.

While the first version of the FGaBP algorithm demonstrates convergence behavior that is proportional to the number of unknowns; this poses an issue for large scale FEM problems with millions of unknowns. We remedy this issue by using a multigrid adaptation of our algorithm referred to as the Finite Element Multigrid Gaussian Belief Propagation (FMGaBP) algorithm, which speeds up the propagation of information on the graphical model by bridging communications between far away nodes. The FMGaBP algorithm results in optimal convergence rates that yields a constant number of operations per unknown independent of the scale of the FEM problem. The numerical analysis of the developed algorithms, shown in chapters 5 and 6, demonstrates near-linear parallel speedup scalability on multicore CPUs. The linear scalability is achieved mainly due to the decoupled localized computation and the reduced iteration count. In addition, the FMGaBP algorithm achieves high parallel efficiency, time speedups, and iteration reductions when compared to state of the art conventional sparse solvers such as the Multigrid-Preconditioned Conjugate Gradient (MG-PCG).

1.2 Literature Review

Many attempts, as in [4–13], were made to improve the performance of the FEM's sparse computations at the expense of sophisticated programming techniques that are tailored to specific CPU architectures, such as cache access optimizations, data-structures and code transformations. Specialized hardware such as Graphics Processing Units (GPUs) and Field Programmable Gate Arrayss (FPGAs) are also used to offload the sparse kernels from the CPU in order to extract more parallelism and benefit from increased memory bandwidth [14–18]. The work in [19–21] presents techniques to accelerate sparse kernels on GPUs. An FPGA chip is used in [22, 23] to accelerate the Sparse Matrix Vector Multiplication

(SMVM) kernel using a custom-hardware made of a systolic pipeline of processing elements operating on specialized striping based data-structures for the sparse matrix. However, such optimizations are known to limit code portability and reusability. In addition, all such optimizations show that sustaining performance for different FEM applications with varying sparsity patterns is impossible. The work in [24] focuses on improving the performance of the sparse kernels used by solvers, such as the SMVM, by employing low-level hardware optimizations and automatic tuning. The use of tuning here is necessary due to the varying sparsity structure of different FEM applications; however, the tuning process can be very costly and can only be performed during run-time when the sparse matrix is provided by the user.

Accelerating CG solvers on parallel architectures is communication-bound; recent attempts to improve the communication overhead of such solvers through reformulation, namely communication avoiding schemes, suffer from numerical instability and limited support for preconditioners [25]. Another reformulation approach is presented in [26] which iteratively solves the FEM in-place element-by-element without assembling any sparse data-structures. While this approach resulted in high parallel scalability, it requires a considerably large number of iterations to converge, which reduces the advantages of parallelism.

While existing generic and optimized libraries such as deal.II [27], GetFEM++ [28], and Trilinos [29] can be used for sparse FEM computations, obtaining a sustained performance can be difficult due to the varying sparsity structure for different application areas. In addition, such libraries can not avoid the costly stage of assembling the large sparse matrix. However, the recent work in [30] uses a matrix free approach to accelerate the SMVM kernel in the FEM solver. While their approach shows promising speedups, it does not depart from the sequential global algebraic setup of the CG solver and limits preconditioning opportunities. Our work, in contrast to all of the above, does not mainly target implementation

algorithms. It instead is based on a higher-level distributed reformulation of the FEM using Belief Propagation (BP) that eliminates the need for any sparse data-structures; hence, circumventing the problem's main bottleneck.

The BP algorithm, as proposed by Pearl in [31], is a message passing algorithm on graphical models to compute marginal distributions, which was initially developed for tree-based graphical models. In practice however, problems generate graphical models containing many loops. In loopy models, the BP algorithm can be used iteratively to obtain an approximation for the marginals [31–34]. BP recently showed excellent empirical results in certain applications such as machine learning, channel decoding, computer vision, signal processing, and bioinformatics [35–47].

The work in [35] shows that BP can be viewed as an instance of the generic max-product algorithm operating on factor graph models. The work also illustrates that various algorithms developed in artificial intelligence, signal processing and digital communications can be viewed as specific instances of the max-product algorithm. Formal representation of graphical models, exponential distributions, and inference algorithms are illustrated in [35, 36, 48]. The relationship of BP on graphical models with free energy approximations are established in [32, 49–51]. Such work provides great insight for our later development of BP based algorithms for the FEM graphical model.

BP on models with Gaussian distributions is referred to as Gaussian BP. The exactness of the marginal means computed by Gaussian BP at convergence was established in [33]. In addition, the work provides analytical proofs on the convergence of Gaussian BP restricted to diagonal dominance guarantees of the model. The convergence condition, however, is expanded later by [52, 53] to include walk-summable models. Such models are characterized by the algebraic property $\rho\left(\left|I - D^{-1/2}AD^{-1/2}\right|\right) < 1$, where A is the sparse matrix representing the model, D is the diagonal entries of A, I is the identity matrix, and $\rho(\cdot)$ is the spectral

radius. The work in [54] introduced the Gaussian BP algorithm as a parallel solver for linear systems of equations by modeling these systems as pairwise graphical models. While the solver showed great promise for highly parallel computations on diagonally dominant matrices [55], it does not scale for large FEM matrices. It also fails to converge for high order FEM problems [56, 57]. The work in [58] introduced a method to force convergence of the Gaussian BP solver for none walk-summable matrices by perturbing the matrix to force diagonal dominance properties, which is then corrected in a nested Newton iteration. Such an approach however, results in a considerably large count of iterations to be useful for large sparse matrices. In addition, as a solver, it would still require assembling a large sparse data-structure. Our developed BP based algorithms for the FEM eliminate such problems. Also, the multigrid adapted BP algorithm referred to as the FMGaBP is, to our knowledge, the first to use multigrid schemes to accelerate the information propagation on the underlying Gaussian models, hence accelerating the convergence of the BP algorithm.

Implementation related work aimed at improving the BP execution performance by exploiting data-parallelism is presented in [59–62]. BP is one of the algorithms used in the library GraphLab [63] which implements higher-level parallelism via the MapReduce scheme [64] and Message Passing Interface (MPI) [65]; nevertheless, it is mainly aimed at machine learning applications. The work in [66, 67] improves the convergence performance of BP by using an informative based scheduling scheme. The focus of all these works, however, is mainly based on discrete state variables where the BP complexity increases with the number of states and the degree of the nodes, which is the number of incident edges on a particular node. Our work, on the other hand, is based on Gaussian variables which are in the real domain with fixed node degrees that are independent of the total number of nodes in the graph. The FEM problem scales differently which is by dramatically increasing the number of elements, or consequently the number of nodes, to orders of millions or more.

While the node degree may have a considerable influence for certain FEM problems, it is rather fixed in comparison with the total number of nodes. In addition, the convergence performance for FEM problems can more efficiently be treated using a specialized multigrid scheme for BP as we show later in Chapter 6. The FGaBP and the FMGaBP algorithms introduced in Chapters 5 and 6 address all of these scalability concerns for the FEM problem.

1.3 Contributions

The following are the highlights of the contributions in the thesis:

1. Implementation oriented scheduling schemes for the Gaussian Belief Propagation on Pairwise Graphical Models (PW-GaBP) solver aimed for FEM matrices. We introduce the Hybrid Update schedule (HUS) for the PW-GaBP solver which better exploits the sparsity structure of the FEM matrices to improve the parallel execution efficiency while at the same time minimizing the increase in iterations due to the parallel message updates.

2. The Relaxed Gaussian Belief Propagation (R-GaBP) algorithm which significantly reduces the number of iterations to convergence.

3. The Dynamically Relaxed Gaussian Belief Propagation (DR-GaBP) algorithm which dynamically adjusts the relaxation factor based on iterative error improvements. The DR-GaBP circumvents the complexity of determining an optimal relaxation factor a priori.

4. The reformulation of the variational FEM problem as a variational inference problem which facilitates the use of computational inference algorithms to solve the FEM problem.

5. The Finite Element Based Factor Graph (FEM-FG) model for the FEMs problem which facilitates the derivation of variants of the FGaBP algorithm.

6. The FGaBP algorithm which solves the FEM in parallel element-by-element using a matrix-free approach eliminating all global algebraic operations such as SMVMs.

7. The Approximate Update Finite Element Gaussian Belief Propagation (AU-FGaBP) algorithm variants which reduce the computational complexity of the FGaBP from $O(n^4)$ to $O(n^2)$ per factor node resulting in significant speedups for higher order and higher dimensionality FEM problems.

8. The element merge solution which improves the FGaBP parallel efficiency by trading-off CPU flops for memory bandwidth improving the CPU's overall computational throughput.

9. The FMGaBP algorithm which achieves optimal convergence rate of $O(N)$ operations for N unknowns while maintaining all the parallel features of the FGaBP algorithm of distributed stencil-like element-by-element computations.

10. A highly scalable multicore CPU implementation showing speedups against state-of-the-art FEM open-source software for both the FEM setup and solve stages.

11. A highly scalable manycore GPU implementation showing considerable speedups over multicore CPU. This implementation was accomplished in collaboration with Dr. Maryam Mehri Dehnavi.

1.4 List of Publications

Key results from this doctoral research are presented in the following publications:

1. Y. El-Kurdi, W. J. Gross, and D. Giannacopoulos, "Efficient Implementation of Gaussian Belief Propagation Solver for Large Sparse Diagonally Dominant Linear Systems," IEEE Transactions on Magnetics, vol. 48, no. 2, February 2012, pp. 471-474.

2. Y. El-Kurdi, W. J. Gross, and D. Giannacopoulos, "Relaxed Gaussian Belief Propagation," IEEE International Symposium on Information Theory Proceedings (ISIT), July 2012, pp. 2002-2006.

3. Y. El-Kurdi, W. J. Gross, and D. Giannacopoulos, "Parallel Multigrid Acceleration for the Finite-Element Gaussian Belief Propagation Algorithm," IEEE Transactions on Magnetics, vol. 50, no. 2, February 2014, pp. 581-584.

4. D. Giannacopoulos, Y. El-Kurdi, W. J. Gross, "Finite Element Methods and Systems," U.S. Patent Application US14/526,651, filed October 28, 2014. Patent Pending.

1.5 Computing Resources

The author would like to acknowledge the following computing resources which were instrumental in obtaining the results necessary to illustrate the performance of the developed algorithms:

- The supercomputer Colosse, managed by Calcul Québec and Compute Canada. The operation of this supercomputer is funded by: the Canada Foundation for Innovation (CFI); NanoQuébec; RMGA; and the Fonds de recherche du Québec - Nature et technologies (FRQ-NT).

- The supercomputer Sandy at the SciNet HPC Consortium. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute Canada; the Gov-

ernment of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto.

1.6 Thesis Outline

The outline of the Thesis is as follows: Chapter 2 presents the background material such as sparse linear solvers, an overview of the FEM formulation, probabilistic graphical models, and BP inference algorithms. Chapter 3 introduces parallel schedule implementation algorithms. It discusses the different scheduling approaches and then presents the HUS algorithm which offers high parallel efficiency for BP execution. Chapter 4 introduces the R-GaBP and the DR-GaBP algorithms which considerably improve the convergence rate of general Gaussian BP algorithms. Chapter 5 details the development of the FGaBP algorithm. The chapter starts by presenting the FEM as a variational inference problem and introduces the FEM-FG model. In addition, Chapter 5 discusses variations of the FGaBP algorithm such as lower complexity approximate formulations as well as parallel scheduling schemes, element merging, and parallel implementation techniques. Chapter 6 details the FMGaBP algorithm which is a distributed multigrid adaptation of the FGaBP algorithm that significantly accelerates convergence. The chapter also presents analysis of the algorithm as well as implementation details for both CPU and GPU architectures. Chapter 7 discusses future directions for the current work such as integrating the developed algorithms into FEM adaptive refinement schemes; and using cluster HPCs platforms with MPI implementation and specialized partitioning algorithms.

CHAPTER 2

Background

2.1 Sparse Linear Systems

Using methods such as the FEM and the FDM to solve linear PDEs commonly results in a sparse linear system of equations taking the algebraic form:

$$Au = b \tag{2.1}$$

where A is a sparse square matrix of order N; u is a vector of unknowns; and b is the right-hand side (or source) vector. The term "sparse" refers to the fact that the matrix A is made up of mostly zero off-diagonal elements. The size of the discretized PDE problem is characterized by N, or by the number of non-zeros (nnz) which is also proportional to N such that $O\big(nnz\big) = O\big(lN\big)$, where l is a constant factor. An important feature of sparse problems is that the factor l is independent of N. In addition, l is also considerably smaller than N ($l \ll N$). For example, l can be found to be in the range between five to a few hundreds, in some rare cases, while N can scale indefinitely to millions or more depending on the desired discretization level of the PDE domain. This feature is mainly due to the

discretization process, or meshing, which results in fixed local connectivities for each node irrespective of the size of the mesh. This can clearly be illustrated by considering a regular rectangular grid. Any interior node will always be surrounded by four rectangles even if the domain size is enlarged or the mesh density is increased by increasing the number of elements. Since off-diagonal elements in the matrix A indirectly represent coupling between nodes, uncoupled elements in the mesh will result in uncoupled nodes and consequently zero off-diagonals in A. This concept also applies to irregular meshes of arbitrary element geometries, however the local connectivity would slightly vary from one node to another.

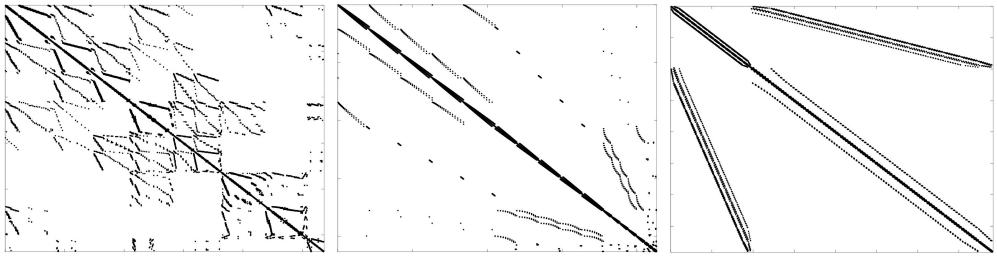
Sparse solvers can exploit the sparsity of the problem in order to gain considerable memory and computational advantages. For example, if we would solve the linear system (2.1) by directly taking the inverse of A using a dense solver, the solver would require $O\big(N^2\big)$ memory and $O\big(N^3\big)$ computational operations. In contrast, sparse solvers in general would require $O\big(N\big)$ memory and $O\big(N^d\big)$ computational operations where d typically is $d \in [1, 3]$. However, we will later see that certain iterative solvers such as multigrid schemes achieve optimal convergence scalability with $d = 1$ for elliptic PDEs.

Nonetheless, implementing sparse linear systems on HPC platforms is difficult. This is mainly due to the fact that sparse systems resulting from complex geometries can exhibit irregular sparsity patterns which can negatively impact their computational efficiency. To illustrate this issue, we present a brief overview of sparse data-structures and solvers.

2.1.1 Sparse Data-Structures

Discretization of simple domains using the FDM can mostly result in sparse systems that exhibit regular sparsity patterns. The sparse matrices for such systems typically have banded structures, where the non-zeros are extended along the diagonal of the matrix. Standard techniques to handle these systems are widely available and usually result in very efficient

HPC execution. However, the FDM method lacks the general applicability offered by the FEM method which enjoys a much wider use in the engineering and scientific community [1, p. 19]. The FEM method, on the other hand, exhibits a more degraded HPC performance if specialized techniques are not used. This degraded performance is mainly due to the irregular sparsity pattern of the matrix A. Fig. 2.1 shows sample sparse FEM matrices clearly illustrating its irregular sparsity patterns. Solutions for such systems will be the primary focus of our work.



(a) (b) (c)

Fig. 2.1 Samples of sparse matrices resulting from different FEM application areas. The matrices are obtained from the Matrix Market online repository [68] and plotted using Octave [69]. The application areas are: (a) Aircraft structure, matrix name: "can 1027" $N = 1027$ $nnz = 12444$. (b) Structural engineering, matrix name: "blackhole" $N = 2132$ $nnz = 14872$. (c) Waveguide, matrix name: "bfw782b" $N = 782$ $nnz = 5982$.

Sparse data-structures mainly store the non-zero elements of the matrix and their locations based on their row and column indices. However, the arrangement of the data-structure will need to support two key objectives. The first objective is to minimize the overall data-structure size by reducing the overhead data, such as pointers, indices, and zero paddings if needed. The second objective is to organize the data-structure in such a way to hide the memory access latencies and improve the memory bandwidth utilization for basic sparse operations such as the SMVM. Obtaining good performance for sparse solvers on HPC platforms usually requires balancing the trade-offs from these two objectives. At the present

time however, memory chips can be provided at commodity prices while memory bandwidth (connectivity with the CPU) remains limited; as a result, the second objective will usually have more impact on performance than the first one.

Examples of commonly used sparse data-structure formats are the Compressed Coordinates (CC), the Compressed Sparse Row (CSR), the Compressed Sparse Column (CSC), and the Jagged Diagonal (JD). In the CC format, each non-zero is stored along with its row and column indices; hence, the CC format is known to be the simplest and most flexible of all formats. The CC format however requires the largest overhead to store. The CSR format, on the other hand, requires less memory and performs relatively better for sparse operations such as the SMVM, hence it became one of the most popular formats. Fig. 2.2 illustrates a simple example of the CC and the CSR formats.

2.1.2 Sparse Linear Solvers

Sparse linear solvers mainly fall into two categories, sparse direct solvers and sparse iterative solvers. Sparse direct solvers perform some type of LU or Cholesky factorization while reducing the effect of fill-ins. Fill-in is a side-effect of direct sparse solvers that occurs from the factorization process by introducing non-zeros in locations that were zeros in the original sparse matrix. The effect of fill-in can negatively impact the CPU's performance by progressively increasing the memory and the computational requirements. Widely used open-source libraries for sparse direct solvers include UMFPACK [70, 71], SuperLU [72, 73], and MUMPS [74]. Specialized matrix reordering algorithms such as the Minimum Degree (MD) [75] reordering and the Nested Dissection (ND) [76] reordering can help in reducing the amount of fill-ins. Sparse direct solvers can be particularly efficient for problems requiring multiple solutions for different right hand side vectors (different b vectors in (2.1)), since only forward-backward substitution is needed to be applied on each right hand side after the first

$$A = \begin{bmatrix} 1. & 0. & 6. & 0. & 0. \\ 0. & 2. & 0. & 0. & 0. \\ 6. & 0. & 3. & 0. & 0. \\ 0. & 0. & 0. & 4. & 8. \\ 0. & 0. & 0. & 8. & 5. \end{bmatrix}$$

(a)

Ae	1.	2.	3.	4.	5.	6.	6.	8.	8.
Ri	1	2	3	4	5	1	3	4	5
Ci	1	2	3	4	5	3	1	5	4

(b)

Ae	1.	6.	2.	6.	3.	4.	8.	8.	5.
Ri	1	3	2	1	3	4	5	4	5
Rp	1	3	4	6	8	10			

(c)

Fig. 2.2 Examples of two sparse data-structure formats where Ae is the elements list of the matrix A, Ri is the element row index list, Ci is the element column index list, and Rp is the row start index list. (a) The sparse matrix. (b) The CC format. (c) The CSR format.

solve. Due to fill-in that substantially increases in 3D problems, the performance of sparse direct solvers is particularly worse than iterative solvers. A sparse Cholesky factorization would require $O\big(N^2\big)$ operations and $O\big(N^{4/3}\big)$ memory for discretized 3D Poisson's problems [77, p. 278]. Therefore, iterative solvers are more popular in HPC applications because they scale more efficiently for larger problems in terms of both memory requirement and execution time. In fact, certain iterative solvers can achieve the optimal computational time of $O\big(N\big)$ and the optimal memory space of $O\big(N\big)$ such as multigrid solvers or preconditioners. Henceforth, our focus in this work will be directed towards iterative solvers. In particular, we consider applications that result in linear systems with A being Symmetric Positive Definite (SPD). Such a property is commonly present in linear systems generated by a wide range of

FEM applications. Also, certain methods developed for SPD matrices can be generalize to non-symmetric positive definite matrices such as the Biconjugate Gradient (BiCG) method or, for more general definite systems, the Generalized Minimal Residual (GMRES) method. However, the two methods, the BiCG and the GMRES, are considered specializations of a wider class of solvers known as the Krylov-Subspace Methods (KSMs).

Two widely used iterative solvers are the CG method and the multigrid scheme. Extensive literature is devoted to these methods since their early conception. Recent references that introduce the theory, as well as various implementation approaches, include [3,8,77–81]. The CG method in particular, which was first introduced in 1950, only found wide use in the early 1970s when the efficiency of the method was vastly improved by the introduction of preconditioning [82]. In the following sections we provide a brief overview of these methods.

2.1.3 The Preconditioned Conjugate Gradient

The CG algorithm, applicable when A is SPD, is a special case of a more general class of algorithms referred to as KSMs. In such algorithms, the sequence of approximate solutions u_m are taken from an evolving subspace spanned by vectors resulting from repeatedly applying the matrix A on the initial residual $r_0 = b - Au_0$. Such a space is referred to as the Krylov-subspace and is denoted as $\mathcal{K}_m = \text{span}\{r_0, Ar_0, A^2r_0, \cdots, A^{m-1}r_0\}$. The CG algorithm can be viewed as a search direction algorithm that obtains A-orthogonal (conjugate) directions from the search space $\mathcal{K}_m$ in order to minimize the following quadratic:

$$\mathcal{Q}(u) = \frac{1}{2}u^T Au - b^T u + c \tag{2.2}$$

where c is a constant. It can be shown that the optimum point, $(u_\star)$, satisfies the gradient of $\mathcal{Q}$ such that $\nabla\mathcal{Q} \mid_{u=u_\star} = Au_\star - b = 0$. Using the residuals as the conjugating vectors

in the Gram-Schmidt orthogonalization process results in the elaborate CG algorithm. A key advantage of using conjugated directions is that in each iteration the error is not only being minimized in the current search space, but rather in all subspaces of the search space. As a result, the CG algorithm theoretically reaches the optimal solution in N iterations. In practice however, the CG algorithm converges in less than N iterations; for example, in Poisson's problems the convergence is upper-bounded by $O\big(N^{3/2}\big)$ for 2D and $O\big(N^{4/3}\big)$ for 3D [81]. Preconditioning considerably improves the convergence of CG by improving the characteristics of the matrix A at the expense of an additional computational cost per each iteration. In Preconditioned Conjugate Gradient (PCG), a SPD preconditioner matrix (M) is selected to approximate A such that the residuals can be made M^{-1}-orthogonal. Clearly for the preconditioning process to be efficient, M must be considerably easier to solve than A. The PCG algorithm pseudocode is listed in Algorithm 1. The key computational kernels in the PCG algorithm are the SMVM, shown in Line 7, and the preconditioner solve step, shown in Line 11.

2.1.4 The Multigrid Scheme

Multigrid accelerated solvers [79, 80, 83] are among the fastest known algorithms to obtain solutions to linear systems of equations resulting from the FEM. The performance of multigrid can, in practice, be shown to be independent of the size of the finest discretization of the domain [79, 80], attaining the optimal complexity of $O\big(N\big)$ for both computation and memory. In this section we present the general principles behind the multigrid scheme.

Multigrid schemes are based on creating different levels of approximations to a problem so as to obtain progressively lower complexities, or lower number of unknowns, on each sub-level. An approximate solution can be obtained very quickly on the lowest complexity level. The approximate solution is then transferred progressively to the problem's actual

Algorithm 1 The Preconditioned Conjugate Gradient (PCG) algorithm.

initialize:
  $r_0 = b - Au_0$
  solve: $Mz_0 = r_0$
  $d_0 = z_0$
repeat {Iterate $t = 1, 2, \cdots$}
  $\theta_t = z_t^T r_t$
  $v_t = Ad_t$
  $\rho_t = \frac{\theta}{d_t^T v_t}$
  $u_{t+1} = u_t + \rho_t d_t$
  $r_{t+1} = r_t - \rho_t v_t$
  solve: $z_{t+1} M = r_{t+1}$
  $\theta_{t+1} = z_{t+1}^T r_{t+1}$
  $\gamma_t = \frac{\theta_{t+1}}{\theta_t}$
  $d_{t+1} = z_{t+1} + \gamma_t d_t$
until convergence: r_{t+1} sufficiently small
Output: u_{t+1}

level of accuracy. This process is repeated iteratively in the hopes of speeding-up the overall computation time. The problem levels have to be related according to a criterion, otherwise the approximated solution can not be relayed across the adjacent levels. In the case of FEM applications, different problem levels can be created by performing nested mesh refinement, referred to as hierarchical mesh refinement. Such hierarchical meshing schemes are obtained by splitting each mesh element into smaller elements. Multigrid schemes using hierarchical meshes are referred to as the Geometric Multigrid (GMG). An example of an FEM library that uses GMG is the open-source library deal.II [27]. Alternatively, different levels of problem operators can be generated using algebraic operations on the matrix A, such multigrid schemes are referred to as the Algebraic Multigrid (AMG).

The key advantage of AMG schemes is that they are designed to be used as black-box solvers where all that is needed from the underlying PDE problem is the sparse linear system (2.1). An example of a solver library that uses AMG is the open-source library Boomer-

AMG [84]. AMG solvers are used in frameworks where the sparse matrix A is assembled once on the finest level. However, an AMG solver requires a number of global sparse algebraic operations as well as processing a number of sparse data-structures for each generated level operator. While GMG, on the other hand, provides the potential of implementing matrix free operations. In addition, GMG performs considerably better because it can better utilize the inherent problem structure in the hierarchical FEM mesh to lower the computational complexity. Therefore, it is advisable to use AMG only when it is impossible, or difficult, to setup GMG. Hence, the choice weather to use AMG or GMG is more application dependent than it is performance dependent. Our work, described in Chapter 6, utilizes GMG schemes to accelerate a parallel formulation of the FEM problem.

To illustrate the key formulations behind the multigrid solvers, we provide here a brief overview of the two-grid correction scheme. Given an initial discretization for a continuous bounded domain Ω, refereed to as a coarse mesh Ω^H, then the mesh is refined further by element splitting schemes into a finer mesh Ω^h. This refinement scheme is illustrated in Fig. 2.3. The multigrid correction scheme is based on the simple reformulation of the linear system $Au = b$ into the residual-correction scheme. Given an exact solution $u_\star$ and an initial approximation $\tilde{u}$, then the correction z can be stated as:

$$z = u_\star - \tilde{u}. \tag{2.3}$$

We start by finding the residual:

$$r = b - A\tilde{u}, \tag{2.4}$$

then the solution to the system $Au = b$ can alternatively be obtained by solving for the correction as:

$$Az = r, \tag{2.5}$$

with the initial guess $z = 0$. The obtained correction z is then added to the initial approximation $\tilde{u}$ to obtain the solution as:

$$u_\star = z + \tilde{u}. \tag{2.6}$$

The residual-correction reformulation has no computational advantage in itself; however, this reformulation is converted into an iterative process by multigrid. Using multigrid, the correction in (2.5) is rather computed on a coarse grid (smaller version of A) which is then used to compute an improved initial approximation for the fine grid. We expect an overall computational advantage, since working on the coarse grid is cheaper than working on the fine grid.

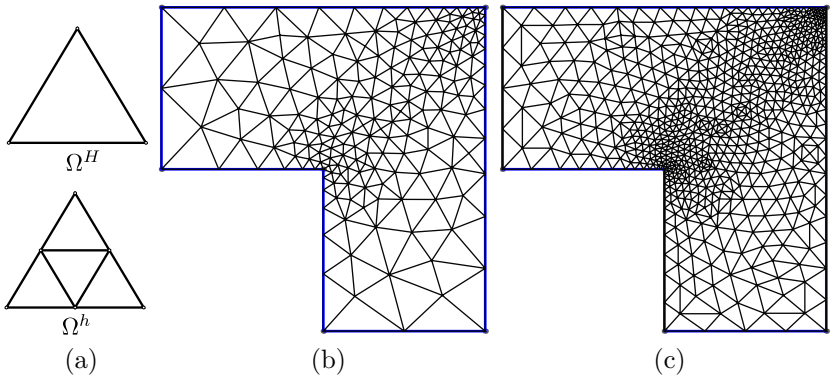


Fig. 2.3 (a) Mesh refinement by splitting each triangle in mesh Ω^H into four geometrically similar sub-triangles to produce a finer mesh Ω^h. (b) Sample coarse irregular mesh. (c) The refined mesh by splitting.

We redefine the linear system in (2.1) as $A^h u^h = b^h$; where the superscript h represents the linear system of the fine mesh Ω^h. If the superscript H is used, then the system represents the coarse mesh. The listing in Algorithm 2 shows the detailed operations of the two-grid V-cycle multigrid. A single iteration of the algorithm is refereed to as a V-cycle which reflects the behavior of the algorithm in transferring approximations from fine to coarse grids and then back from coarse to fine grids. Other key components of the algorithm are

Algorithm 2 The two-grid V-cycle multigrid process.

1: Set initial guess $z_0^h = 0$
2: **repeat** {V-cycle iterate}
3: Obtain the approximate solution $\tilde{u}_0^h$ from $A^h u^h = b^h$ on the fine grid Ω^h using v_1 iterates (pre-smoothing) of a relaxation algorithm with initial guess z_0^h.
4: Compute the residual $r^h = b^h - A^h \tilde{u}_0^h$.
5: Restrict the residual on the coarse grid Ω^H using $r^H = \mathcal{R}_h^H r^h$, where $\mathcal{R}_h^H$ is a fine-to-coarse restriction operator.
6: Obtain the coarse grid correction $\tilde{e}^H$ by solving $A^H e^H = r^H$ with initial guess $e^H = 0$.
7: Compute $z_1^h = \tilde{u}_0^h + \mathcal{I}_H^h \tilde{e}^H$, where $\mathcal{I}_H^h$ is a coarse-to-fine prolongation operator.
8: Obtain the approximate solution $\tilde{u}_1^h$ from $A^h u^h = b^h$ using v_2 iterates (post-smoothing) with initial guess z_1^h.
9: Set new initial guess $z_0^h = \tilde{u}_1^h$.
10: **until** Convergence on fine grid.

the restriction and prolongation operators. The restriction operator $\mathcal{R}_h^H$ acts on the residual from the fine grid by mapping it from the fine grid onto the coarse grid using restriction. On the other hand, the prolongation operator $\mathcal{I}_H^h$ acts on the correction from the coarse grid by mapping it onto the fine grid using interpolation. Since restriction and prolongation operators act on the global residual and correction vectors, they are in fact some form of global sparse operations. However, in certain applications, such as the FDM, these operators can be performed on a point-by-point basis. In linear transfer operations, the dimensions of $\mathcal{R}$ is given by $(N_H \times N_h)$ where N_H is the number of unknowns in the coarse grid, while N_h is the number of unknowns in the fine grid. On the other hand, the dimension of $\mathcal{I}$ is $(N_h \times N_H)$. This residual-correction scheme is also effective in progressively minimizing the error added due to both the restriction and the prolongation operations as the approximate solution on the fine grid approaches the exact solution. In addition, since the problem has a unique solution, it can be shown that the whole multigrid process can be viewed as a fixed-point process for the exact solution on the fine grid.

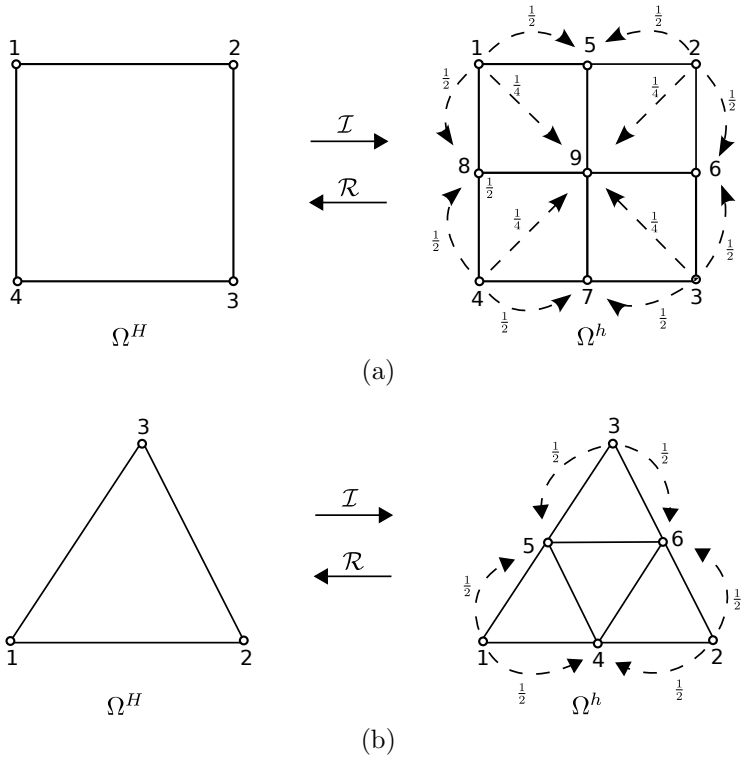


Fig. 2.4 Grid transfer operations between: (a) two 2D quadrilateral meshes and (b) 2D triangular meshes. The fractions indicated on the fine meshes represent nodal weightings for the case of linear interpolation.

To elaborate more on the behavior of the restriction and the prolongation operations, we consider a simple example using a 2D quadrilateral mesh shown in Fig. 2.4(a) The coarse mesh contains only one quadrilateral element, while the fine mesh contains four refined quadrilateral elements by splitting the coarse grid element evenly. The nodal numbers on the diagram reflect global numbering of the indices. Similarly as indicated in Fig. 2.4(b), a triangular mesh can be refined by splitting a triangular element into four geometrically similar child triangles. This splitting is performed by connecting nodes inserted at the mid point of each edge in the parent triangle.

The prolongation operator $\mathcal{I}_H^h$ can be defined using linear interpolation. In linear interpolation, the correction values of the fine grid nodes, which do not overlap the coarse grid

nodes, are obtained by averaging the surrounding nodes from the coarse grid. The overlapping nodes on the fine grid can retain the same correction values from the coarse grid. Hence the operator $\mathcal{I}_H^h$ matrix can be defined as:

$$\mathcal{I}_H^h = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}. \tag{2.7}$$

The restriction operator $\mathcal{R}_h^H$ is defined by linear injection, that is by assigning the residual values of the coarse grid nodes the same restriction values of the overlapping fine grid nodes. Other forms of averaging schemes can also be employed in certain cases. Using injection, the operator $\mathcal{R}_h^H$ matrix can be defined as:

$$\mathcal{R}_h^H = \begin{bmatrix} 1 & 0 & 0 & 0 & \\ 0 & 1 & 0 & 0 & \\ & & & & [0] \\ 0 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 1 & \end{bmatrix}. \tag{2.8}$$

However, in certain multigrid applications, $\mathcal{I}_H^h$ can simply be defined by enforcing the con-

dition:

$$\mathcal{R}_h^H = a\left(\mathcal{I}_H^h\right)^T \text{ and } a \in \mathbb{R}^+, \tag{2.9}$$

where a is a constant.

In practice, the two-grid V-cycle process is generalized to a recursive multigrid process by restricting the residual to lower levels of consecutive coarser grids. Once the coarsest grid is reached, the correction e^H is obtained using a direct solver. The correction is then prolongated progressively to higher levels of finer grids. On each level down the V-cycle, v_1 number of iterations are executed, while v_2 number of iterations are executed on each level up the V-cycle.

Multigrid techniques can vary widely by using different relaxation algorithms with different approximation properties, the approach of generating coarse grids operators, and the types of grid transfer operations that involve restriction and prolongation [79,80]. Hence, the performance of any multigrid scheme can strongly depend on these variations. In general, multigrid accelerated algorithms show much better scalability and parallel efficiency than their non-multigrid counterparts. This is particularly evident when using multigrid as a preconditioner for the CG method. In [80, p. 284], it was found that the complexity of solving Poisson's equation using Incomplete Cholesky-Preconditioned Conjugate Gradient (IC-PGC) was $O\left(N^{5/4}\right)$ for 2D and $O\left(N^{9/8}\right)$ for 3D; whereas, the MG-PCG can achieve approximately $O\left(N\right)$ complexity which is an optimal result [85]. Trottenberg et al. [80, p. 278] presents MG-PCG as a multigrid accelerated by a Krylov-subspace method by recombining successive approximations in a way to minimize the residual using a Gram-Schmidt orthonormalization process. In addition, the MG-PCG has been shown to exhibit a robust convergence on domains with geometric singularities. Due to the optimal performance expectation of MG-PCG, we will use it as a performance comparison to illustrate the advantages of the new

FMGaBP algorithm, presented later in Chapter 6, using test cases of Poisson's problems on domains with geometric singularities, or non-convex domains, such as the L-shaped domain shown in Fig. 2.3.

2.2 The Finite Element Method

In this section, a brief overview of the Finite Element Method (FEM) is presented. The FEM is a widely used numerical technique for obtaining approximate solutions to PDE problems. The FEM is also a computationally intensive method which can greatly benefit from efficient execution on parallel platforms. After introducing the FEM, we will illustrate its computational challenges and elaborate on the shortcomings of conventional schemes which attempt to accelerate the FEM computations.

The Helmholtz equation is a particular class of PDEs that addresses many of the electromagnetic formulations resulting from Maxwell's equations. The FEM is a key method that is commonly used to obtain numerical solutions to the Helmhotz equation. The FEM employs two types of formulations for approximating the PDE problem, the Ritz formulation and the Galerkin formulation. To obtain further details on these two methods including derivations for a wider set of electromagnetic applications, the reader can refer to Silvester et al. [86] or Jin [1]. Our overview here presents only the key formulations for obtaining approximate solutions to the Helmholtz equation using the FEM Ritz formulation. We chose the Ritz formulation since it is based on a variational formulation of the underlying Helmholtz PDE. More importantly, the FEM variational formulation provides critical insights that facilitates the derivation of our highly parallel FGaBP method, later addressed in Chapter 5. The

general scalar Helmholtz equation is stated as follows:

$$\nabla \cdot (p\nabla u) + k^2 qu = g \tag{2.10}$$

where u is the unknown field defined on the bounded domain Ω; p and q are known parameters associated with the material properties of the domain; k^2 is a constant quantity independent of position. Lapace, and Poisson PDE equations are special cases of (2.10). For example, the familiar Laplace's equation

$$\nabla^2 u = 0 \tag{2.11}$$

results by setting $p = 1$, $q = 0$ and $g = 0$. Whereas introducing material properties $\epsilon = p(x, y, z)$ and allowing sources $g = -\rho(x, y, z)$, we obtain the more general Poisson's equation

$$\nabla \cdot (\epsilon \nabla u) = -\rho. \tag{2.12}$$

The equations (2.10), (2.11) and (2.12) are also referred to as Boundary Value Problems (BVPs) because they can only be uniquely solved by defining conditions on the domain's boundary ($\partial\Omega$). These boundary conditions are generally stated as follows:

$$u = u_0 \text{ on } \partial D \tag{2.13}$$

$$\nabla u \cdot \hat{n} + au = v_0 \text{ on } \partial C \tag{2.14}$$

where ∂D and ∂C are boundary segments such that $\partial D \cup \partial C = \partial\Omega$; the parameters a, u_0, and v_0 are known parameters associated with the properties of the boundary; and the vector $\hat{n}$ is the outward normal unit vector of the boundary. The condition in (2.13) is commonly referred to as the Dirichlet boundary condition. When $a = 0$, (2.14) is referred to as the

Neumann boundary condition; however, with the added condition $v_0 = 0$ it is termed the homogeneous Neumann boundary condition.

Considering scalar potential problems with lossless media and general inhomogeneous boundary conditions as stated in (2.13) and (2.14), the solution to the scalar Helmholtz equation can be obtained by rendering stationary the following functional:

$$\mathcal{F}(U) = \frac{1}{2}\int_{\Omega}\left(p\nabla U \cdot \nabla U - k^2 q U^2 + 2gU\right)\,\mathrm{d}\Omega + \frac{1}{2}\int_{\partial C} apU^2 \mathrm{d}\Gamma - \int_{\partial C} pUv_0 \mathrm{d}\Gamma \qquad (2.15)$$

where U is an admissible approximating function space of the unknown field u and Γ is a boundary segment. While the functional $\mathcal{F}$ provides a means of finding the solution to the Helmholtz equation by avoiding the original BVP, it does not, however, provide an indication on how to chose the admissible functions U. This is where the FEM comes into play.

2.2.1 The FEM Solution

The FEM provides a rigorous method for obtaining an approximate solution to the BVP through two main steps. The first step is the discretization of the continuous domain Ω into finite subdivisions referred to as finite elements. The second step is choosing appropriate interpolation functions that setup the function space U while guaranteeing its continuity across the discretized elements within the domain Ω. Different approaches of the FEM can vary widely based on the details of how the above steps are carried out; however, we will highlight here the key concepts behind these steps.

The process of discretization, where the domain is subdivided into connected finite elements of a particular shape, is commonly referred to as meshing. The finite elements can be of any geometrical shape as long as they are connected through their vertices. Most commonly used geometrical shapes are triangles or quadrilaterals for 2D domains and tetra-

hedrons or hexahedrons for 3D domains. Figure 2.5 shows an example 2D domain discretized with four connected triangular elements.

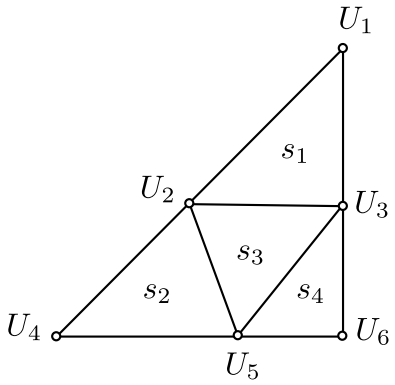


Fig. 2.5 A sample 2D domain discretized with four triangular elements of the first order.

Two types of meshes are commonly used which are referred to structured meshes and unstructured meshes. The vertices in a structured mesh lie on grid lines that pass uninterrupted throughout the domain. As a result, locality information, such as neighboring vertices or edges, can be directly computed from the nodes' indices. This offers a critical advantage in reducing the overall CPU's memory communication overhead. However, structured meshes do not adapt well to arbitrary geometries. Unstructured meshes, on the other hand, place vertices on arbitrary locations in the domain. As a result, the CPU will require more memory communication overhead in order to process an unstructured mesh. The key advantage of unstructured meshes though, is that they can better adapt to arbitrary geometries than structured meshes. In addition, unstructured meshes can allow for increased refinement on local patches situated in interesting domain locations while maintaining coarser patches in other areas. This results in an improved overall solution accuracy with a lower computational cost. Fig. 2.6 illustrates the two types of meshes. It is important to note here that, the FGaBP algorithm does not depend on the type of mesh used; as will later be shown, the FGaBP was implemented and verified to work with both structured and unstructured

meshes.

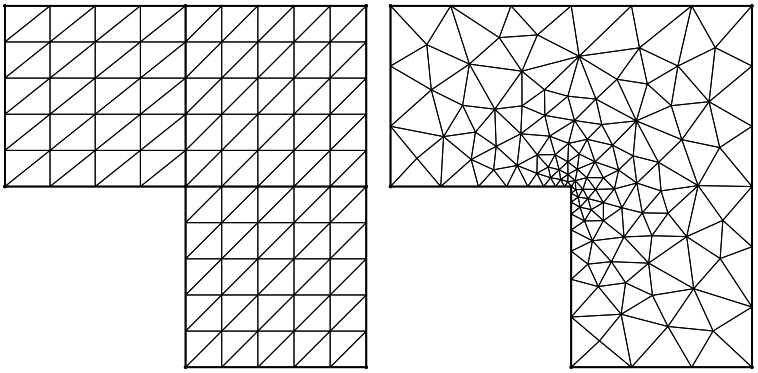


Fig. 2.6 Structured and unstructured meshes of the L-shaped domain. (a) The structured mesh. (b) The unstructured mesh. The unstructured mesh provides more refinement at areas of more interest in the domain such as the reentrant corner of the L-shaped region.

The second step in the FEM process involves choosing the interpolation polynomials, or as alternatively known the basis functions, in order to approximate the unknown field U within each of the finite elements. The interpolation polynomials approximate the unknown field using a discrete set of real-valued parameters spatially positioned along each of the finite elements' vertices or edges. The number of used parameters identify the polynomial interpolation order; for example, a linear triangular element has three parameters positioned along its vertices whereas a second order triangular element has six parameters positioned along its vertices and edges. For certain FEM applications, the quality of the approximation will improve with increasing interpolation order; however, the computational complexity of the FEM will substantially increase. An expression of this approximation can generally be stated as follows:

$$U^s(x,y,z) = \sum_{k=1}^{n} U_k^s P_k^s(x,y,z), \quad \forall s \in \mathcal{S} \tag{2.16}$$

where s is the finite element and $\mathcal{S}$ is the total elements' set spanning the domain Ω; n is the

number of local discrete values; and P is the interpolating function. Note that the subscript k in U_k^s refers to the local element node numbering and not the global node numbering; for example, the element s_3 in Fig. 2.5 has the globally numbered discrete parameters U_2, U_3 and U_5 of globally unique indices 2, 3, and 5. Typically in FEM implementations, a mapping of local to global node numbering needs to be defined. Another important aspect of interpolation polynomials is that the gradient of the field within each element can also be approximated as:

$$\nabla U^s(x,y,z) = \sum_{k=1}^{n} U_k^s \nabla P_k^s(x,y,z), \quad \forall s \in \mathcal{S}. \tag{2.17}$$

The specifics of the interpolation functions vary for each FEM scheme; however, they need to maintain the important property of ensuring the continuity of the U field across the finite element edges.

Since the functional $\mathcal{F}$ in (2.15) can, up to a constant, be viewed as an expression of the total energy stored in the system; therefore, $\mathcal{F}$ can be expressed using the sum of energies of each finite element as follows:

$$\mathcal{F}(U) = \sum_{s \in \mathcal{S}} \mathcal{F}_s(U_s) \tag{2.18}$$

where $\mathcal{S}$ is the total elements' set spanning the domain Ω; U_s is the set of the field discrete unknowns, or degrees of freedom, for the finite element s; and $\mathcal{F}_s$ is the energy contribution of s. Therefore by rendering stationary the functional in (2.18), an approximate solution to the Helmholtz equation can be found using the FEM scheme. Substituting each of (2.16) and (2.17) into (2.15) and applying a standard derivation process, an expression for the local finite element $\mathcal{F}_s$ can, generally, be formulated as:

$$\mathcal{F}_s(U_s) = \frac{1}{2} U_s^T M_s U_s - B_s^T U \tag{2.19}$$

in which we refer to M_s as the element characteristic matrix with dimensions n-by-n where n is the number of unknowns per element; and B_s is the element source vector. It is clear that (2.19) can also incorporate the essential boundary conditions as stated for the BVP when $\mathcal{F}_s$ represents a boundary finite element.

Finally, the FEM solution is computed by executing three main procedures. First, all the contributions from all the finite elements in terms of the M_s and the B_s elements are added up into a global and sparse linear system of equations. This process is referred to as the assembly process. Second, the essential boundary conditions are incorporated into the linear system. Last, the resulting linear system of the form (2.1) is solved using sparse linear solvers such as the PCG method.

2.2.2 The FEM Parallel Acceleration Issues

Conventional FEM implementations consist of two computationally expensive stages which are the sparse matrix A assembly stage, and the solving stage of the linear system using iterative solvers. Acceleration of these stages using parallel processing will be limited by memory bandwidth issues due to their dependency on the global sparse data-structure of the matrix A. This scalability issue has been the focus of extensive research lately, because computing manufacturers are increasing their chip's computational throughput by adding more cores rather than frequency scaling due to power limitations at the chip level. Therefore, in order to obtain good performance from emerging parallel computing architectures, the FEM computation must scale well with parallel computing. To better illustrate this issue, we consider the PCG method, which is a widely used iterative solver for FEM applications. The PCG solver requires the execution of a number of global algebraic operations in each iteration such as a SMVM and a preconditioner solve. The SMVM operation, in particular, can strongly impact the overall performance of the PCG solver since the memory access time

will dominate the computational time due the underlying sparse data-structure. Considering the SMVM defined as $y = Ax$, where A is a symmetric sparse matrix while x and y are dense vectors, the operation, in its simplest implementations, can be executed as a number of concurrent vector dot products where each dot product consists of x and a row from A. The result from each dot product updates a single value in the output vector y. The dot products are independent of each other, hence they can be executed in parallel. While this SMVM approach seems very intuitive and convenient to implement, it actually results in a very poor performance for key reasons. The data within each row of A is very sparse and require irregular memory access resulting in very low cache hit rates, which substantially increases the CPU's idle time by waiting for data to arrive, causing the CPU to attain only a small fraction of its peak performance. Without improving the SMVM execution on a single CPU, parallelizing the SMVM will not sustain the potential speedup of the parallel platform. In fact, this issue is even more pronounced for multicore architectures. As the number of cores increases, the memory requests consequently increases causing a greater bottleneck for the limited memory bandwidth resources. This in turn severely limits the parallel scalability of the SMVM operations as will later be demonstrated in the numerical results of Section 6.4.

Many attempts are made to improve the performance of the SMVM; however, typical implementations of the SMVM yield poor performance of no more than 10-20% of the peak CPU throughput [4]. Improving the performance of the SMVM kernel will require sophisticated programming techniques, code transformations, and data-structures that are tailored to specific CPU, cache and memory architectures [24]. Specialized sparse data-structures are devised in order to either exploit the sparsity structure of matrices resulting from specific application areas, or target specific HPC architectures. For example, a recent work in [87] uses blocking schemes to obtain better performance on multicore architectures. The work

in [22,88] uses stripe-based sparse data-structures to accelerate the SMVM on FPGAs using a highly parallel hardware architecture based on a systolic pipeline of processing elements. The numerical results of these efforts indicate that sustaining good performance for all sparse matrices, even within the same FEM application area, is nearly impossible. While there also exist generic and optimized libraries such as Trilinos [29] and PETSc [89], which can be used to solve the sparse linear system in parallel, obtaining a sustained performance can prove difficult due to the varying sparsity structure of A resulting from different application areas. In addition, such libraries do not help with the assembly stage of the FEM, which in many cases can require more time than the solve stage, such as the case for non-linear applications. In certain cases, matrix reordering can improve the performance of SMVM by rearranging its elements so that they are more clustered around the main diagonal. One of the most popular reordering algorithms to improve the SMVM performance is the Reverse Cuthill-McKee (RCM) algorithm [90]. However, reordering requires considerable overhead processing for large matrices.

Another important sparse operation is the sparse matrix assembly and the buildup of the sparse data-structure based on the matrix specific sparsity pattern. For specialized data-structures, this operation can be very costly. Parallelizing the matrix assembly also requires special care. Since the sparse matrix is a shared data-structure, updating the data-structure from different parallel processes needs to avoid memory collisions when accessing shared elements in the matrix. One way to overcome this issue is by using element coloring schemes, where adjacent elements are given different colors. Elements of the same color group can then be assembled concurrently. While, for certain cases, the assembly needs to be done only once and, therefore, its cost can be tolerated; however, for many cases such as AMG solvers, adaptive refinement, and non-linear applications, the assembly stage can dominate the solve stage. This is specially so for non-linear applications requiring non-linear

solvers, such as the Newton-Raphson method, where the assembly stage must be repeated each linearizing iteration. Our numerical results in Section 6.4 demonstrate that the assembly time was considerably long in comparison with the solve time.

As mentioned earlier, computing manufacturers are increasing the number of CPU cores on a single chip as opposed to frequency scaling in order to keep up with Moore's law [91] of increasing chip transistor density; therefore, new and inherently parallel algorithms need to be devised for the FEM in order to efficiently scale with these parallel computing trends. In this work, we take a different approach for FEM computation by directly minimizing the energy functional (2.18) using a newly derived FGaBP algorithm. The FGaBP algorithm is based on computational inference on GMs that exploits the inherent structure of the FEM problem resulting in localized computation involving dense matrices of very small sizes. The overall convergence of the algorithm progresses by passing update messages between processing nodes in a flexible manner allowing adaptable memory bandwidth utilization for various HPC platforms. The FGaBP eliminates the need to generate any large sparse matrices, or use any form of sparse data-structures, by eliminating all the global algebraic operations, including SMVMs, proving great potential for scalable parallel computational throughput. Not only the new algorithm efficiently parallelizes the solving stage, but also it allows for an embarrassingly parallel implementation of the assembly stage using only dense and contiguous data-structures.

2.3 Graphical Models

In this section we introduce an overview of the GM concept, which is central in the development of the FGaBP algorithm in Chapter 5. A GM is a graphical representation of a multivariate probability distribution describing a particular problem [48]. The GM captures

the interactions of the random variables which represent the connectivity structure of the underlying problem. Specifically, the GM can assist in the analysis of such problems that require the computation of local parameters about the problem's latent random variables such as the means of the latent random variables or even the joint mean of a subset of latent random variables. For that purpose, GMs can greatly assist in devising inference algorithms that can reuse intermediate results, or exhibit distributed computations that are potentially efficient for parallelism. For example, executing an inference algorithm such as the BP algorithm, described in more details in Section 2.4, on a tree structure GM can result in optimal execution where only a single computational step is performed per variable regardless of the tree size. However, practical problems result in more complicated GMs than simple trees, where such models can contain loops or clique substructures. Inference on such GMs can take an approximate form with larger number of iterations than loop-free GMs. The number of iterations on loopy GMs is typically proportional to the number of variables in the model.

A GM is composed of a set of vertices (or nodes) $\mathcal{V}$ connected by a set of edges $\mathcal{E}$, which is denoted as $\mathcal{G}(\mathcal{V}, \mathcal{E})$. The vertices are denoted by indices such that $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$. The edges are links between pairs of vertices and are denoted as follows: given a set of pairs $\mathcal{E} = \{e_{ij} = (v_i, v_j) \mid v_i, v_j \in \mathcal{V}\}$ such that $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$, where $\times$ is the Cartesian product. GMs can either be undirected or directed. In undirected GMs we have the condition: if $e_{i,j} \in \mathcal{E}$ then $e_{j,i} \in \mathcal{E}\, \forall v_i, v_j \in \mathcal{V}$; while, such a condition is not necessarily true for directed GMs. The term undirected edge refers to the fact that diagrammatically, the edges $e_{i,j}$ and $e_{j,i}$ are represented using a single edge without an arrow indicating any particular direction. Fig. 2.7 illustrates examples of directed and undirected GMs. In this work, we only utilize undirected GMs for all our developments of parallel algorithms.

Using a GM of a particular distribution, an inference algorithm can be devised based on the interactions between the latent variables in the model as well as the information

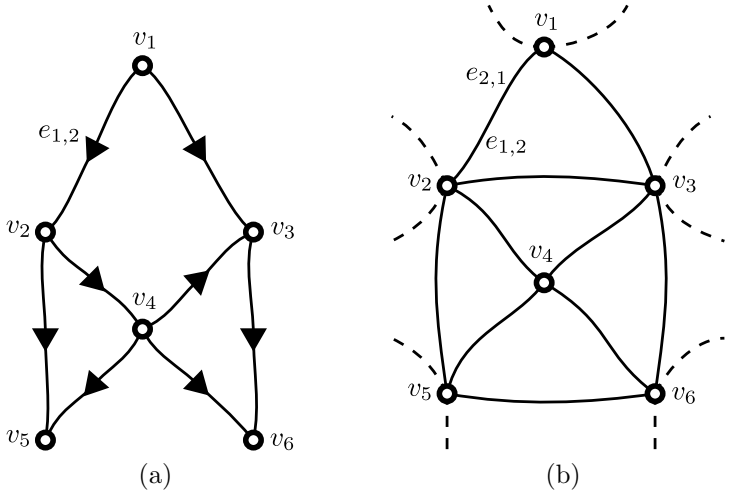


Fig. 2.7 Examples of directed and undirected GMs. (a) Loop free directed GM. (b) Loopy undirected GM.

propagating along its edges, therefore the following additional notations are required in order to simplify the formulation. A random variable U located on a vertex v_i is denoted as U_i. A message m sent on an edge $e_{i,j}$ from vertex i to vertex j is denoted as m_{ij}. It is worth noting that in undirected GMs where the edges $e_{i,j}$ and $e_{j,i}$ coexist, it is not necessarily true that the messages m_{ij} and m_{ji} are equal.

In essence, the messages in a GM describe a probability distribution in terms of either the target node variable $m_{ij}(U_j)$ or the source node variable $\eta_{ij}(U_i)$ which actually constitutes the flow of information in the GM. Sometimes, we are interested in the joint distribution of a subset of random variables; if no confusion can arise, a subset of variables can simply be denoted as follows: if $c = \{1, 2, \cdots, n\}$ then $U_c = \{U_1, U_2, \cdots, U_n\}$. Similarly, a subset of vertices is denoted as $v_c = \{v_1, \cdots, v_n\}$. The number of individual variables in U_c is referred to as the cardinality of U_c and is denoted by $|U_c|$.

A clique $C(v_c)$ in a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a subset of nodes v_c that are fully connected such that if $v_i, v_j \in v_c$, then $e_{i,j} \in \mathcal{E}$. A clique is maximal if, and only if, there is no other node

from outside the clique that can be added to it and still form a clique. In other words, only a nonmaximal clique is a proper subset of a maximal clique. The simplest example of a clique is a clique formed by any pair of nodes connected by an edge. Also, any three nodes connected by three undirected edges form a clique.

Undirected GMs represent a class of distributions that can be factorized based on cliques in the graph. Let $\mathcal{C}$ be the set of maximal cliques in an undirected GM, then a family of distributions can be defined as follows:

$$\mathcal{P}(U) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \Psi_C(U_c) \tag{2.20}$$

where Z is a normalizing constant and $\Psi_c(U_c)$ is usually referred to as a compatibility function which also can be regarded as a local distribution of the clique variables U_c. Such families of GMs are referred to as Markov Random Field (MRF) or Gibbs distributions [36]. The maximal cliques requirement for (2.20) can sometimes be relaxed to nonmaximal cliques. For example, consider the following factorized distribution:

$$\mathcal{P}(U) = \frac{1}{Z} \prod_{(i,j) \in E} \Psi_{i,j}(U_i, U_j) \prod_{i \in V} \Phi_i(U_i) \tag{2.21}$$

where $\Psi_{i,j}(U_i, U_j)$ is referred to as the pairwise compatibility function and $\Phi_i(U_i)$ is the self potential function. Distributions of this form can be represented by a GM that is pairwise connected, which we refer to as Pairwise Graphical Model (PWGM), where each vertex represents a variable node.

Since, in general, distributions corresponding to loopy undirected GMs contain factors that have coupled, or correlated, variable sets, inference on such models is not necessarily a trivial task. However, the factorization structure of these distributions, as exposed by the

GM, can be exploited by certain inference algorithms to produce computationally efficient algorithms.

Distributions of the form (2.20) are more conveniently represented using a special type of graph referred to as a Factor Graph (FG) [35]. FGs are bipartite graphs denoted as $\mathcal{G}(\mathcal{V}, \mathcal{F}, \mathcal{E})$, where there are two types of vertices, the vertex $v_i \in \mathcal{V}$ representing the variable nodes and the vertex $f_a \in \mathcal{F}$ representing the compatibility (or factor) functions $\Psi_a(U_a)$. An edge $e_{i,a}$ in a FG connects a vertex $v_i \in \mathcal{V}$ to a vertex $f_a \in \mathcal{F}$ only when the corresponding vertex variable U_i is an argument of the factor function $\Psi_a(U_a)$. Fig. 2.8 shows a PWGM model and the corresponding FG with factor functions based on maximal cliques. FGs are particularly useful when nonmaximal cliques are also configured; in such a case, different message update algorithms can be devised for the same problem resulting in different implementations with different numerical properties. This feature will be exploited by our FG model for the FEM problem, the FEM-FG model, later introduced in Chapter 5. The FEM-FG model can be used to produce variants of the FGaBP algorithm, later described in Section 5.6, which has higher memory communication efficiency.

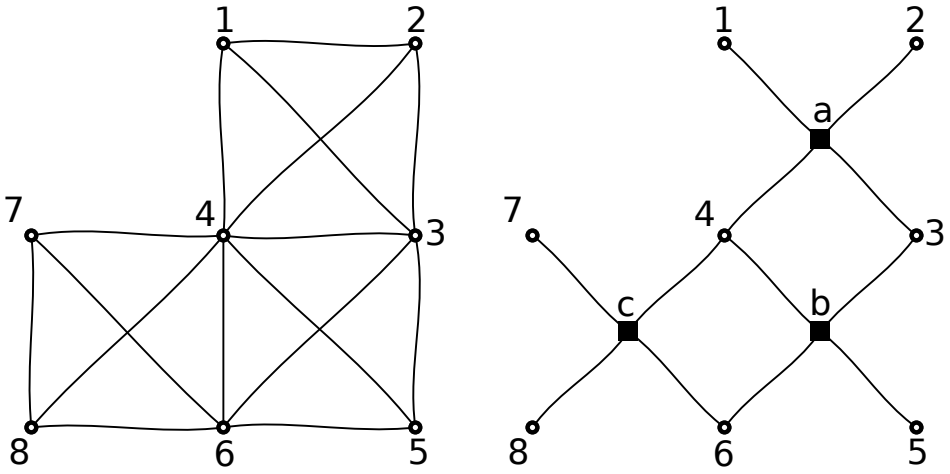


Fig. 2.8 The PWGM (or MRF) on the left and the corresponding FG model configured with maximal cliques on the right, variable nodes are represented by circles and factor nodes are represented by squares.

For general GMs, we will denote the number of variable nodes or vertices as N_v, the number of edges as N_e and the number of factor nodes, when present, as N_f. In addition, we will typically use small letters such as n and n_e to denote the local factor or clique quantities such as the number of vertices and edges correspondingly.

2.4 The Belief Propagation Algorithm

The Belief Propagation (BP) algorithm, as proposed by Pearl in [31], is a message passing algorithm on GM that efficiently computes the marginal distribution of each variable node by recursively sharing intermediate results. If the GM is a tree, then BP is guaranteed to converge to exact marginals. However, if the GM contains cycles, as typically the case in many practical applications, then BP takes an iterative form, referred to as Loopy Belief Propagation (LBP), which can be used to obtain an approximation for the marginals [31–34]. BP recently showed excellent empirical results in certain applications, such as machine learning, expert systems, computer vision, and channel decoding [35–45].

2.4.1 Belief Propagation on Factor Graph Models

Executing BP on FGs consists of passing two types of messages. A factor node message m_{ai} sent from a factor node Ψ_a to a connected variable node U_i; and a variable node message η_{ia} sent back from the variable node U_i to the factor node Ψ_a. BP messages on FG models are probability distributions; such that, the factor node message m_{ai} constitutes a distribution in terms of the continuous random variable U_i, or the most probable state of U_i, as observed from the factor node f_a with local function Ψ_a. In return, the variable node message η_{ia} constitutes a distribution in terms of U_i by combining messages from all connected factor nodes excluding the message from the factor f_a. The general BP update rules on FGs are

formulated as follows:

$$m_{ai}^{(t)}(U_i) \propto \int_{U_{\mathcal{N}(a)\setminus i}} \Psi_a(U_a) \prod_{j \in \mathcal{N}(a)\setminus i} \eta_{ja}^{(t_\star)}(U_j)\ \mathrm{d}U_{\mathcal{N}(a)\setminus i} \tag{2.22}$$

$$\eta_{ia}^{(t)}(U_i) \propto \prod_{k \in \mathcal{N}(i)\setminus a} m_{ki}^{(t_\star)}(U_i) \tag{2.23}$$

$$b_i^{(t)}(U_i) \propto \prod_{k \in \mathcal{N}(i)} m_{ki}^{(t)}(U_i) \tag{2.24}$$

where t and $t_\star$ are iteration counts such that $t_\star \leq t$; $\mathcal{N}(a)$ is the set of all node indices connected to node a, referred to as the neighborhood set of node a; $\mathcal{N}(a) \setminus i$ is the neighborhood set of a minus node i index; $b_i(U_i)$ is referred to as the belief at node i. The integral in (2.22) is a multidimensional integral of the random variable set $U_{\mathcal{N}(a)\setminus i}$ each integrated over all of their possible values.

It is important to note that message updates are performed according to a particular schedule. For example, if we would use a fully parallel schedule, that is $t = t - 1$, then all factor and variable nodes will update their messages simultaneously based on the message values of the previous iteration; this update schedule is also referred to as synchronous update. Alternatively we could traverse each variable or factor node sequentially based on a particular order, and then compute new messages based on the most recently available messages; this update schedule is also referred to as asynchronous updates. Such a schedule does not offer the most potential for parallelism, but it was found empirically to exhibit the fastest to converge rate [66].

2.4.2 Belief Propagation on Pairwise Graphical Models

Considering PWGMs resulting from distributions with pairwise compatibility functions as shown in (2.21), each node sends and receives messages along all the pairwise edges connected to it. Since variable nodes only communicate with other variable nodes of the same type, there can be only one type of message computed along the pairwise edges. The general BP update rule for PWGMs is stated as follows:

$$m_{ij}^{(t)}(U_j) \propto \int_{U_i} \Psi_{ij}(U_i, U_j)\Phi_i(U_i) \prod_{k \in \mathcal{N}(i)\setminus j} m_{ki}^{(t_\star)}(U_i)\mathrm{d}U_i. \tag{2.25}$$

The integral in (2.25) is taken for the single random variable U_i over all its possible values. A message $m_{ij}(U_j)$ is communicated from variable node v_i to variable node v_j over the edge $e_{i,j}$ using messages previously received from the neighborhood $\mathcal{N}(i) \setminus j$. Once the messages converge, the marginal distribution for each variable can be obtained as:

$$b_i^{(t)}(U_i) \propto \Phi_i(U_i) \prod_{k \in \mathcal{N}(i)} m_{ki}^{(t)}(U_i). \tag{2.26}$$

2.4.3 The Gaussian Belief Propagation Solver on Pairwise Graphical Models

In this section we present a brief overview of the Gaussian Belief Propagation on Pairwise Graphical Models (PW-GaBP) [33, 54] which was a recently introduced as a solver for linear systems of equations based on BP executed on PWGM. While our initial experimentations with PW-GaBP for linear systems resulting from FEM applications reveal that its performance does not match the existing state-of-the-art solvers such as IC-PGC and multigrid based solvers for large systems; it does offer however an important insight on the potential of using BP style algorithms to parallelize the FEM problem as discussed in Chapters 3 and

4. In addition, the PW-GaBP requires the assembly of the large sparse linear system as in (2.1) or the sparse matrix A, which is a costly step in the FEM process.

A Gaussian PWGM model is formulated by taking the exponential of the negated quadratic, defined in (2.2), as:

$$\mathcal{P}(U) = \frac{1}{Z} \exp(-\frac{1}{2}U^T AU + b^T U + c) \tag{2.27}$$

where Z is a constant. It can be shown that the solution of the linear system $u_\star = A^{-1}b$ is the maximum point of $\mathcal{P}$ which is also the point where the quadratic (2.2) is minimum. The exponential expression in (2.27) is a multivariate Gaussian probability distribution where the Gaussian mean vector μ equals the solution to the linear system $\mu = A^{-1}b$. Hence the solution to the linear system can alternatively be obtained by executing the BP algorithm on the Gaussian PWGM represented by $\mathcal{P}$ in order to obtain its marginal means.

The Gaussian PWGM is constructed using the sparse matrix A as follows. The matrix A can be viewed as an undirected graph where each nonzero element ($A_{ij} \neq 0$) represents an undirected edge between a pair of variable nodes indexed as v_i and v_j. Equivalently, we can factor the graph's distribution $\mathcal{P}$ into nodal functions $\phi_i(U_i) \triangleq \exp(-\frac{1}{2}A_{ii}U_i^2 + b_i U_i)$ and edge functions $\psi_{i,j}(U_i, U_j) \triangleq \exp(-\frac{1}{2}U_i A_{ij} U_j)$. BP executing on Gaussian PWGM results in messages communicated between variable nodes as follows. Each node v_i computes a new message towards node v_j on a particular edge $e_{i,j}$ using all messages received from nodes in the neighborhood $\mathcal{N}(i)$ of node v_i excluding the message received from v_j. The message update from each node is performed either sequentially or concurrently subject to a specific schedule. Since the underlying distribution is Gaussian, the belief updates will be based on

propagating only two parameters α and β, as shown below:

$$\alpha_{ij}^{(t)} = -A_{ij}^2(\alpha_{i\setminus j}^{(t_\star)})^{-1} \tag{2.28}$$

$$\beta_{ij}^{(t)} = -A_{ij}\beta_{i\setminus j}^{(t_\star)}(\alpha_{i\setminus j}^{(t_\star)})^{-1} \tag{2.29}$$

where:

$$\alpha_{i\setminus j}^{(t_\star)} = \alpha_i^{(t_\star)} - \alpha_{ji}^{(t_\star)} \tag{2.30}$$

$$\beta_{i\setminus j}^{(t_\star)} = \beta_i^{(t_\star)} - \beta_{ji}^{(t_\star)} \tag{2.31}$$

and:

$$\alpha_i^{(t)} = A_{ii} + \sum_{k\in N(i)} \alpha_{ki}^{(t)} \tag{2.32}$$

$$\beta_i^{(t)} = b_i + \sum_{k\in N(i)} \beta_{ki}^{(t)}. \tag{2.33}$$

For large sparse systems, the overall PW-GaBP computational complexity per iteration is $O\big(lN\big)$, where N is the number of unknowns and l is a constant ($l \ll N$) determined by the sparsity of the underlying problem, for example the average number of links per node. The marginal means, constituting the approximate solution, can then be computed by:

$$\bar{U}_i^{(t)} = \frac{\beta_i^{(t)}}{\alpha_i^{(t)}}. \tag{2.34}$$

GaBP was shown in [52] to converge for a particular class of matrices referred to as the walk-summable model. The walk-summability condition states that the spectral radius of the normalized off-diagonals of A is less than 1 in the absolute sense, that is $\rho(|I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}|) <$

1, where D is the diagonal elements of A. Such class of matrices includes the symmetric positive-definite diagonally dominant systems that arise from many FEM applications. PW-GaBP is one of the algorithms implemented by the parallel library Graphlab [63] aimed primarily at machine learning applications.

2.5 Convergence Testing

In practical implementations of the Gaussian BP algorithms, the l^2-norm of the differences in the successive approximate solutions, the relative norm for short, can be used as a computationally efficient measure to test the convergence of the algorithm. The relative norm (e_r) is computed as follows:

$$\begin{aligned} e_r^{(t)} &= \frac{\| \bar{u}^{(t)} - \bar{u}^{(t-1)} \|_2}{\| \bar{u}^{(t)} \|_2} \\ &= \left(\frac{\sum_{i=1}^{N} (\bar{u}_i^{(t)} - \bar{u}_i^{(t-1)})^2}{\sum_{i=1}^{N} (\bar{u}_i^{(t)})^2} \right)^{\frac{1}{2}} \end{aligned} \tag{2.35}$$

where (t) is the iteration number, $\bar{u}^{(t)}$ is the solution vector estimate at iteration (t) and $\bar{u}_i$ is the solution estimate at node i. The algorithm can be terminated when the relative norm reaches a certain tolerance value. The relative norm is also used for our development of the faster convergent relaxed PW-GaBP algorithm introduced in Chapter 4.

Another measure for convergence is the normalized residual l^2-norm, which is computed as follows:

$$R^{(t)} = \frac{\| b - A\bar{u}^{(t)} \|_2}{\| b \|_2} \tag{2.36}$$

The residual R provides an upper bound for the relative norm; however, the relative norm is used in practice for PW-GaBP to test convergence since it is less costly to compute

by avoiding the SMVM operation ($A\bar{u}^{(t)}$). However, in CG and multigrid algorithms, the residual is typically used for convergence testing since it is iteratively computed as part of these algorithms. In addition, the residual is used as the final convergence verification criteria in our experiments when we compare different algorithms.

2.6 The Condition Number and Diagonal Dominance

The Condition number of a matrix is defined as:

$$k(A) \triangleq \frac{\lambda_{max}(A)}{\lambda_{min}(A)} \tag{2.37}$$

where $\lambda_{max}(A)$ and $\lambda_{min}(A)$ are the largest and smallest eigenvalues of A. In numerical linear algebra, the condition number measures the sensitivity of the solution of the linear system to small perturbations in A. It may also be used to bound the convergence rate of iterative solvers of linear systems. Well-conditioned matrices have $k(A) \approx 1$ while ill-conditioned matrices can have a much larger condition number.

The matrix A is known to be Weakly Diagonally Dominant (WDD) if

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \quad \text{for all } i, \tag{2.38}$$

with strict inequality ($>$) for at least one index i. If the inequality is replaced by ($>$) for all i, then the matrix A is referred to as Strictly Diagonally Dominant (SDD).

2.7 Speedup

The execution time speedup (SU) is the performance measure used in our experiments. The SU is defined as the ratio of the execution times of two algorithms or implementations and computed as follows:

$$\text{SU} = \frac{\text{execution time of algorithm } Y}{\text{execution time of algorithm } X} \tag{2.39}$$

which states the speedup of algorithm X over algorithm Y. If algorithm X is the parallel implementation of a sequential algorithm Y, then this is referred as the parallel SU.

CHAPTER 3

Schedule Implementations for Gaussian Belief Propagation

In this chapter, we introduce implementation-oriented scheduling schemes for the Gaussian Belief Propagation on Pairwise Graphical Models (PW-GaBP) solver aimed at large sparse linear systems obtained from FEM applications. We analyze different message scheduling schemes used to enhance implementations, CPU time, and parallel executions. Our empirical experiments show that PW-GaBP when used with Strictly Diagonally Dominant (SDD) matrices can potentially result in execution-times that are competitive with the widely used Diagonally-Preconditioned Conjugate Gradient (D-PCG) solver.

We present empirical results of the Sequential Update Schedule (SUS) of the PW-GaBP solver. Compared with the D-PCG solver, the SUS demonstrates considerable reduction in iteration count for large sparse SDD linear systems. Next, we present the Hybrid Update schedule (HUS) for the PW-GaBP. The HUS algorithm uses an appropriate mix of sequential and parallel message updates. The parallel implementations of the HUS demonstrates that, the HUS facilitates efficient parallel execution while demonstrating faster convergence rates typical of PW-GaBP using SUS.

3.1 Introduction

In this work, we present various scheduling implementations for the PW-GaBP to solve large sparse linear systems arising from the FEM problem. We, therefore, assume that the sparse matrix A for the underlying FEM problem is already assembled by other FEM software such as GetFEM [28]. The computational efficiency of a PW-GaBP implementation will depend largely on two factors: the data-structure used to store the nodes' connectivities as provided by the Pairwise Graphical Model (PWGM) along with communicated messages, and the message transfer medium such as the memory bandwidth for shared-memory architectures or the network bandwidth for CPU-clusters. From an implementation perspective, three distinct stages can be designated for the nodal message update process of the PW-GaBP algorithm. First, the node receives messages from sparse connections. Second, the node performs local computations using the received messages. Last, the node responds with new messages on the same sparse connections. For CPU implementations, the choice of the data-structure required to store the messages will have a critical impact on the overall performance of the PW-GaBP algorithm due to the predominant message access time, data-locality and vectorization of computational loops. However, sparse matrices arising from typical FEM applications can exhibit a banded sparsity structure when proper reordering algorithms are used, which can be exploited to increase the parallel PW-GaBP performance.

PW-GaBP message updates are performed subject to a particular schedule. Two basic scheduling schemes common in general BP algorithms are the SUS, and the Parallel Update Schedule (PUS), also known as flooding. In the SUS, nodes are processed in sequence according to a particular ordering; while messages are propagated sequentially to receiving nodes in the same iteration. Hence, the SUS scheme provides little opportunity for parallel processing. The PUS, on the other hand, facilitates fully concurrent execution of the nodes;

however, the PUS requires a considerably larger number of iterations compared to SUS. Specifically, the empirical analysis in [66, 92] and other references within, indicates that the convergence rate of the SUS is upper-bounded by the convergence rate of the PUS. The work in [66, 67] also uses an informative based scheduling scheme which is used to accelerate convergence; in addition this scheme, in certain cases, can achieve convergence for models where basic BP fails. However, this work mainly uses discrete-domain probabilistic models of relatively small size compared to the large models resulting from FEM applications where the number of real-domain variables range in the order of millions or billions. Therefore, we believe that the BP scheduling for FEM problems should be treated as an implementation feature to enhance parallelism of the algorithm rather than to improve the convergence rate. The convergence rate, on the other hand, can more efficiently be treated using a specialized multigrid scheme for BP as we later show in Chapter 6.

This chapter is organized as follows. In Section 3.2 we present the SUS implementation of the PW-GaBP algorithm. In Section 3.3 we present the PUS implementation. In Section 3.4 we present the HUS implementation. Finally, in Section 3.5 we conclude with performance results.

3.2 Sequential Update PW-GaBP

In Algorithm 3 we present a particular implementation of PW-GaBP which makes efficient use of contiguous data-structures such as queues or stacks to store and retrieve messages. This particular implementation uses the SUS scheme. Using simple data-structures such as stacks or queues results in constant-time per message access complexity for retrieval and insertion, reducing the algorithm's overall message processing complexity per iteration to $O\big(nnz\big)$, where nnz is the number of non-zeros in A. Using such data-structures is facilitated

by adding the source node pointer (n_i) and the edge element parameter ($A_{i,j}$) to the message. Stacks are used in all our implementations; however, queues can also be used instead of stacks without much impact on the algorithm's performance. It is important to note that this implementation results in a fixed increase in the memory requirement per each message. However, since this fixed increase in the message size is independent of the problem's overall size N, where N is the number of variables, the algorithm's overall memory complexity scales linearly with N as expected for sparse problems.

In the SUS, the nodes need to be processed according to a particular order. In addition, the initialization stage is critical for the correct operation of the algorithm. As shown in lines 2 to 6, a unique nodal ordering (κ) needs to be defined. The nodes are also initialized with zero messages subject to the ordering κ. In this algorithm, a single processing node is assumed that has access to the full data-structure of A. While the SUS is not practical for parallel implementations, its study provides a hypothetical lower bound on the iteration count of the PW-GaBP solver when used with linear systems resulting from FEM applications.

3.3 Parallel Update PW-GaBP

The PUS version of the PW-GaBP algorithm is shown in Algorithm 4. This algorithm uses arrays of local static memory to store the node's edge messages. The local memory array provides constant time access on shared-memory machines by assigning a specific memory location to each destination node. To facilitate concurrent execution of all nodes, each node contains two message arrays, a current-message array that stores messages received in the previous iteration and is used to compute the new messages in the current iteration, and a new-message array that is used to store the new messages being received from the edge

Algorithm 3 The sequential update PW-GaBP implementation-oriented algorithm.

Initialize:
Define node ordering: κ
for each edge $A(i,j)$ **do**
  if $n_i <_\kappa n_j$ **then**
    $n_i.\,\text{push}(n_j.\,\text{pointer}, A_{ij}, P_{ji}=0, \mu_{ji}=0)$
  end if
end for
Compute:
repeat {iterations}
  for each node $n_i \in \kappa$ **do**
    $P_{agg} = A_{ii} + \sum_k P_{ki}$
    $\mu_{agg} = b_i + \sum_k \mu_{ki}$
    for each received message $n_j \rightarrow n_i$ **do**
      $\mu_{ij} = -A_{ij}(P_{agg} - P_{ji})^{-1}(\mu_{agg} - \mu_{ji})$
      $P_{ij} = -A_{ij}^2(P_{agg} - P_{ji})^{-1}$
      n_i.pop_message
      $n_j.\,\text{push}(n_i.\,\text{pointer}, A_{ij}, P_{ij}, \mu_{ij})$
    end for
    $\overline{x_i} = \mu_{agg} P_{agg}^{-1}$
  end for
until *convergence:* all $\overline{x_i}$ converged
Output: $\overline{x_i}$

nodes in the current iteration. Alternately, coloring schemes can be used to resolve memory collisions as proposed later in Chapter 6. At the beginning of each iteration, each node collects its edge messages into the new message buffer then swaps the current and the new message buffer pointers. Constant-time access data-structures such as queues and stacks can be used instead of static memory arrays. Also, there is no need to define any node ordering as the case in the SUS algorithm. Since the number of processing elements in computing environments is typically much less than the number of nodes, the PUS algorithm may not be practical for implementations targeted for large linear systems; nonetheless, the algorithm has conceptual importance for approximating the upper bound on the parallel iteration count.

Algorithm 4 The parallel update PW-GaBP implementation-oriented algorithm.

Initialize:
for each edge $A(i, j)$ **do**
  $n_i.\text{send}(n_j.\text{pointer}, A_{ij}, P_{ji} = 0, \mu_{ji} = 0)$
end for
Compute:
repeat {iterations}
  for each node n_i **do**
    $n_i.\text{swap\_pointers}(\text{currMessArray}, \text{newMessArray})$
  end for
  for each node n_i **do** {concurrent execution}
    Using the current-message buffers
    $P_{agg} = A_{ii} + \sum_k P_{ki}$
    $\mu_{agg} = b_i + \sum_k \mu_{ki}$
    for each received message $n_j \rightarrow n_i$ **do**
      $\mu_{ij} = -A_{ij}(P_{agg} - P_{ji})^{-1}(\mu_{agg} - \mu_{ji})$
      $P_{ij} = -A_{ij}^2(P_{agg} - P_{ji})^{-1}$
      $n_j.\text{send}(n_i.\text{pointer}, A_{ij}, P_{ij}, \mu_{ij})$
    end for
    $\overline{x_i} = \mu_{agg} P_{agg}^{-1}$
  end for
until *convergence:* all $\overline{x_i}$ converged
Output: $\overline{x_i}$

3.4 Hybrid Update PW-GaBP

Since in typical parallel computing environments the number of processing elements is limited, a degree of sequential processing will have to be performed for large problems. Our implementation-oriented HUS algorithm, shown in Algorithm 5, takes advantage of this imposed sequentiality to propagate faster message information, which reduces the PW-GaBP iteration count while exploiting parallelism in both single-node and multi-node HPCs.

By creating a partitioning scheme of relatively adjacent nodes, where each partition is assigned to a processing element to be executed in parallel, updates between nodes in different partitions can be performed using the PUS algorithm, while updates between nodes in the

same partition can be performed using the SUS algorithm. This flexibility allows the HUS algorithm to easily implement different sequential-parallel scheduling variations by varying the partition boundaries and the node ordering within each partition. That is, by choosing a partitioning and an ordering scheme that exploits node locality in terms of connectivity. The number of iterations increase due to parallel message updates can be expected to be considerably reduced, which will be demonstrated later in our results.

3.5 Results and Discussions

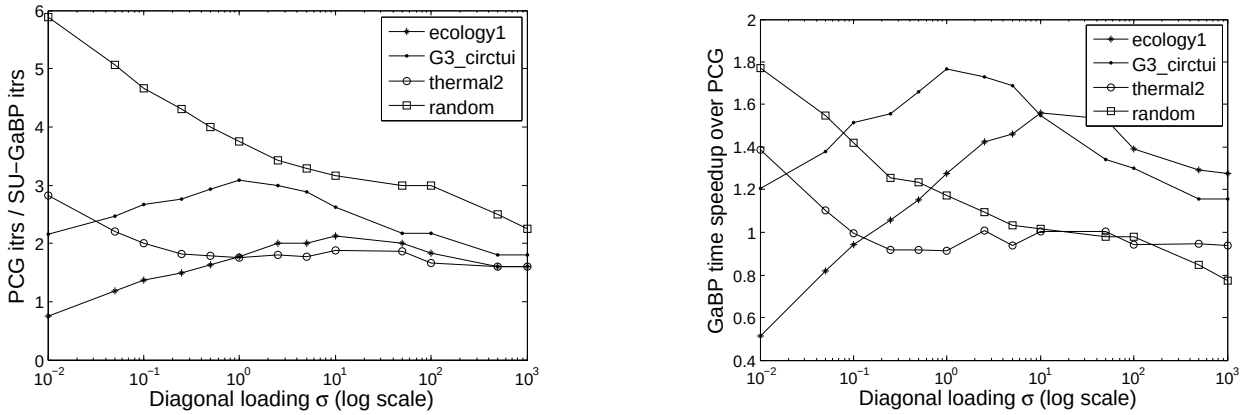
3.5.1 Performance Comparison with D-PCG

In order to assess the computational speed of the SUS, we compare it with the D-PCG algorithm. All runs are executed on single CPU workstation with Intel Core2 Quad @ 2.8GHz using double-precision computations. The D-PCG code was executed on the same CPU and it was obtained from the optimized GMM++ library [93]. Iterations were stopped when the l^2-norm of the residual reached $\epsilon = 10^{-9}$. The test matrices, shown in Table 3.1, are obtained from [94], with the exception of the matrix "Random" which is generated randomly using the Matlab software [95]. The matrix "thermal2" results from an unstructured steady-state FEM application. The matrices were made diagonally dominant by loading the diagonals with a uniformly distributed positive random number having a standard deviation $\sigma \in [10^{-2}, 10^{3}]$, in order to make the matrices conform with the walk-summability criteria [52] required for the PW-GaBP convergence. The plots in Fig. 3.1 show the performance results of the PW-GaBP solver using the SUS scheme against the D-PCG solver. Iteration reductions, up to 6×, are obtained by the SUS PW-GaBP algorithm. Also, the SUS implementation was able to achieve time speedups for many cases reaching up to 1.8×. It is worth noting that the obtained time speedup gains do not match the gains from the iteration reductions, which

Algorithm 5 The hybrid update PW-GaBP implementation-oriented algorithm.

Initialize:
Define node partitioning ζ
Define node ordering κ in each ζ
for each edge $A(i,j)$ **do**
  if $n_i =_\zeta n_j$ **then** {nodes in the same partition}
    if $n_i <_\kappa n_j$ **then**
      $n_i.\,\text{push}(n_j.\,\text{pointer}, A_{ij}, P_{ji}=0, \mu_{ji}=0)$
    end if
  else {nodes not in the same partition}
    $n_i.\,\text{send}(n_j.\,\text{pointer}, A_{ij}, P_{ji}=0, \mu_{ji}=0)$
  end if
end for
Compute:
repeat {iterations}
  for each node n_i on partition edge **do**
    $n_i.\,\text{swap\_pointers}(\text{currMessArray}, \text{newMessArray})$
  end for
  for each partition in ζ **do** {concurrent execution}
    for each node $n_i \in \kappa$ **do** {sequential execution}
      $P_{agg} = A_{ii} + \sum_k P_{ki}$
      $\mu_{agg} = b_i + \sum_k \mu_{ki}$
      for each received message $n_j \to n_i$ **do**
        $\mu_{ij} = -A_{ij}(P_{agg} - P_{ji})^{-1}(\mu_{agg} - \mu_{ji})$
        $P_{ij} = -A_{ij}^2(P_{agg} - P_{ji})^{-1}$
        if n_j in the same partition **then**
          $n_i.\,\text{pop\_message}$
          $n_j.\,\text{push}(n_i.\,\text{pointer}, A_{ij}, P_{ij}, \mu_{ij})$
        else
          $n_j.\,\text{send}(n_i.\,\text{pointer}, A_{ij}, P_{ij}, \mu_{ij})$
        end if
      end for
      $\overline{x_i} = \mu_{agg} P_{agg}^{-1}$
    end for
  end for
until *convergence:* all $\overline{x_i}$ converged
Output: $\overline{x_i}$

we attribute to the fact that the GMM++ implementation of the D-PCG algorithm has more efficient memory bandwidth utilization for sequential execution than our algorithm. However, the PW-GaBP algorithm is more tailored towards parallel implementations than



(a) (b)

Fig. 3.1 Speedup of the implementation-oriented PW-GaBP using the SUS scheme. (a) Iteration improvement of PW-GaBP over D-PCG. (b) PW-GaBP time speedup over D-PCG.

the PCG algorithm.

Table 3.1 Sparse test matrices.

Category	ecology1	G3_circuit	thermal2	random
N (nodes)	1,000,000	1,585,478	1,228,045	1,000,000
nnz	4,996,000	7,660,826	8,580,313	8,999,976

3.5.2 Hybrid-Update Scheduling Performance

The parallel behavior of the PW-GaBP algorithm using the HUS implementation was simulated on a single-core CPU. In order to simulate different partitioning and node ordering, matrix reordering techniques were used. Two common reordering techniques used here are: the Reverse Cuthill-McKee (RCM) [90], which reduces the matrix bandwidth; and Approximate Minimum Degree (AMD), which produces large blocks of zeros [75]. Fig. 3.2 shows the parallel results of our HUS implementation algorithm on the matrix "thermal2" as the number of parallel partitions increases from 1 to 4096. The HUS algorithm demonstrates

gradual rate of increase in iterations on the original unordered matrix, while ordered matrices using the RCM and the AMD algorithms showed a considerably lower rate of increase. These results demonstrate the HUS potential for parallel gains and its ability in exploiting the problem's connectivity structure by maintaining a lower iteration count than the PUS scheme. These results also suggest that good speedup gains can be obtained from implementing the HUS on CPU-clusters and many-core architectures in comparison with leading iterative methods such as the D-PCG algorithm.

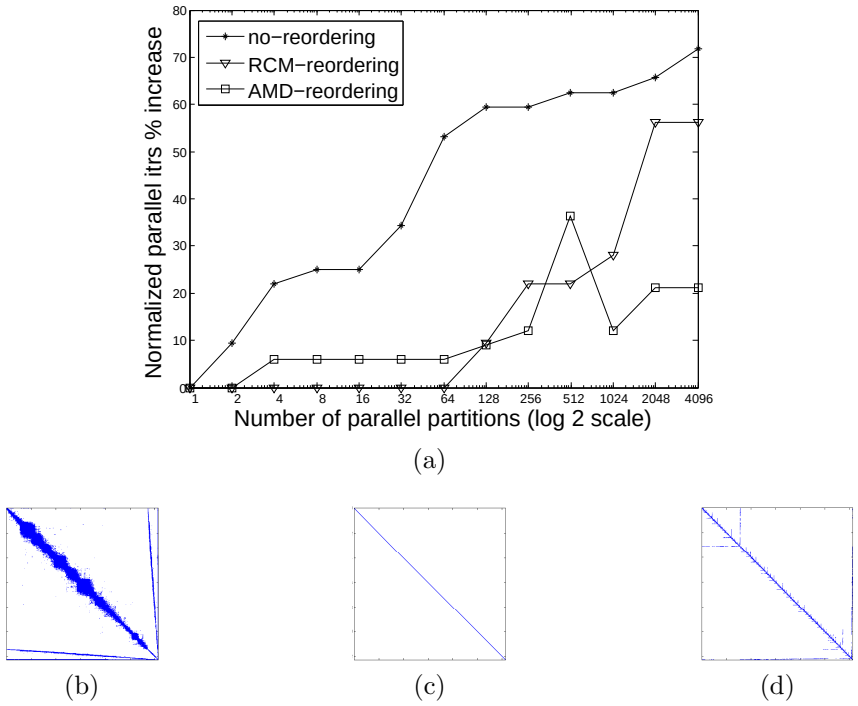


Fig. 3.2 The PW-GaBP algorithm results using the HUS scheme for the matrix "thermal2'. (a) The HUS parallel iteration increase rate. (b) Sparsity structure of original "thermal2." (c) RCM reordered. (d) AMD reordered.

3.6 Conclusion

Implementation-oriented algorithms of PW-GaBP were presented which demonstrated speedups of up 1.8× over the D-PCG algorithm. Also, both improvements in execution time and reduction in iteration count over the D-PCG solver were demonstrated using the PW-GaBP algorithm with the SUS scheme. While the PW-GaBP solver demonstrated an advantage over the D-PCG solver for SDD matrices, the performance for more general FEM matrices is clearly not competitive against leading solvers such as IC-PGC or multigrid; nonetheless, this study provides valuable insights for the later development and implementation of the FGaBP and the FMGaBP algorithms presented in Chapters 5 and 6 which fix the convergence rate issue. The HUS algorithm was introduced and its implementation demonstrated enhanced parallel performance for PW-GaBP. The HUS algorithm results showed considerable reductions in parallel iterations demonstrating great potential for efficient implementations on CPU-clusters and many-core HPC platforms.

CHAPTER 4

Relaxed Gaussian Belief Propagation

The solution developed in this chapter is motivated by the fact that the PW-GaBP solver requires a large number of iterations to converge if the sparse matrix is Weakly Diagonally Dominant (WDD) or is ill-conditioned. Such matrices can arise from many application domains including the FEM. In this work, we present a relaxed form of PW-GaBP that reduces the number of iterations, resulting in significant computational reduction (up to 12.7×) for ill-conditioned large linear systems. In addition, to circumvent the need of determining the relaxation factor for the new algorithm a priori, we propose a second algorithm that incrementally determines a suitable relaxation factor based on iterative error improvements that also results in similar reductions in PW-GaBP iterations. We show that the new algorithms can be implemented without any significant increase, over the original PW-GaBP, in both the computational complexity and the memory requirements. We demonstrate the advantages of our algorithms using empirical results of large, ill-conditioned, and WDD matrices.

4.1 Introduction

PW-GaBP was empirically found to exhibit fast convergence for problems where the inverse covariance matrix of the underlying multivariate Gaussian distribution, or similarity the matrix A of the sparse linear system, is Strictly Diagonally Dominant (SDD) [33,54], especially when compared to the D-PCG solver as shown in Chapter 3. However, if A is large, sparse and ill-conditioned, then PW-GaBP may require a large number of iterations. The work in [33] provides a rough upper bound on the number of iterations required by PW-GaBP to reach a given convergence tolerance ϵ; however, it is only applicable for SDD matrices. The work in [54] uses a common convergence acceleration method referred to as the Aitken-Steffensen's acceleration to speedup the PW-GaBP convergence. In the Aitken-Steffensen's acceleration scheme, an improved estimate of a message is computed at each third iteration as:

$$m \approx \hat{m}^{(t)} = m^{(t-3)} - \frac{(m^{(t-2)} - m^{(t-3)})^2}{m^{(t-1)} - 2m^{(t-2)} + m^{(t-3)}} \tag{4.1}$$

where $m^{(t)}$ represents a message estimate at iteration t. However, when this acceleration is used in PW-GaBP with ill-conditioned matrices, it yields unstable results even with double-precision implementations. In addition, this acceleration scheme requires the additional storage of prior iteration messages which makes it very costly when implemented for large problems.

In this study, we will consider ill-conditioned matrices that are not necessarily SDD, which require considerably larger number of PW-GaBP iterations. Finding a theoretical upper bound for the convergence rate of PW-GaBP for such matrices is still an open research question. In addition, some of the solutions developed in this chapter are not only applicable to PWGMs but, rather, can also be used for general GMs such as the ones developed for the FEM problem as will be illustrated in Chapter 5. However, since in this chapter the

numerical analyses of the new solution were all performed in comparison with the original PW-GaBP, we will present the new formulation as an extension of the PW-GaBP solver.

This chapter is organized as follows. In Section 4.2, we present the new relaxation scheme for the PW-GaBP algorithm. Section 4.3 presents the dynamic relaxation algorithm. Finally, in Section 4.4 we present the numerical results and concluding remarks.

4.2 The Relaxed PW-GaBP Algorithm

In this section we will detail the relaxation formulation of the PW-GaBP algorithm, which was previously presented in the background chapter in Section 2.4.2. We will proceed with our discussion whereby all the information available from the underlying problem is the matrix A. In addition, we consider matrices that are large, sparse, ill-conditioned, and WDD. With such matrices, the original PW-GaBP algorithm is known to require a very large number of iterations to converge.

4.2.1 Edge Message Relaxation

As can be seen from (2.30), (2.31) and (2.34), the marginal mean of node i, which is also the solution estimate of $U_i^{(t)}$ at iteration t, can be obtained using the two sums $\alpha_i^{(t)}$ and $\beta_i^{(t)}$ of messages received on all connected edges $\mathcal{N}(i)$. It can also be noted from these equations that applying any relaxation on the β messages does not affect the α messages convergence properties. Relaxation on β_{ij} can be applied as follows:

$$\hat{\beta}_{ij}^{(t)} = \gamma\beta_{ij}^{(t)} + (1-\gamma)\beta_{ij}^{(t-1)} \tag{4.2}$$

where γ is referred to as the relaxation factor. The relaxation factor γ is obtained from the limit $\gamma \in [0, 2]$. If γ is in $[0, 1]$, the method is referred to as under-relaxation or damping and,

in certain cases, is used to force a divergent PW-GaBP to converge at the expense of a high iteration count. If γ is in $[1, 2]$, the method is referred to as over-relaxation and is used to accelerate convergence, which is the objective of this work. The new relaxed message $\hat{\beta}_{ij}^{(t)}$ can be communicated instead of $\beta_{ij}^{(t)}$. It can be seen that if γ is chosen such that the PW-GaBP algorithm is convergent, the relaxed messages converge as $\beta_{ij}^{(t)} \approx \beta_{ij}^{(t-1)}$. This indicates that the stationarity point of the relaxed algorithm is that of the original PW-GaBP. We refer to this style of relaxation as Edge Message Relaxation (EMR). It is clear that this relaxation does not require additional memory since it only requires the previous iteration message. The EMR relaxation is the most flexible and can be applied to any GM.

4.2.2 Nodal Message Relaxation

An alternative strategy to EMR, we can apply relaxation to the nodal sum β_i messages as opposed to each individual edge message β_{ij}. The β_i messages are then relaxed as follows:

$$\hat{\beta}_i^{(t)} = \gamma \beta_i^{(t)} + (1 - \gamma)\beta_i^{(t-1)}. \tag{4.3}$$

We refer to this relaxation scheme as the Nodal Message Relaxation (NMR). Relaxing β_i messages requires additional memory of order $O\big(N\big)$ to store the previous iteration's $\beta_i^{(t-1)}$ value. However, based on the distinct observation that for every matrix we tested from the class of WDD matrices, the α messages converged much faster than the β messages to within 10 to 20 iterations as demonstrated by our empirical results. If we decide to use this property, we can eliminate the additional memory requirement due to relaxing β_i messages by reformulating the updates as follows:

$$\hat{\beta}_i^{(t+t_o)} = \alpha_i^* \hat{\mu}_i^{(t+t_o)} \tag{4.4}$$

where:

$$\hat{\mu}_i^{(t+t_o)} = \gamma \mu_i^{(t+t_o)} + (1-\gamma)\mu_i^{(t+t_o-1)} \tag{4.5}$$

where α_i^* is the fixed point reached after t_o iterations. The $\mu_i^{(t-1)}$ values are reused here since they are used to store the final solution.

To illustrate the correctness of the NMR based on the original PW-GaBP update rules of β_{ij} messages, we substitute (4.3) into (2.31) and then into (2.29) to obtain the following:

$$\hat{\beta}_{ij}^{(t+t_o)} = \frac{-A_{ij}}{\alpha_{i\backslash j}^*} \hat{\beta}_{i\backslash j}^{(t+t_o)} \tag{4.6}$$

where,

$$\hat{\beta}_{i\backslash j}^{(t+t_o)} = \hat{\beta}_i^{(t+t_o)} - \beta_{ji}^{(t+t_o)} \tag{4.7}$$

$$\begin{aligned} &= \gamma \beta_{i\backslash j}^{(t+t_o)} + (1-\gamma)\beta_{i\backslash j}^{(t+t_o-1)} \\ &\quad - (1-\gamma)\Delta\beta_{ji}^{(t+t_o)} \end{aligned} \tag{4.8}$$

and,

$$\Delta\beta_{ji}^{(t+t_o)} = \beta_{ji}^{(t+t_o)} - \beta_{ji}^{(t+t_o-1)}. \tag{4.9}$$

It can be seen from the above modified BP rule for β_{ij} messages that if γ is chosen such that the relaxed PW-GaBP is convergent, the additional terms $\Delta\beta_{ji}^{(t+t_o)}, \forall j$ will approach zero and the relaxed algorithm's fixed point will be equal to the fixed point solution of the original PW-GaBP algorithm. The listing of the relaxed PW-GaBP algorithm is shown

in Algorithm 6. We refer to this algorithm as the Relaxed Gaussian Belief Propagation (R-GaBP) algorithm.

Algorithm 6 The R-GaBP algorithm.

1: *Initialize:* $\forall i, j$
   $\alpha_{ij} = 0 \quad \beta_{ij} = 0$
   $\gamma \in [1, 2] \quad u_i^{(0)} = 0$
2: **repeat** {Start PW-GaBP iteration: $t = 1, 2, \cdots$}
3: **for** each node i **do**
4: $\alpha_i = A_{ii} + \sum_{k \in N(i)} \alpha_{ki}$
5: $\beta_i = b_i + \sum_{k \in N(i)} \beta_{ki}$
6: **if** $\frac{\|\Delta\alpha_i\|}{\|\alpha\|} < \epsilon \; \forall i$ **then**
7: $\hat{\beta}_i = \gamma\beta_i + (1-\gamma)\alpha_i\mu_i^{(t-1)}$ {Relaxation using γ}
8: **else**
9: $\hat{\beta}_i = \beta_i$
10: **end if**
11: $\mu_i = \hat{\beta}_i / \alpha_i$
12: {Message update subject to a schedule}
13: **for** each edge $i \to j$ **do**
14: $\alpha_{ij} = -A_{ij}^2(\alpha_i - \alpha_{ji})^{-1}$
15: $\beta_{ij} = -A_{ij}(\hat{\beta}_i - \beta_{ji})(\alpha_i - \alpha_{ji})^{-1}$
16: **end for**
17: **end for**
18: Compute: $e_r = \left(\frac{\sum(\mu_i - \mu_i^{(t-1)})^2}{\sum \mu_i^2} \right)^{\frac{1}{2}}$
19: **until** Convergence check: $e_r < \epsilon$
20: *Output:* $\overline{u} = [\mu_i] \, \forall i$

The condition for variance convergence in step-6 of Algorithm 6 is not inherently required but rather is used to reduce the memory requirement. As explained earlier, once the variances converge, we can obtain $\beta_i^{(t-1)}$ from $u_i^{(t-1)}$ in order to relax $\beta_i^{(t)}$ which saves implementation memory of up to $O(N)$.

By using an over-relaxation factor $\gamma \in [1, 2]$, our empirical results indicate that the optimal γ_{opt} that yields the lowest iteration count for a given tolerance (ϵ) for the relative norm, as defined in Section 2.5, depends strongly on the elements of the matrix A, which

makes γ_{opt} dependent on the underlying problem. Hence, finding a suitable γ may prove difficult especially since using the wrong value will cause the algorithm to fail to converge. In the following section, we propose a heuristic algorithm that iteratively and incrementally finds an approximation to γ that, in general, produces a sufficiently fast convergence by iteratively reducing the relative norm.

4.3 Dynamic Over-relaxation

The solution presented here, which circumvents determining γ_{opt} a priori, is motivated by the following empirical observations of the R-GaBP algorithm: a threshold γ_{opt} exists in the interval $[1, 2]$ that is also found to be a maximum in the same interval for given initial conditions, any further increase in γ_{opt} will cause a substantial increase in $e_r^{(t)}$. As a result, we can propose a dynamic update scheme for γ based on the progress of the R-GaBP relative norm. More precisely, we can gradually increase γ starting from an initial value by adding an increment $\Delta\gamma$ each d number of iterations as long as the e_r is improving. However, if e_r is found not to improve, we can likewise decrement γ. Algorithm 7 shows the details of an algorithm that can be used to find a rough estimate of the over-relaxation γ which results in a high relative norm decrease rate.

Algorithm 7 relies on two basic settings $\Delta\gamma$ and d. The parameter $\Delta\gamma$ is a fixed increment or decrement size which nominally can be set to 0.1, while d is the iteration interval length on which e_r can be tested in order to adjust γ. The relative norm sampling interval d should be chosen wide enough so that the fluctuations in e_r, resulting from the prior γ change, can diminish resulting in a stable value for e_r that can be sampled. In all of the cases we analyzed for ill-conditioned WDD matrices, a good value for d was found to be around 10 to 20 iterations. We refer to this algorithm as the DR-GaBP algorithm.

Algorithm 7 The Dynamically Relaxed Gaussian Belief Propagation (DR-GaBP) algorithm.

Initialize:
  $e_{best} = 1.0 \quad \gamma^{(0)} = 1.0$
  $d = 10 \qquad \Delta\gamma = 0.1$
repeat {PW-GaBP iteration: $t = 1, 2, \cdots$}
  if $t \mod d = 0$ **then**
    if $e_r^{(t)} < e_{best}$ **then**
      $\gamma^{(t)} = \gamma^{(t-1)} + \Delta\gamma$ {Increment γ}
      $e_{best} = e_r^{(t)}$
    else
      $\gamma^{(t)} = \gamma^{(t-1)} - \Delta\gamma$ {Decrement γ}
      if $\gamma^{(t)} < 1.0$ **then**
        $\gamma^{(t)} = 1.0$
      end if
    end if
  end if
until PW-GaBP terminates

If $\Delta\gamma$ is chosen sufficiently small with a sufficiently wide relative norm sampling iteration interval d, the DR-GaBP algorithm should converge. Unlike the Aitken-Steffensen or similar acceleration methods, this algorithm is computationally more stable specially for ill-conditioned matrices. Another key advantage of the DR-GaBP algorithm is that it does not require a significant increase in both computation and memory over the original PW-GaBP algorithm.

4.4 Results and Discussions

The test matrices are obtained from the classical L-shaped conductor problem in electromagnetics. As shown in Fig. 4.1, the potential in the space between the two square conductors carrying different voltages is found by solving the Laplace equation, $\nabla^2 u = 0$. However in practice, Laplace's equation is solved numerically by the FEM, which starts by dividing the interconductor space into triangular elements. Using a first order FEM, the problem typi-

cally requires the solution of a linear system of equations that is large, sparse, ill-conditioned, and WDD. In this section, we demonstrate the effectiveness of our developed algorithms, the R-GaBP and the DR-GaBP, by solving the linear system arising from this Laplace problem. We also use a selected set of other generated matrices in order to illustrate our empirical results. We used asynchronous message scheduling for all our algorithms. The relative norm e_r is recorded at each iteration. All algorithms were terminated when the normalized residual (l^2-norm) reached $R < 10^{-9}$.

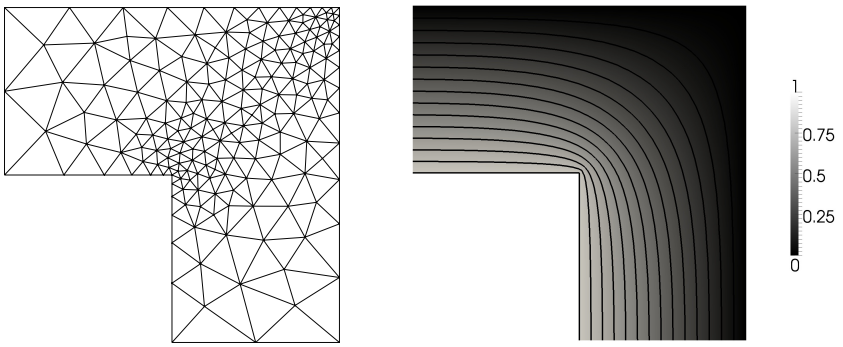


Fig. 4.1 L-shaped conductor problem of dimensions equal to 1cm. (a) Illustrated discretization using a small mesh. (b) Equipotential lines of the potential solution of the Laplace's equation, volts scale.

The plots in Fig. 4.2 demonstrate the iteration reduction due to the R-GaBP algorithm. The size of the linear system characterized by the matrix A is $N = 2700$ unknowns, with number of non-zeros $nnz = 17572$. The original PW-GaBP algorithm required 2449 iterations while the R-GaBP algorithm required as low as 389 iterations for $\gamma = 1.538$ resulting in a reduction factor of 6.2. It can be observed that the overall relative norm decreases consistently as γ increases to a certain value in the interval $[1, 2]$. It is expected that for this problem the best γ_{opt} can be empirically obtained as $\gamma_{opt} \approx 1.538$, any further increase on γ will cause e_r to increase, and consequently causing the algorithm to fail to converge.

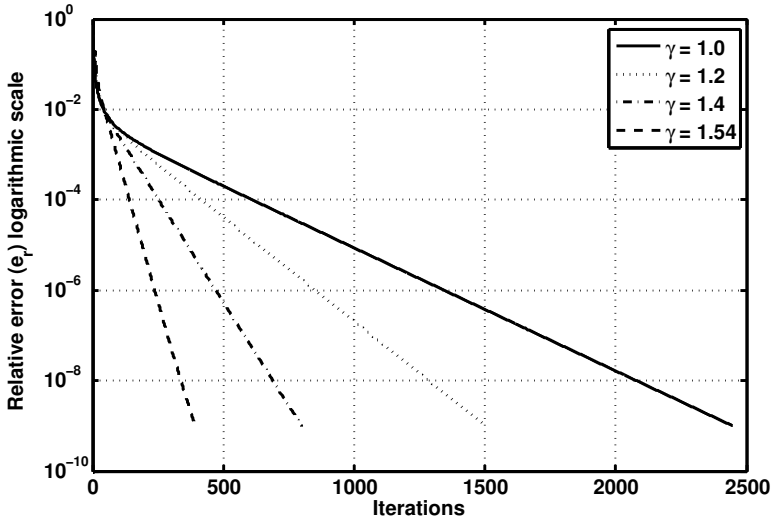


Fig. 4.2 The relative norm e_r plots for the R-GaBP algorithm with different relaxation factors γ.

The plots in Fig. 4.3 show the results of the DR-GaBP algorithm in obtaining a considerably lower PW-GaBP iteration count without prior knowledge of γ_{opt}. The parameter $\Delta\gamma$ is varied from the fine value of 10^{-2} to the coarser value of 2×10^{-1}. A relative norm sampling interval of $d = 10$ was used for all plots. It is worth noting here that the best iteration reduction was found for $\Delta\gamma = 2 \times 10^{-1}$ resulting in 337 iterations. This is lower than the previously reported 389 iterations by R-GaBP assuming prior knowledge of γ_{opt}. This reduction may be attributed to the dynamics of the algorithm in alternating between two values of γ which are (1.3 and 1.6).

The case of DR-GaBP with $\Delta\gamma = 10^{-2}$ resulted in 1381 iterations. While this performance is still considerably better than the original PW-GaBP, the larger comparative iterations may be attributed to the fact that γ fluctuates between two values which do not necessarily produce the lowest overall relative norm. However, the algorithm converged for all considered values of $\Delta\gamma$. In general, the smaller the values of $\Delta\gamma$ are, the wider the applicability of the algorithm at the expense of a higher iteration count. We found that

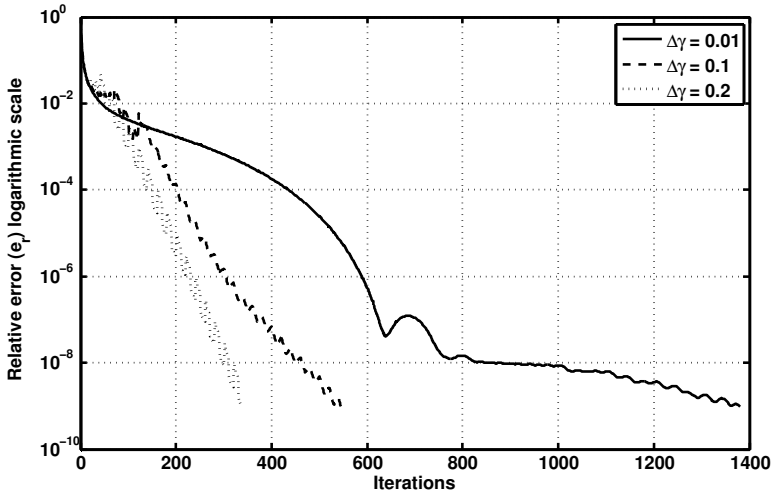


Fig. 4.3 The relative norm e_r plots for the DR-GaBP algorithm with different relaxation increments $\Delta\gamma$.

the nominal $\Delta\gamma = 10^{-1}$ is suitable for most matrices that are ill-conditioned and weakly diagonally dominant.

We illustrate in Table 4.1 a selected set of test matrices. One matrix was obtained from the Matrix Market website repository [68]. The other two matrices were generated randomly using Matlab and set to be sparse and WDD. The generated matrices all have negative off-diagonals which results in ill-conditioned matrices. For all algorithms we used $d = 10$. The variances for all matrices converged in less than 10 iterations. The density % of the matrix is measured by $nnz/N^2 \times 100$.

Table 4.1 demonstrates that reductions in iterations were produced by the R-GaBP algorithm in all test cases. The second matrix shows a reduction factor of 12.7 for $\gamma = 1.8$. Table 4.2 shows the results of DR-GaBP which also produced significant iteration reductions versus the original PW-GaBP. The performance of DR-GaBP algorithm can depend on the choice of $\Delta\gamma$, however the algorithm converged in all tested cases. It was observed that the nominal value of $\Delta\gamma = 0.1$ resulted in best reductions on average.

Table 4.1 Results for the R-GaBP algorithm on selected test matrices.

Matrix	N	Density (%)	PW-GaBP	R-GaB		Red. Factor
				Itrs	γ	
gr_30_30 [68]	900	0.956	859	115	1.59	7.5
Sp Rand 1	5000	0.4	5571	437	1.8	12.7
Sp Rand 2	3000	0.4	3028	241	1.67	12.6

Table 4.2 Results for the DR-GaBP algorithm on selected test matrices.

Matrix	DR-GaBP			Red. Factor
	$\Delta\gamma = 0.01$	$\Delta\gamma = 0.1$	$\Delta\gamma = 0.2$	$\Delta\gamma = 0.1$
gr_30_30 [68]	456	783	647	1.1
Sp Rand 1	1873	872	1492	6.4
Sp Rand 2	732	759	963	4.0

4.5 Conclusion

We have presented the R-GaBP algorithm which implements new relaxation schemes for the PW-GaBP algorithm to considerably accelerate its convergence for ill-conditioned WDD inverse covariance matrices. We have demonstrated empirical reductions in iterations of up to 12.7 times. We have also introduced the DR-GaBP algorithm that avoids the complexity of setting a prior over-relaxation factor. The DR-GaBP algorithm demonstrates performance comparable with the R-GaBP algorithm with a prior knowledge of an optimal relaxation factor. The new algorithms do not require any increase of the computational complexity or the memory requirements of the original PW-GaBP algorithm hence facilitating efficient implementations on parallel architectures.

CHAPTER 5

Parallel Solution of the Finite Element Method Using Gaussian Belief Propagation

The proliferation of parallel architectures such as manycore CPUs and GPUs drives the need to redesign conventional algorithms with parallelism in mind. However, the parallel performance of conventional FEM implementations can be limited by the inherent coupling of the underlying sparse data-structures. In this chapter, we look into Belief Propagation (BP) inference algorithms to address these challenges. We introduce a probabilistic Graphical Model (GM) of the FEM based on Factor Graphs (FGs) by recasting the global deterministic FEM as a localized variational inference problem. The resulting algorithm eliminates the need to assemble any global sparse data-structure or perform any global sparse algebraic operation.

5.1 Introduction

We have presented in Section 2.4.2 a background of the Gaussian Belief Propagation on Pairwise Graphical Models (PW-GaBP) algorithm and showed that it is a recursive message passing algorithm that offers distributed computations [31,33,54]. PW-GaBP demonstrated

better empirical results than conventional iterative solvers such as Jacobbi, Gauss-Seidel (GS), Successive Over-Relaxation (SOR) and Diagonally-Preconditioned Conjugate Gradient (D-PCG) for SDD matrices [54,55,57]. We have also shown in Chapter 3 that the PW-GaBP algorithm can potentially parallelize FEM applications when using scheduling schemes that facilitates its parallel execution [57]. While new schemes were introduced in Chapter 4 to accelerate the PW-GaBP for FEM matrices using the dynamic relaxation DR-GaBP algorithm, the overall performance of the DR-GaBP algorithm was still short of popular solvers such as the IC-PGC solver. However, the PW-GaBP solver, and similarly all other conventional solvers, still requires the assembly of a large sparse matrix data-structure. Many applications that require repeated reassembly of the global matrix, such as adaptive multigrid and non-linear applications, can suffer from the long setup time of the sparse matrix.

We believe that the assembly of the sparse matrix and the algebraic operations on such data-structures are the sources of bottlenecks that are hindering efficient parallel implementations for the FEM computation on HPC platforms. Novel and inherently parallel algorithms, therefore, need to be developed from a completely different perspective in order to address this issue. The classical variational formulation of the FEM provides useful physical insight for the underlying BVP, which leads us to exploring the use of a different breed of algorithms used in variational inference such as BP [31]. In this work, we present a new BP-based algorithm specifically derived for the FEM that is referred to as the Finite Element Gaussian Belief Propagation (FGaBP). We first develop a variational inference formulation for the FEM in order to facilitate its interpretation as a GM problem. We then introduce the FG model for the variational inference FEM referred to as the Finite Element Based Factor Graph (FEM-FG) model. Next, we use the new FEM-FG model to develop the FGaBP algorithm. We show that the FGaBP algorithm is applicable for arbitrary element geometry and interpolation order, which is needed to address a wide variety of FEM applications. The

FGaBP algorithm provides better parallel computations than conventional algebraic methods by avoiding the assembly of the global sparse matrix, and by solving the FEM in parallel element-by-element without the need to perform any global sparse matrix operations. The new algorithm provides flexible memory bandwidth utilization due to its use of distributed message-based computations, referred to as message scheduling, which allows us to adapt its implementation on hardware using various memory architectures without impacting the algorithm's computational stability. Such a critical advantage is usually lost when adapting reformulation techniques of the CG solver such as the Communication Avoiding (CA) [25, p. 34] which is based on the s-step [96] method.

The chapter is organized as follows. In Section 5.2 we present the formulation of the FEM as a variational inference problem. In Section 5.3 we introduce the FEM-FG model, and in Section 5.4 we derive BP update rules on the FEM-FG and present the FGaBP algorithm. In Section 5.5 we present reformulations of the FGaBP algorithm that considerably reduce its computational complexity. In Section 5.6 we present the element merge solution that enhances the memory bandwidth utilization of the FGaBP algorithm. In Section 5.7 we detail implementation analysis for the FGaBP algorithm. Finally, in Section 5.8 we conclude with our results and discussions illustrating the FGaBP execution on FEM problems.

5.2 The FEM as a Variational Inference Problem

The work in [32, 49], introduced by Yedidia et al., shows that the BP solution on FG models corresponds to the Bethe approximation of the free energy [97, 98] associated with the underlying inference model. Using that insight, we will derive the FGaBP algorithm to minimize the variational energy of the FEM problem. To that end, we first model the FEM in terms of a probabilistic GM referred to as the FEM Factor Graph (FEM-FG). We start by

reformulating the FEM as a variational inference problem by modifying the FEM functional (2.18) as follows:

$$\mathcal{P}(U) = \frac{1}{Z} \exp\left[-\sum_{s\in\mathcal{S}} \mathcal{F}_s(U_s)\right] \tag{5.1}$$

$$= \frac{1}{Z} \prod_{s\in\mathcal{S}} \Psi_s(U_s) \tag{5.2}$$

where Z is a normalizing constant and $\Psi_s(U_s)$ are local factor functions of the local finite element variables U_s defined as:

$$\Psi_s(U_s) = \exp\left[-\frac{1}{2}U_s^T M_s U_s + B_s^T U_s\right]. \tag{5.3}$$

It is worth noting that Ψ_s as defined in (5.3) takes a multivariate Gaussian form albeit unnormalized when the element characteristic matrix M_s is Symmetric Positive Definite (SPD). Appendix A presents more details about Gaussian distributions and their properties.

It can be shown that $\mathcal{P}$ as in (5.1) represents a Gaussian multivariate probability distribution when $\mathcal{F}$ is convex quadratic [36]. Hence, the optimality condition of $\mathcal{F}$ can be restated as:

$$\arg\min_U \mathcal{F} = \arg\max_U \mathcal{P}. \tag{5.4}$$

Here, we assume that the variables vector U will represent joint Gaussian random variables under the distribution $\mathcal{P}$. Since $\mathcal{P}$ is maximized when $U = \mu$, where μ is the mean vector of $\mathcal{P}$, the FEM minimization problem of the functional $\mathcal{F}$ is transformed into a computational inference problem of finding the marginal means of the random variables U under the distribution $\mathcal{P}$. Hence, BP inference algorithms can be employed to compute the marginal means

of the random variables U.

Now, we turn our attention to the normalizing constant Z in (5.1). Wainright and Jordan have presented a framework for variational inference on exponential distributions in [36]. We will follow a similar approach to analyze the distribution $\mathcal{P}$. In our case, Z ensures that $\mathcal{P}$ is a valid distribution such that:

$$\int_U \mathcal{P}\, \mathrm{d}U = 1 \tag{5.5}$$

or alternatively:

$$Z(M_s, B_s) = \int_U \exp\left(-\mathcal{F}\right)\, \mathrm{d}U < +\infty, \quad \forall s \tag{5.6}$$

that is Z is a function of M_s and B_s such that $Z(M_s, B_s)$ is finite. Since $\mathcal{F}$ is quadratic, and by convex analysis, finiteness of the integral can be met by ensuring that the element matrices M_s are positive definite. Observing that $\mathcal{F}$ can also be represented in an assembled form, then the finiteness requirement can alternatively be met by ensuring that the assembled form of the FEM matrix is positive definite. Note that the symmetry condition on M_s can be imposed only to simplify the BP update rules derivations by assuming Gaussian distributions for the underlying random variables, as will be shown in the following sections.

5.3 The FEM Factor Graph Model

Given the distribution $\mathcal{P}(U)$, one can define a GM to perform computational inference in order to infer the marginal distributions of the random variables in U. A widely used class of GMs is the FG [35], which is a bipartite GM that directly represents the factorization of $\mathcal{P}$. We refer to the FG model derived from the distribution $\mathcal{P}$ as the FEM-FG model. The FEM-FG, as shown in Fig. 5.1(c), includes two types of nodes, a random variable node (u_i) representing each node in the unknown vector U, and a factor node representing the local factors

Ψ_s. An edge is inserted between a variable node u_i and a factor node Ψ_s if u_i is an argument of the local factor Ψ_s. For example, the two element FEM shown in Fig. 5.1(a) can be represented by the probability functional $\mathcal{P}(U) = \Psi_a(u_1, u_2, u_3, u_4, u_5, u_6)\Psi(u_4, u_5, u_6, u_7, u_8, u_9)$. Likewise, the FEM-FG shown in Fig. 5.1(c) contains two factor nodes labeled FN_a and FN_b each for Ψ_a and Ψ_b correspondingly. The node FN_a has edges to the variable nodes set $\{u_1, u_2, u_3, u_4, u_5, u_6\}$ that are arguments of Ψ_a and, similarly, the node FN_b is connected to variable nodes that are arguments of Ψ_b.

The Pairwise Graphical Model (PWGM), corresponding to the same mesh, is also provided in Fig. 5.1(b) for comparison. As mentioned in Section 2.4.2, the PWGM model is used to derive the PW-GaBP algorithm as a solver for linear systems of equations [54]. From that perspective, it may seem that the PW-GaBP algorithm has a greater degree of generality; however, the PW-GaBP has major shortcomings when used for FEM problems. Mainly, the PWGM does not benefit from the underlying FEM problem structure; as a result, the number of communication links in the PWGM is much higher in comparison to the FEM-FG model for certain meshes. While it may be easy to deduce the structure of the FEM-FG model from the underlying FEM mesh, it is important to note that the number of vertices or edges in a mesh does not necessarily correspond to the number of vertices or edges in a FEM-FG model as shown in Fig. 5.1(b). We can illustrate the link reduction advantage of the FEM-FG model by counting the links in the same example diagrams. From the figures, the FEM-FG requires 12 links while the PWGM requires 27 links for a mesh of two second order triangles. Similarly, link reductions can also be illustrated considering first order quadrilateral and hexahedral meshes and second order tetrahedral meshes. In fact, the reduction in the number of links for the FEM-FG model progressively increases with the order of the FEM element. More detailed analysis on this will be provided later in Section 5.7, where we discuss the implementation details. First order triangular and tetrahedral meshes

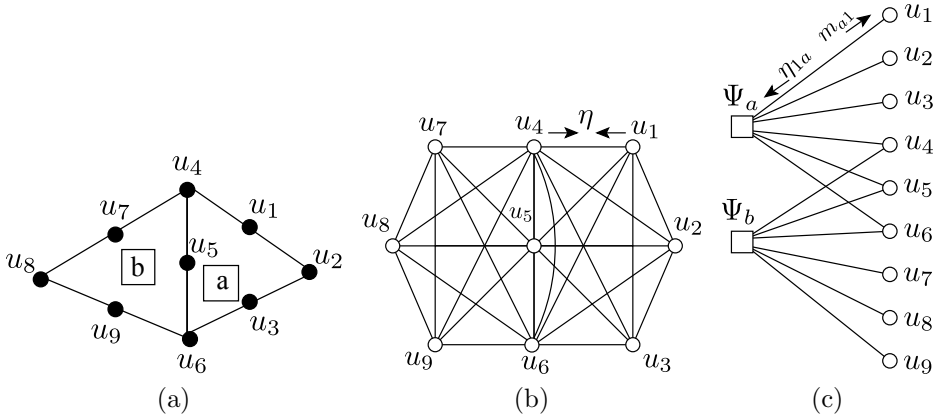


Fig. 5.1 (a) Sample FEM mesh of two second order triangles. Two GM representations are shown in (b) and (c); the messages η in a GM are communicated recursively either sequentially or in parallel. (b) The conventional PWGMs. (c) The new FEM-FG model with reduced number of communication links and improved parallelism.

are the only case where the FEM-FG model will contain more links than the PWGM. To remedy this situation, we provide a solution that lowers the number of links in the FEM-FG model for first order triangular and tetrahedral meshes by merging elements that share edges or faces. The element merging solution is detailed in Section 5.6.

An important remark to make on the FEM distribution $\mathcal{P}$ is that it is conveniently represented in a factorized form due to its direct derivation from the FEM functional (2.18). This factorization reflects the structure of the underlying FEM mesh that is used to derive the FEM-FG model. In fact, the local factor functions Ψ_s are functions over the cliques in the PWGM model representing the sparse matrix we would have obtained, if we chose to assemble the FEM linear system of equations. Nonetheless, the term clique is used here for technical convenience since, due to the factorized nature of the exponential distribution $\mathcal{P}$, higher orders of cliques can be formed in the FEM-FG model by introducing zero edges in order to form other supersets of cliques. This can be used, for instance, to improve the memory bandwidth utilization of the algorithm for certain triangular or tetrahedral meshes

as later discussed in Section 5.6. This indicates that the FEM-FG representation is more adept in exploiting the FEM problem structure than the PWGM representation used for the PW-GaBP solver. In addition to this flexibility, the FGaBP algorithm, based on the FEM-FG model, demonstrates improved convergence versus the PW-GaBP, as shown later in Section 5.8. We believe that this improved convergence is a direct consequence of the FEM-FG model better exploiting the underlying FEM problem's structure. In addition, inference on FEM-FG can perform more dense and localized computations (correlating more of the local variables states) resulting in less communication as opposed to inference on PWGMs.

Lastly and most importantly, the PW-GaBP, as well as other conventional iterative solvers still require the assembly of the large sparse matrix A while the FEM-FG model eliminates this costly step. Another key advantage of using the FEM-FG model as opposed to the PWGM is that the FEM-FG configuration draws naturally from the FEM mesh. To illustrate this advantage, we consider the hypothetical scenario where one would attempt to generate an FEM-FG (or maximal clique FG configuration) from a PWGM. Clearly, considerable overhead processing is needed in order to use specialized algorithms, such in [99–103], to identify the maximal cliques in the PWGM. In addition to identifying cliques, one would still need to perform the proper edge coefficient splitting on edges shared between cliques while maintaining the numerical integrity of the BP inference algorithm. A task that may prove difficult for ill-conditioned systems.

5.4 The FGaBP Algorithm

To derive the update rules of the FGaBP algorithm, two levels of specialization need to be applied to the general BP rules introduced previously in Section 2.4. First, we setup

the BP inference algorithm on the new FEM-FG model of the introduced FEM variational inference functional (5.1). Then, recognizing that the FEM inference functional (5.1) takes a Gaussian form; as a result, all the update messages will take a Gaussian form. Hence, the multidimensional integral in (2.22) can be solved in a closed form which reduces the update messages to propagating only the Gaussian parameters.

5.4.1 The FGaBP Update Rules

In this section we detail all the BP update rules formulation for the FGaBP algorithm.

Gaussian Parameterization

For our development of the FGaBP, we use a specific Gaussian distribution parametrization referred as the canonical parameterization, or the information form. More details on this parameterization and its relationship to the normal Gaussian distribution form is provided in Appendix A. The univariate canonical Gaussian form is defined as:

$$G(\alpha, \beta) \propto \exp\left[\frac{-1}{2}\alpha u^2 + \beta u\right] \tag{5.7}$$

where α is the first canonical parameter that is equivalent to the reciprocal of the Gaussian variance; and β is the second parameter. For multivariate Gaussian distributions, we use the following parameterized form:

$$\mathcal{G}(W, K) \propto \exp\left[\frac{-1}{2}U^T W U + K^T U\right] \tag{5.8}$$

where U is a vector of random Gaussian unknowns of arbitrary dimension, e.g. n; W is the first canonical parameter which is an n-by-n matrix assumed to be SPD; K is the

second parameter which is a vector of the same dimension n. We use proportionality in the Gaussian distributions because we do not require any normalization. In general, the normalization constant is used to generate valid probability distributions such that the area underneath the distribution is equal to one. Since in our setting we are only interested in the parameters of the distribution, α and β, rather than the distribution itself, which are sufficient to identify all the messages' distributions in the FGaBP algorithm without the need of any normalization.

We start by making the assumption that the random variables $u_i \in U$ take Gaussian distributions as follows:

$$u_i \sim G(\alpha_i, \beta_i). \tag{5.9}$$

As a result, the BP messages m_{ai} and η_{ia}, as in (2.22) and (2.23), take Gaussian forms parametrized by α and β. This follows from the properties of multiplication and marginalization of Gaussian distributions as detailed in Appendix A. Therefore substituting the FEM local element function Ψ_a (5.3) into (2.22), the BP update rules in (2.22), (2.23), and (2.24) can be reduced to propagating parameters α and β between factor and variable nodes in the FEM-FG graph.

Variable Node Updates

For each Variable Node i (VN_i) process, compute the outgoing messages $(\alpha_{ia}, \beta_{ia})$ to each Factor Node a (FN_a), such that $a \in \mathcal{N}(i)$, as follows:

$$\alpha_{ia}^{(t)} = \sum_{k \in \mathcal{N}(i) \setminus a} \alpha_{ki}^{(t_\star)} \tag{5.10}$$

$$\beta_{ia}^{(t)} = \sum_{k \in \mathcal{N}(i) \setminus a} \beta_{ki}^{(t_\star)}. \tag{5.11}$$

where t and $t_\star$ are iteration counts such that $t_\star \leq t$. However, this computational complexity can be reduced considerably if we compute the VN messages in two stages. In the first stage, combine all incoming messages $(\alpha_{ai}, \beta_{ai})$ from each FN_a, $a \in \mathcal{N}(i)$, into the nodal parameters (α_i, β_i) as follows:

$$\alpha_i^{(t_\star)} = \sum_{k \in \mathcal{N}(i)} \alpha_{ki}^{(t_\star)} \tag{5.12}$$

$$\beta_i^{(t_\star)} = \sum_{k \in \mathcal{N}(i)} \beta_{ki}^{(t_\star)}. \tag{5.13}$$

Then in the second stage, compute the outgoing messages $(\alpha_{ia}, \beta_{ia})$ to each FN_a as follows:

$$\alpha_{ia}^{(t)} = \alpha_i^{(t_\star)} - \alpha_{ai}^{(t_\star)} \tag{5.14}$$

$$\beta_{ia}^{(t)} = \beta_i^{(t_\star)} - \beta_{ai}^{(t_\star)}. \tag{5.15}$$

It is important to maintain the BP message consistency when using the second approach. The BP message consistency is maintained here when the same $t_\star$ value on each individual link is used for all equations in (5.12), (5.14), (5.13), and (5.15). Using this approach, the complexity of a VN message computation is reduced from $O\big(2m(m-1)\big)$ to $O\big(4m\big)$ for $m > 3$, where m is the number of the VN edges.

Factor Node Updates

For each FN_a process:

1. Receive messages $(\alpha_{ia}^{(t_\star)}, \beta_{ia}^{(t_\star)})$, where $i \in \mathcal{N}(a)$.

2. Define $\mathcal{A}^{(t_\star)}$ and $\mathcal{B}^{(t_\star)}$ such that $\mathcal{A}^{(t_\star)}$ is a diagonal matrix of incoming $\alpha_{ia}^{(t_\star)}$ parameters,

and $\mathcal{B}^{(t_\star)}$ is a vector of incoming $\beta_{ia}^{(t_\star)}$ parameters as follows:

$$\mathcal{A}_a^{(t_\star)} = \begin{bmatrix} \alpha_{i_1 a}^{(t_\star)} & 0 & \cdots & 0 \\ 0 & \alpha_{i_2 a}^{(t_\star)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_{i_n a}^{(t_\star)} \end{bmatrix}, \quad \mathcal{B}_a^{(t_\star)} = \begin{bmatrix} \beta_{i_1 a}^{(t_\star)} \\ \beta_{i_2 a}^{(t_\star)} \\ \vdots \\ \beta_{i_n a}^{(t_\star)} \end{bmatrix}, \tag{5.16}$$

where $\{i_1, i_2, \ldots, i_n\} = \mathcal{N}(a)$. Then, compute W and K as follows:

$$W^{(t_\star)} = M + \mathcal{A}^{(t_\star)} \tag{5.17}$$

$$K^{(t_\star)} = B + \mathcal{B}^{(t_\star)} \tag{5.18}$$

where M and B are the element a characteristic matrix and source vector as defined in (5.3) with $s = a$.

3. Partition $W^{(t_\star)}$ and $K^{(t_\star)}$ as follows:

$$W^{(t_\star)} = \begin{bmatrix} W_{d,d}^{(t_\star)} & V^T \\ V & \bar{W}^{(t_\star)} \end{bmatrix}, \qquad K^{(t_\star)} = \begin{bmatrix} K_d^{(t_\star)} \\ \bar{K}^{(t_\star)} \end{bmatrix} \tag{5.19}$$

where $d = \mathcal{L}(i)$ is the local index corresponding to the global variable node i.

4. Compute and send FN_a messages $(\alpha_{ai}^{(t)}, \beta_{ai}^{(t)})$ to each VN_i, $i \in \mathcal{N}(a)$, as follows:

$$\alpha_{ai}^{(t)} = M_{d,d} - V^T (\bar{W}^{(t_\star)})^{-1} V \tag{5.20}$$

$$\beta_{ai}^{(t)} = B_d - (\bar{K}^{(t_\star)})^T (\bar{W}^{(t_\star)})^{-1} V. \tag{5.21}$$

Note that the incoming messages $(\alpha_{ia}, \beta_{ia})$ have been subtracted from (5.20) and (5.21). Also, it is worth mentioning that the arrangement of $\alpha_{i_\star a}$ and $\beta_{i_\star a}$ into $\mathcal{A}_a$ and $\mathcal{B}_a$ correspondingly is not arbitrary. This arrangement is based on the association between the local indices and the global ones which is locally maintained in each FN. For purposes of formulation, we represent this mapping as $d = \mathcal{L}(i)$ as shown in (5.19).

Message Initialization

A priori, all the FNs messages are initialized as $\alpha = 1$ and $\beta = 0$. In fact, messages can be initialized to any arbitrary value as long as $\alpha > 0$. Using $\alpha = 1$ or greater improves the robustness of the algorithm in the first few iterations, since such a value for α will strengthen the diagonal dominance of the matrix W_a which improves the numerical properties of the inversion procedure.

Nodal Means

Once the messages converge, the nodal means μ_i, or solutions, can be obtained by:

$$\bar{u}_i^{(t)} = \mu_i^{(t)} = \frac{\beta_i^{(t)}}{\alpha_i^{(t)}}. \tag{5.22}$$

The FGaBP update rules, or messages, can be seen as messages communicating parameters of distributions, referred to as beliefs, containing information on the most probable states, or FEM solutions, of target variable nodes as seen from the perspective of each connected factor node, or FEM element. Note that the factor nodes can also represent composite FEM elements as shown in the element merge solution presented in Section 5.6. Another important property that can be observed from the FGaBP update rules, the update rules are based on distributed local computations performed using matrices of order $(n - 1)$ or

less; while computations of each FN is independent of other FNs in a given iteration. In other words, there is no need to build any large global system of equations nor there is any requirement to perform SMVM operations as required in certain iterative solvers such as the PCG. This is expected to eliminate the parallel scalability bottleneck present in conventional FEM solvers due to their inherent sequential nature. In addition, the FGaBP update rules are applicable to any FEM with arbitrary element geometrical shape or interpolation order, hence introducing great flexibility for FGaBP implementation.

5.4.2 Boundary Conditions

The treatment of essential boundary conditions can be performed at the setup stage of the FGaBP algorithm in an element-by-element parallel manner. The boundary conditions can be incorporated directly into the elements that are located on the boundary by simply modifying M_s and B_s in (5.3). When incorporating the Dirichlet boundary conditions on boundary elements, the dimensionality of the boundary elements is reduced by eliminating the fixed nodes. Hence, the FGaBP algorithm communicates informative messages only between variable nodes. To illustrate how to incorporate the Dirichlet boundary conditions, we partition the local variable vector U_s into $U_s = \left[U_{\{v\}}, U_{\{c\}}\right]^T$ where v is the set of interior variable nodes indices and c is the set of boundary Dirichlet nodes indices. The boundary nodes are then assigned the boundary values $U_c = \mu_c$. Correspondingly, we partition M_s and B_s as follows:

$$M_s = \begin{bmatrix} M_{vv} & M_{vc} \\ M_{cv} & M_{cc} \end{bmatrix}, \qquad B_s = \begin{bmatrix} B_v \\ B_c \end{bmatrix} \tag{5.23}$$

Incorporating the partitions into (5.3), we obtain:

$$\Psi_s(U_v, U_c) = \exp\Big[-\frac{1}{2}\big[U_v^T M_{vv} U_v + U_v^T M_{vc} U_c + U_c^T M_{cv} U_v + U_c^T M_{cc} U_c\big] + B_v^T U_v + B_c^T U_c\Big]. \tag{5.24}$$

Given a normalized Ψ_s being a valid Gaussian distribution, we intend to compute the conditional distribution given the boundary conditions $\Psi_s(U_v \mid U_c = \mu_c)$. From probability theory we obtain:

$$\Psi_s(U_v \mid U_c = \mu_c) = z\frac{\Psi_s(U_v, U_c = \mu_c)}{\int_{U_v} \Psi_s(U_v, U_c = \mu_c)\,\mathrm{d}U_v} \tag{5.25}$$

where z is an arbitrary normalizing constant. Clearly, the integral in the denominator is a constant, hence (5.25) simplifies to:

$$\Psi_s(U_v \mid U_c = \mu_c) \propto \Psi_s(U_v, U_c = \mu_c). \tag{5.26}$$

Which states that the element's function for a boundary element with fixed boundary nodes can be obtained by assigning the boundary variable nodes their corresponding boundary values.

After fixing the boundary nodes and eliminating the constant terms, (5.24) simplifies to:

$$\Psi_s(U_v, U_c = \mu_c) \propto \exp\Big[-\frac{1}{2}U_v^T M_{vv} U_v + \big[B_v - \frac{1}{2}M_{vc}\mu_c - \frac{1}{2}M_{cv}^T\mu_c\big]^T U_v\Big]. \tag{5.27}$$

Which is similar to locally reassigning M_s and B_s as follows:

$$M_s = M_{vv} \tag{5.28}$$

$$B_s = B_v - \frac{1}{2}M_{vc}\mu_c - \frac{1}{2}M_{cv}^T\mu_c. \tag{5.29}$$

Considering the symmetry condition ($M_{vc} = M_{cv}$), (5.29) reduces to:

$$B_s = B_v - M_{vc}\mu_c. \tag{5.30}$$

Hence, the FGaBP update rules can be executed normally on boundary elements after eliminating boundary nodes using (5.28) and (5.30).

5.4.3 Message Scheduling

Message communication in FGaBP can be performed subject to a particular schedule which can be sequential, in parallel, or in any order. To explain these types of message scheduling, we first need to define what an FGaBP iteration is. An FGaBP iteration is defined by the process of traversing and updating all the FNs exactly once. This definition is only intended to help quantify the computational complexity of the FGaBP algorithm; however, in actual implementations the iteration boundaries and the message synchronization can be as flexible as needed. In fact, one of the key empirical observations of the FGaBP algorithm is the flexibility in message communication, which enables implementations that efficiently trade off computation with communication on various parallel architectures without compromising the numerical stability of the algorithm. However, message scheduling can considerably affect the number of iterations required for the algorithm to converge; therefore, a good message schedule exposes parallelism by exploiting the underlying connectivity structure of the problem [55] with small impact on the iteration count, as demonstrated in Chapter 3.

There are two basic scheduling schemes for general BP messages, sequential (asynchronous) and parallel (synchronous). In sequential scheduling, all the FNs are sequentially traversed according to a particular order while their messages are communicated and synchronized one FN at a time. Therefore, each FN computes its messages based on the most

recent nodal messages available within the iteration, that is $t_\star = t$. This message schedule results in the lowest number of FGaBP iterations; however, it does not offer much parallelism. In parallel message scheduling, all the FNs are processed in parallel while using α_i and β_i values computed at a previous iteration, that is $t_\star = t - 1$. An additional loop is needed to traverse all the VNs in parallel to compute new α_i and β_i values from updated α_{ai} and β_{ai}. Such scheduling offers a high degree of parallelism; however, it requires considerably higher number of iterations due to the slower propagation of information. To address shared memory architectures, we propose an element-based coloring schedule that exploits parallelism inherent in the FEM-FG graphical model while not significantly increasing the number of FGaBP iterations.

Color-Parallel Message Scheduling

To implement a Color-Parallel Message Schedule (CPS), an element coloring algorithm needs to be used. The mesh elements are colored in such a way that no two adjacent elements have the same color symbol. Elements are deemed adjacent if they share at least a node. A simple mesh coloring diagram is illustrated in Fig. 5.2 using two types of meshes, a quadrilateral mesh and a triangular mesh. FN messages in each color group are computed and communicated in parallel.

To facilitate a CPS scheme, The FGaBP message updates are modified as follows:

$$\alpha_i^{(t)} = \alpha_i^{(t_\star)} + (\alpha_{ai}^{(t)} - \alpha_{ai}^{(t_\star)}) \tag{5.31}$$

$$\beta_i^{(t)} = \beta_i^{(t_\star)} + (\beta_{ai}^{(t)} - \beta_{ai}^{(t_\star)}). \tag{5.32}$$

In other words, a running sum of α_i and β_i parameters are kept in each VN, while differences on the FN edge messages are only communicated by FN processes. The running sums α_i and

β_i must be initialized to zero before starting the FGaBP iterations. In this scheme, there is no need for an additional loop to traverse and synchronize the VNs.

The FN processes are synchronized before starting each color group. This scheme is particularly efficient for multi-threaded implementations on multicore CPUs or manycore GPUs, since thread-safety is automatically guaranteed by the CPS scheme. A typical coloring algorithm would aim to produce the least number of colors; since, this will reduce the number of thread synchronizations needed at the end of each color group. However, since FEM meshes contain a very large number of elements, producing a reasonable number of colors using a low complexity algorithm can be sufficient as long as each color contains enough elements for a nearly balanced multithreaded execution.

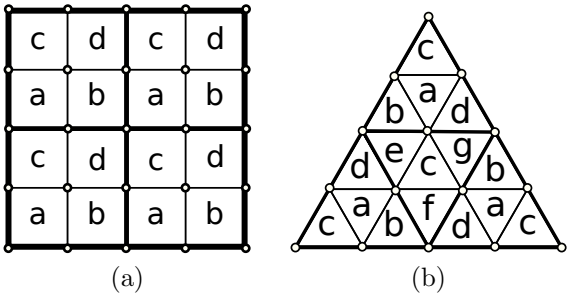


Fig. 5.2 Mesh element coloring showing two types of meshes. (a) Structured quadrilateral mesh containing a total of four colors. (b) Triangular mesh containing a total of six colors.

Partition-Parallel Message Scheduling

Since FEM elements exhibit geometrical locality in meshes, factor nodes adjacent to each other could be scheduled on the same processing element in order to compute using sequential updates; while communications between factor nodes in different processing elements can be scheduled using parallel updates. This scheduling scheme is referred to as the Partition-Parallel Message Schedule (PPS) and is illustrated in Fig. 5.3. The nodes using the parallel

schedule lie on the boundary of partitions. As a result, the number of parallel scheduled nodes is considerably less than the number of sequentially scheduled nodes that lie in the interior of the partition. A similar scheduling scheme was previously introduced in Chapter 3 and was referred to as the HUS scheme [55]; however, the key distinction here is that the CPS scheme can be used within each partition.

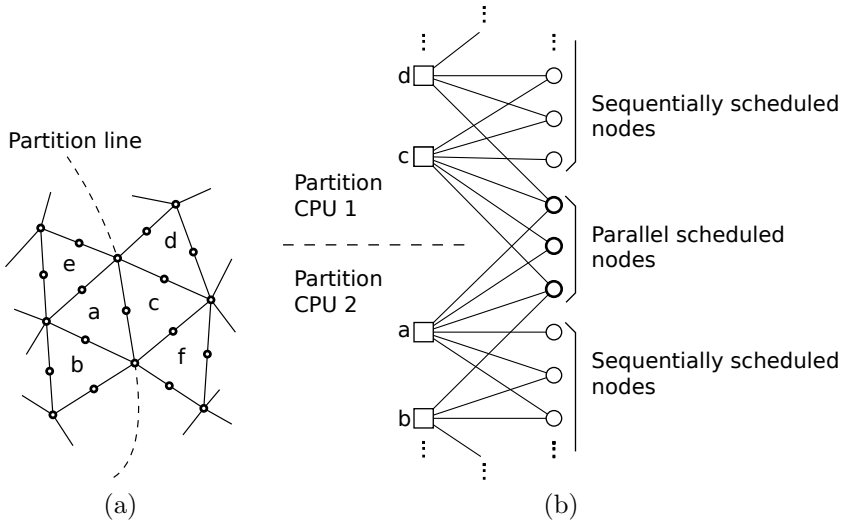


Fig. 5.3 The partition-parallel Message schedule. (a) Hypothetical mesh. (b) Partitioned FEM-FG.

5.4.4 The FGaBP Algorithm Listing

A high level listing of the FGaBP algorithm is shown in Algorithm 8. The steps 3 and 4 are considered the setup phase of the FGaBP algorithm and can be performed completely in parallel. The loop in line 7 reflects parallelism using partitions across CPUs in a cluster, which typically is programmed using MPI. The loop in line 9 reflects parallelism using coloring across CPUs in a shared memory machine, which is programmed using multithreading. CPS implementations on shared memory machines can be performed by simply creating only one partition. On the other hand, partition-parallel implementations can be performed

by considering that each factor has its unique color; hence, the loop in line 9 is executed sequentially within a partition. The FGaBP algorithm, which encompasses a complete FEM execution process, performs its computation completely in parallel without assembling or operating on any large sparse matrices.

Algorithm 8 The FGaBP algorithm.

1: Obtain FEM mesh
2: Perform partition-color
3: Generate element matrices M_s and source vectors B_s
4: Incorporate boundary conditions
5: *Initialize:* $\alpha_{ij} = 1,\ \beta_{ij} = 0,\ \forall i, j$
6: **repeat** {FGaBP iteration: $t = 1, 2, \cdots$}
7: **loop** {Assign partitions to CPUs in cluster}
8: **loop** {For each color}
9: **loop** {Schedule factor nodes to all available CPU cores or threads}
10: Compute messages according to (5.14), (5.15), (5.20), and (5.21)
11: **end loop**
12: Synchronize threads
13: **end loop**
14: Synchronize parallel message buffers
15: **end loop**
16: **until** Convergence check on messages
17: *Output:* $\mu_i = \frac{\beta_i}{\alpha_i}$

5.5 Lower Complexity FGaBP

As can be seen from the FGaBP update rules in (5.14), (5.15), (5.20), and (5.21), the FGaBP computational complexity per iteration is dominated by the FN processes. Because of the matrix inversion required by each FN, the total FGaBP complexity per iteration, when using e.g. the Cholesky algorithm, is $O\big(N_f n(n-1)^3/3\big) \approx O\big(N_f n^4/3\big)$ where N_f is the number of FNs in the FEM-FG model. For FEM problems, N_f is typically proportional to N_v, where N_v is the number of VNs. In addition, we always have $n \ll N_f$, e.g. for first order triangular

meshes $n = 3$ and first order tetrahedral meshes $n = 4$. More importantly, n is independent of N_f which implies that the FGaBP algorithm can offer high parallel scalability for a good choice of message scheduling. However, due to the high FN computational complexity, the local computational cost may dominate as n increases when supporting higher degree FEM basis functions, for example when $n \geq 5$. In the following, we present reformulations of the FGaBP algorithm that considerably reduces the FN complexity to $O\big(n^2\big)$.

We can reformulate the FN update rules using the partitioned matrix inversion identity as stated in the following:

$$Z = \begin{bmatrix} P & Q \\ R & S \end{bmatrix}, \qquad Z^{-1} = \begin{bmatrix} \tilde{P} & \tilde{Q} \\ \tilde{R} & \tilde{S} \end{bmatrix} \tag{5.33}$$

where:

$$\begin{aligned} \tilde{P} &= V \\ \tilde{Q} &= -VQS^{-1} \\ \tilde{R} &= -S^{-1}RV \\ \tilde{S} &= S^{-1} + S^{-1}RVQS^{-1} \\ V &= \big(P - QS^{-1}R\big)^{-1} \end{aligned}$$

Comparing (5.33) to (5.19) in the FGaBP algorithm, we can see that $P = W^{(t_\star)}_{\mathcal{L}(i)}$, which is a single element matrix; and $R = Q^T = V$, which is a column matrix with dimensions $(n-1)$-by-1; and finally, $S = \bar{W}^{(t_\star)}$. Then, the FN update rules can alternatively be obtained

by directly computing the inverse of W and partitioning it as follows:

$$(W^{(t_\star)})^{-1} = \begin{bmatrix} \tilde{W}_{\mathcal{L}(i)} & \tilde{C}^T \\ \tilde{C} & \tilde{D} \end{bmatrix} \tag{5.34}$$

The resulting FN updates will be:

$$\alpha_{ai}^{(t)} = \frac{1}{\tilde{W}_{\mathcal{L}(i)}} - \alpha_{ia}^{(t_\star)} \tag{5.35}$$

$$\beta_{ai}^{(t)} = B_{\mathcal{L}(i)} + \frac{1}{\tilde{W}_{\mathcal{L}(i)}} (\bar{K}^{(t_\star)})^T \tilde{C} \tag{5.36}$$

Using an in-place Cholesky inversion algorithm to compute $(W^{(t_\star)})^{-1}$, the FN complexity can be reduced to $O\big(n^3/3\big)$. Since W is small and dense, optimized linear algebra libraries can be used to compute its inverse efficiently. Examples of such libraries are Eigen [104], Gmm++ [93], and Lapack [105].

For cases where n is relatively large, e.g. $n \geq 5$, computing the inverse can still be costly. As shown by the FEM distribution $\mathcal{P}$ in (5.4), the FEM solution requires only finding the marginal means as computed by the FGaBP algorithm. The marginal means are computed iteratively by the ratios of the nodal parameters β_i and α_i as shown in (5.22). Substituting (5.15) into (5.36) and rearranging terms we then obtain:

$$\bar{u}_a^{(t)} = (W^{(t_\star)})^{-1} K^{(t_\star)} \tag{5.37}$$

which can be seen as the local element approximate solution obtained from the FN_a for the VN_i $i \in \mathcal{N}(a)$ and for fixed α messages. If we partition W as $W = D - E - F$ such that D is the main diagonal of W, while E and F are the lower and the upper off-diagonals of W correspondingly, a local stationary, fixed-point, iterative process can be created within each

FN. For example, the following will constitute a GS iteration:

$$\bar{u}_a^{(t)} = (D-E)^{-1} F \bar{u}_a^{(t_\star)} + (D-E)^{-1} K^{(t_\star)}. \tag{5.38}$$

Other fixed-point iterative methods such as Jacobi or SOR can also be configured.

The α messages can be fixed after allowing the FGaBP algorithm to execute normally for a couple of iterations, or until the α messages converge to a very high tolerance such as 10^{-1}. This should replace the initial values in the α messages with better estimates. Then a number of iterations is executed using (5.37) to obtain a better estimate for $\bar{u}_a$. In all of our experiments, one or two iterations of GS was practically sufficient. Finally, the new $\beta_{ai}^{(t)}$ messages are obtained from the $\bar{u}_a^{(t)}$ estimates as follows:

$$\beta_{ai}^{(t)} = \bar{u}_i^{(t)} \alpha_i^{(t_o)} - \beta_i^{(t_\star)} + \beta_{ai}^{(t_\star)}, \tag{5.39}$$

where t_o is the iteration where the α messages are fixed. Clearly, using this approach the FN process complexity is reduced to approximately $O\big(n^2\big)$. It can be shown that the fixed-point solutions, or marginal means, of the original FGaBP update rules are equal to the fixed-point solutions of the new FGaBP update rules. However, the final message parameters, α and β, may be different between the two algorithms. We refer to this reduced complexity algorithm as the Approximate Update Finite Element Gaussian Belief Propagation (AU-FGaBP) algorithm. Table 5.1 summarizes the equations for the FGaBP FN update rules and the associated computational complexity for a relatively large n.

Table 5.1 Summary of the FGaBP update rules.

Algorithm	FN Update Equations	FN Complexity
Exact updates	(5.20), (5.21)	$O(n^4)$
Exact updates	(5.35), (5.36)	$O(n^3)$
Approximated updates	(5.37)[1] (5.38)[2] (5.39)	$O(n^2)$

[1] Fixed α messages.

[2] Using the GS iteration.

5.6 Element Merging

In this section, variations on the graphical structure of the FGaBP algorithm are proposed in order to increasing the parallel efficiency of the FGaBP execution on multicore architectures with suitable cache capacities. By virtue of the factorization structure of the FEM-FG model, we can easily redefine new factors in (5.2) by joining other factors as follows:

$$\Psi_{\hat{s}}(U_{\hat{s}}) = \Psi_{s_1}(U_{s_1})\Psi_{s_2}(U_{s_2}), \tag{5.40}$$

where factors s_1 and s_2 are joint into factor $\hat{s}$, which is a function of the variable set $U_{\hat{s}} = U_{s_1} \cup U_{s_2}$. Once suitable elements are identified for merging, the FGaBP algorithm can be executed normally after locally assembling the merged elements' matrices and source vectors, $M_{\hat{s}}$ and $B_{\hat{s}}$ correspondingly. When the factors are adjacent, the cardinality of $U_{\hat{s}}$ is $|U_{\hat{s}}| = |U_{s_1}|+|U_{s_2}|-|U_{s_1} \cap U_{s_2}|$. If U_{s_1} and U_{s_2} are not adjacent, or disjoint, then $U_{s_1} \cap U_{s_2} = \emptyset$. Element Merging (EM) can be particularly advantageous if elements that share edges in 2D meshes or surfaces in 3D meshes are merged. As a result, the memory bandwidth utilization can improve due to the considerable reduction in the total number of edges in the FEM-FG model. By merging adjacent elements, we can better utilize the high CPU computational throughput in exchange for the slower memory bandwidth and latency.

EM can be mainly useful for 2-D triangular and 3-D tetrahedral meshes with first order

FEM elements. To quantify the effect of merging, we define the Merge Reduction Ratio (MRR) percentage by dividing the amount of reduced memory from the merge by the original amount of memory before the merge. The MRR is computed as:

$$\mathrm{MRR} = \frac{\sum_{a\in\mathcal{D}} M_a - \sum_{a\in\mathcal{C}} M_a}{\sum_{a\in\mathcal{D}} M_a} \cdot 100\%, \tag{5.41}$$

where $\mathcal{D}$ is the set of all factors before any merge, $\mathcal{C}$ is the set of merged factors, and M_a is the amount of memory required for FN_a. As later shown in Section 5.7, the memory complexity per FN can be obtained by $M_i \propto O\big(4n_a + n_a^2\big)$ where n_a is the number of FN_a edges, or the rank of M_a. If we consider structured meshes for illustration purposes only, the resulting MRR is 23.8% when merging every two adjacent triangles into a quadrilateral, or 46.4% when merging every four fully connected triangles into a quadrilateral. Fig. 5.4 illustrates the different FEM-FG graphs resulting from merging every two or every four adjacent triangles in a structured triangular mesh.

For irregular triangular meshes with large numbers of triangles, the Euler's formula [106, p. 28] states that each vertex will be surrounded by six triangles on average. Thus, when merging mostly six triangle groups into hexagons, the MRR increases, on a per merge basis, to 38.9%, or to 42.9% when merging five triangles. Merging triangles does not have to be uniform. We may decide to merge patches of 2, 3, 4, 5, 6, or more as long as the triangles share edges and form connected or, preferably, convex regions. Similarly for 3D tetrahedral meshes, merging tetrahedrons sharing faces can also result in significant memory storage reductions. If two tetrahedrons of first order are merged, the MRR is 29.7%, and 53.1% when merging three tetrahedrons sharing faces and are enclosed by five vertices.

While merging elements based on a structured mesh is a trivial operation, we can still efficiently merge certain element configurations in unstructured meshes with the aid of par-

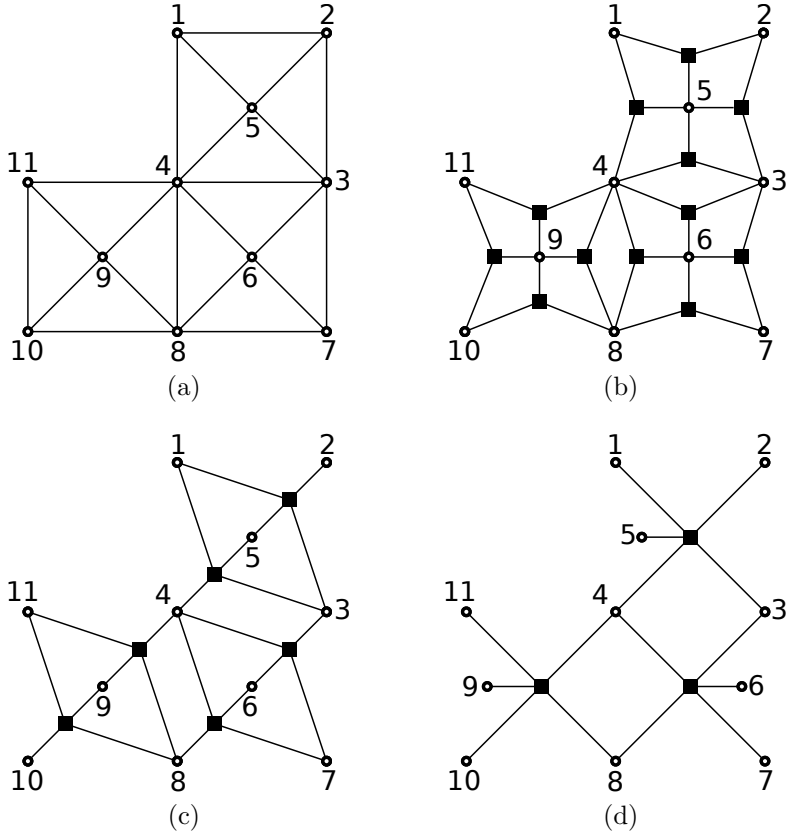


Fig. 5.4 Triangular element merging examples. (a) Original triangular mesh. (b) The initial FEM-FG using single element factorization. (c) Merging two adjacent triangles. (d) Merging adjacent four triangles.

titioning algorithms [107–110]. For such cases, the partitioning algorithms can be used to isolate the desired patches of elements. Specifically, the work in [111] presents algorithms to create macro-elements by joining adjacent elements in unstructured grids. Partitioning may add some overhead in the preprocessing stage; however, since in practice the number of factors is much greater than the number of CPU cores, a lower partition quality can be used to lower the partitioning overhead time [108] without having much impact on the overall parallel efficiency.

The element merging does not affect the underlying FEM mesh discretization proper-

ties, it does however affect the FGaBP numerical complexity as a solver. Our results in Section 5.8.4 reveal that the overall computational complexity of the merged FEM-FG can be higher than that of the original, un-merged one. Nonetheless, the FGaBP demonstrates considerable speedups for the merged FEM-FG structure, because of better utilization of available memory bandwidth and cache resources resulting from an improved computational locality. These observations are illustrated in Section 5.8.4. To conclude, we propose to use the merge feature as a high-level means to control trade-offs between CPU resources such as memory bandwidth, cache utilization, and CPU cycles to facilitate fine-tuned implementations on manycore architecture.

5.7 Implementation

5.7.1 Data-structures

Using the CPS scheme of FGaBP and assuming all FNs are of the same size, the overall storage requirement of FGaBP is $O\big(2N_v + N_f(n^2+4n)\big)$ in 64-bit words. This includes two vectors of size N_v for the VN's α_i and β_i, and a data-structure of size N_f containing the FNs data-structures where each FN's data-structure contains a dense matrix M_a, vectors B_a, α_{ai}, and β_{ai}, and a 64-bit integer vector storing local to global index associations. This setup, also, assumes that all indices are stored in 64-bit integers and all real numbers are stored in 64-bit Double-Precision Floating-Point (DP-FP), which is essential for very large problems. Since usually $O\big(N_f\big) \approx O\big(N_v\big)$, the overall FGaBP memory complexity is $O\big(N_v\big)$, typical for sparse problems. However, unlike conventional sparse data-structures such as the Compressed Sparse Row (CSR), all the FGaBP data is contiguous, or dense. Hence, the FGaBP data-structure adds minimal overhead by eliminating the need to store complicated sparsity patterns. For certain cases where the problem size does not require 64-bit indices,

e.g. when approximately $10^6 < N_v < 10^8$, 32-bit indices can be used instead. For such a case, the FGaBP memory complexity will be slightly lower to $O\big(2N_v + N_f(n^3 + 2.5n)\big)$.

In order to compare the FGaBP data-structure with the other sparse data-structures in terms of storage size, we use the Storage Ratio Metric (SRM) as follows:

$$\text{SRM} = \frac{\text{Other sparse storage}}{\text{FGaBP storage}} \tag{5.42}$$

$$= \frac{2nnz + 2N_v}{2N_v + N_f(n^3 + 4n)}. \tag{5.43}$$

In (5.43) we are considering the CSR storage size as an example for comparison due to its popularity. As shown in Fig. 2.2(c), the size of the CSR format includes a vector of size N_v for the row start index, and two vectors of size nnz for the element value and its row index. In addition, the right-hand-side vector of size N_v is added to the CSR format, in order to provide a consistent comparison with the FGaBP format which already includes the elements' source vectors in its storage. It is important to note that in practice the sparse storage for other solvers is typically much larger than what is used here in the SRM; since, iterative solvers such as the PCG, as in Algorithm 1, would required additional memory to store the preconditioner as well as the search and residual vectors. On the other hand, the FGaBP data-structure includes all the information needed to solve the FEM problem rather than just storing the sparse matrix of the global linear system. However, since implementations of the PCG algorithm may vary from one library to another, we will not consider its full storage in the SRM analysis, but rather only the storage required for the sparse matrix and the right-hand-side vector.

5.7.2 The FEM-FG Edges

Another metric we use in our analysis is the Link Ratio Metric (LRM) defined as follows:

$$\text{LRM} = \frac{\text{Number of edges in the PWGM}}{\text{Number of edges in the FEM-FG model}} \tag{5.44}$$

$$= \frac{(nnz - N_v)/2}{N_f n}. \tag{5.45}$$

The LRM is a measure for the amount of information the algorithm is required to communicate per iteration. As we will see later in Section 5.8, this ratio may explain some of the convergence behavior of FGaBP in comparison to other sparse algorithms. It is important to note that the LRM is not a measure for the memory bandwidth requirement of a particular algorithm. The SRM may be a closer measure to the memory bandwidth required by the solver per iteration. However, the memory bandwidth is not easy to measure because, not only does it depends on the amount of data needed to be accessed, but rather it depends more on the access pattern of the sparse data-structure. That is because at the hardware level, the memory controllers respond to the CPU's memory requests by sending data in contiguous chunks, as in memory lines, which is stored temporarily in the cache memory [112]. The CPU on the other hand, uses the cache memory to retrieve exactly what it needs. This is the key reason behind the notoriously poor performance of sparse data-structures. If the data is not contiguous in memory, then the CPU only utilizes a small fraction of the data transfer to the cache. This leads to under-utilized cache or poorly utilized memory bandwidth causing an overall poor computational throughput. The FGaBP, on the other hand, utilizes contiguous data-structures which considerably improves its memory bandwidth utilization.

Table 5.2 illustrates the SRM and the LRM trends for a number of meshes. The L-shaped domain is used as the example domain. The open-source software Gmsh [113] is used to mesh

the domain. The triangular, tetrahedral and quadrilateral meshes are unstructured, while the hexahedral meshes are structured and also generated by Gmsh. The stiffness matrix was obtained for the Laplace equation using the software package GetFEM [28]. Using the stiffness matrix, we can obtain the number of edges for the corresponding PWGM using the actual *nnz* of the matrix. Clearly, for both ratio metrics the FGaBP shows a progressively increasing advantage as the FEM order increases for all types of meshes. In particular, the LRM shows considerable advantage for all types of meshes except the first order triangular and tetrahedral meshes. However, this can be efficiently remedied using the EM feature discussed earlier in Section 5.6.

For certain cases, the LRM data can be analytically explained. Considering the Euler theorem [106, p. 28] for triangular meshes of the first order, the ratio of edges to elements is 3/2, now dividing by the number of factor variables (which is $n = 3$ for triangular factors), then the resulting ratio should be 0.5 as predicted by the tabulated results for first order triangles. If we consider approximating the unstructured quadrilateral mesh with a regular one, with uniform gird points, and noting that for an isolated quadrilateral element there are six FEM edge couplings, then the ratio of edges to elements approaches 4 which explains the 1.01 ratio. Similarly for hexahedral meshes on cubical domains of dimension N, the number of edges for N^3 hexahedrons approaches $14N^3 + O\big(N^2\big)$ leading to a ratio of 14/8 which explains the 1.79 and 1.73 ratios.

5.7.3 CPU Multicore Implementation

The FGaBP code was designed using C++ Object Oriented Programming (OOP) [114, 115]. The OOP based design of the FGaBP software facilitates its integration with existing frameworks of open-source libraries such as deal.II [27] and GetFEM++ [28]. The following section presents the numerical results on the performance of both the basic FGaBP and the

Table 5.2 The FEM-FG comparison analysis.

Element Geometry	n [1]	N_v	N_f	nnz	PWGM Edges	Sparse Storage[2]	FEM-FG Edges	FEM-FG Storage [3]	SRM	LRM
triangle	3	349	634	2313	982	6372	1902	14012	*0.45*	*0.52*
triangle	6	1331	634	14831	6750	36318	3804	40702	*0.89*	*1.77*
triangle	10	2947	634	48967	23010	112670	6340	94654	*1.19*	*3.63*
triangle	15	5197	634	119937	57370	265860	9510	191084	*1.39*	*6.03*
triangle	3	1092	2070	7414	3161	20289	6210	45654	*0.44*	*0.51*
triangle	6	4253	2070	48059	21903	117384	12420	132706	*0.88*	*1.76*
triangle	10	9484	2070	159196	74856	365813	20700	308768	*1.18*	*3.62*
triangle	15	16785	2070	390505	186860	864936	31050	623520	*1.39*	*6.02*
quadrilateral	4	1057	1000	9169	4056	23624	4000	34114	*0.69*	*1.01*
quadrilateral	9	4113	1000	64449	30168	149464	9000	125226	*1.19*	*3.35*
quadrilateral	16	9169	1000	225841	108336	497528	16000	338338	*1.47*	*6.77*
tetrahedron	4	689	2668	8343	3827	20132	10672	86754	*0.23*	*0.36*
tetrahedron	10	4521	2668	112989	54234	248584	26680	382562	*0.65*	*2.03*
tetrahedron	20	14165	2668	619257	302546	1309340	53360	1308970	*1.00*	*5.67*
hexahedron	8	2448	1815	54400	25976	121041	14520	179136	*0.68*	*1.79*
hexahedron	27	16951	1815	966985	475017	2018726	49005	1553057	*1.30*	*9.69*
hexahedron	8	4896	3993	115574	55339	255629	31944	393120	*0.65*	*1.73*
hexahedron	27	35443	3993	2099065	1031811	4375346	107811	3413027	*1.28*	*9.57*

[1] n is the number of variables per element.
[2] Using CSR format.
[3] All storage estimates are in the order of DP-FP numbers.

modified AU-FGaBP algorithms. We utilize meshes and FEM setup generated from three open-source libraries, which are Gmsh, GetFEM++, and deal.II. We validate the FGaBP algorithms on both regular and irregular meshes using various FEM interpolation orders and geometrical elements shapes such as triangles, quadrilaterals, and hexahedrons. We utilize 2D and 3D domains with various boundary conditions.

For CPU multicore implementations, the OpenMP [116] standard Application Programming Interface (API) is used to parallelize the FGaBP code. Using the CPS scheme, the color loops are parallelized such that each thread executes a FN routine. Thread synchronization directives are used only at the execution end of each color group; that is, the number of synchronization directives is equal to the number of color groups in the CPS scheme.

5.8 The FGaBP Numerical Results and Discussions

5.8.1 FGaBP Verification

In this section, we verify the numerical results of the new AU-FGaBP formulation using the definite Helmoltz problem with known solution as provided by the deal.II library for 2D and 3D domains as well as higher order FEM elements. The definite Helmholtz is formulated as follows:

$$-\nabla \cdot \nabla u + u = g, \text{ on } \Omega \tag{5.46}$$

$$u = f_1, \text{ on } \partial D \tag{5.47}$$

$$\mathbf{n} \cdot \nabla u = f_2, \text{ on } \partial N \tag{5.48}$$

where ∂D and ∂N are the Dirichlet and Neumann boundaries such that $\partial D \cup \partial N = \partial \Omega$, and Ω is the square or cubic domain bounded by $[-1, 1]$. Equation (5.48) constitutes the

non-homogeneous Neumann boundary condition. The right-hand-side functions are set, such that the exact solution is:

$$u(p) = \sum_{i}^{3} \exp\left(-\frac{|p - p_i|^2}{\sigma^2}\right) \tag{5.49}$$

where p is a spatial variable in (x, y, z), p_i are exponential center points chosen arbitrarily, and $\sigma = 0.125$. The library deal.II creates the mesh, and provides the FGaBP class with the elements' M_s, B_s matrices, as well as, the local to global index data. The FGaBP processes the FEM problem element-by-element in parallel and sends the end solution back to deal.II which computes the final error relative to the exact solution.

The AU-FGaBP uses α message tolerance of 10^{-1} and two GS iterations. The test cases are obtained by varying the FEM element order from the 1^{st} to the 3^{rd} order for both 2D quadrilaterals and 3D hexahedrals. Fig. 5.5 shows the error plots for each test case. The global error for each test case is obtained by summing the squares of the l^2-norm error of each element and then taking the square root of that value. It can be seen that the AU-FGaBP obtains the expected error trends for the FEM showing increased accuracy on all test cases for both trends of increasing number of elements as well as increasing FEM order.

5.8.2 Performance Comparison

The FGaBP algorithm performance is demonstrated by solving Poisson's equation on a Shielded Strip Conductor (SSC) with two dielectric media as shown in Fig. 5.6(a). The domain includes Dirichlet and homogeneous Neumann boundary conditions and is meshed with an irregular triangular mesh. Different trials are performed by increasing the order of the FEM interpolation from the 1^{st} order to the 5^{th} order while reducing the number of elements so as to keep the number of unknowns approximately similar. As can be seen from the LRM data, the FEM-FG model reduces memory communication for element orders

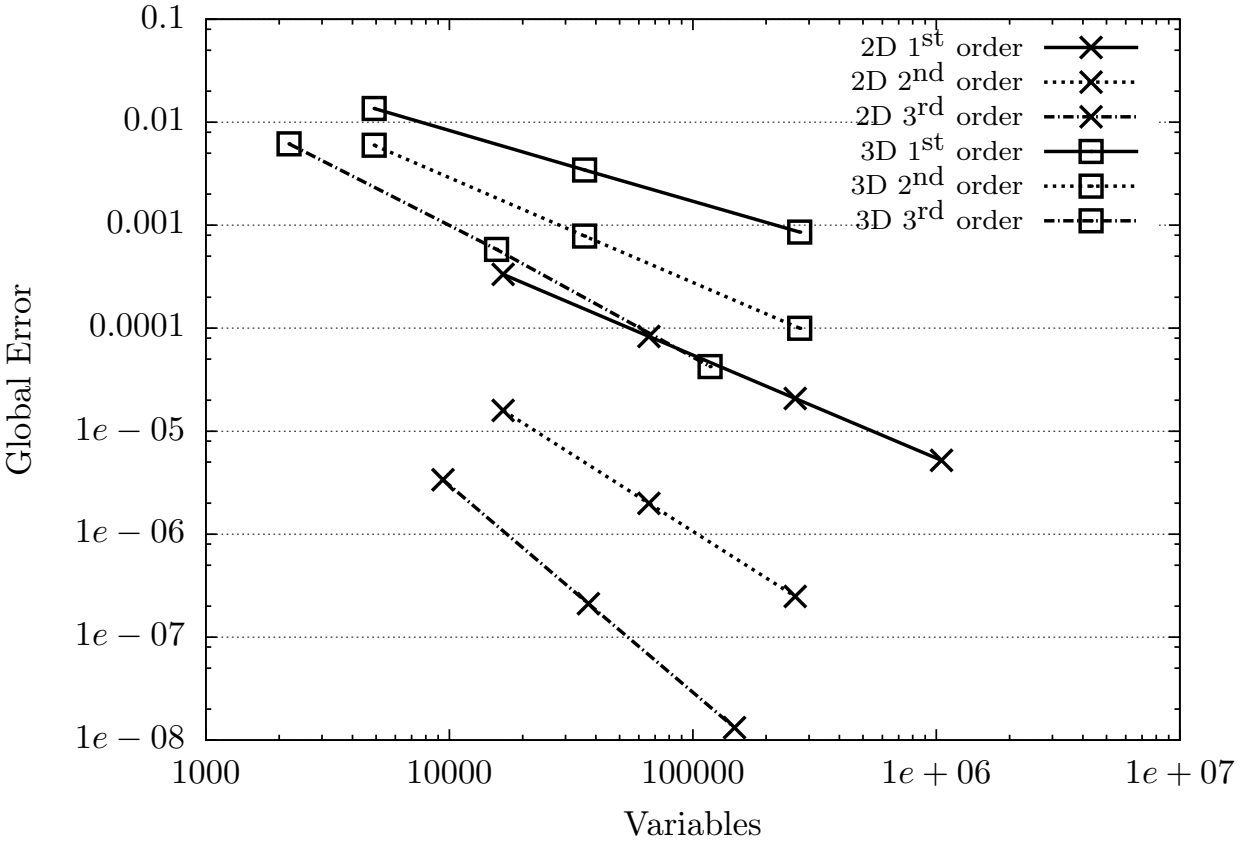


Fig. 5.5 The global error of the AU-FGaBP obtained from element-based l^2-norm error relative to the exact solution.

greater than one by reducing the number of nodal links in the resulting FEM-FG model compared to solvers such as the PW-GaBP. With respect to algebraic solvers such as the CG, the link number directly correlates with the required memory communication.

Also shown in Table 5.3 are the iteration results of the FGaBP, the PW-GaBP, and the D-PCG. The iterations were terminated on all experiments when the l^2-norm was dropped to 10^{-9}. We used our automatic relaxation scheme, previously introduced in Chapter 4, to accelerate the convergence of both the PW-GaBP and the FGaBP. The PW-GaBP used Nodal Message Relaxation (NMR); whereas, the FGaBP used Edge Message Relaxation

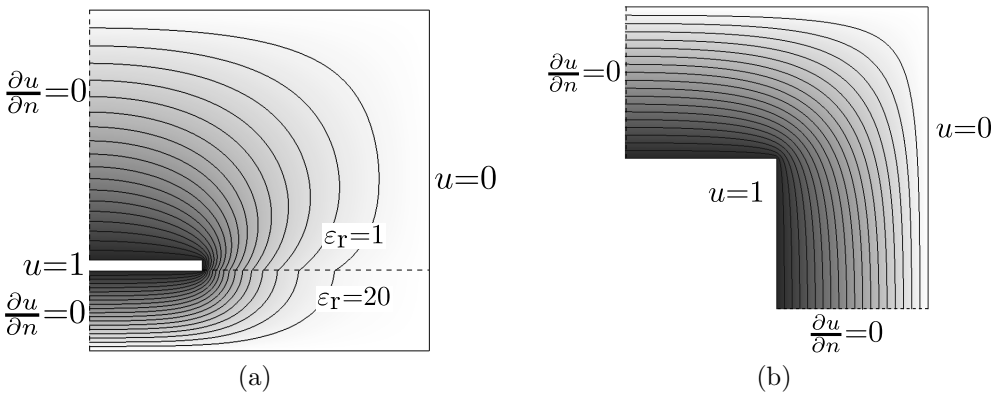


Fig. 5.6 The Laplace equipotential contours obtained using FGaBP, the dimensions of both structures are within the unit interval $[0, 1]$ cm. (a) The right symmetrical side of the shielded strip between two different materials. (b) The top-right symmetrical corner of the square coaxial conductor.

(EMR). It is important to note that the PW-GaBP and the D-PCG require global large sparse matrix operations in each iteration which limit their effectiveness for parallel implementations; while in contrast, the FGaBP requires completely decoupled computations performed element-by-element. The last column of Table 5.3 shows the SU results of FGaBP using a CPS implementation on a quad-core CPU over the sequential one on the same workstation. Although the implementation was not optimized to take advantage of specialized CPU features, the FGaBP illustrated increased efficiency of parallel implementation as the interpolation order increases. This is due to the FGaBP making good use of increased localized computations and reduced memory communications which is a direct result from the FEM-FG graph that reduces nodal links.

While the D-PCG algorithm demonstrated the lowest iteration count, the FGaBP algorithm demonstrated a trend of reducing number of iterations as the FEM order increases; which illustrates its increased computational stability. This is mainly due to the FGaBP algorithm performing localized computations on small dense matrices up to order 5 instead of operating on very large sparse matrices that become increasingly ill-conditioned as the

order of the FEM interpolation increases. This is a key advantage in comparison to algebraic based solvers such as the PCG whose iteration count increases as the condition number of the global characteristic matrix increases due to increasing FEM order.

Even though at this stage, the FGaBP algorithm does not seem to be competitive enough with the PCG solver iteration-wise, this problem will be remedied by introducing a multigrid scheme for the FGaBP algorithm as will be shown later in Chapter 6. Lastly, as can also be seen in Table 5.3, the PW-GaBP failed to converge as the interpolation order increased beyond the 2^{nd} order, since the resulting global matrix is becoming increasingly ill-conditioned. In contrast, the FGaBP successfully converged for all cases demonstrating a significant improvement over the PW-GaBP, and likewise the GS and the SOR solvers. This result may be attributable to the structure induced by the FEM-FG model, which, unlike the PWGM model, factors the graph over cliques causing the FN computation in the FGaBP to correlate more local variables which increases the stability of the overall algorithm. Fundamentally, factoring the graph over cliques significantly reduces the number of loops in the graph which causes the BP computation to be more stable. This result incites further investigation on the convergence behavior of Gaussian BP type algorithms for both the FEM-FG model as well as general GMs.

Table 5.3 Shielded microstrip results for FGaBP, PW-GaBP, and D-PCG.

Order	N_v	N_f	LRM	PW-GaBP itrs [1]	D-PCG itrs	FGaBP itrs	*CPU SU* [2]
1^{st}	10076	19252	0.46	542	239	1486	*1.1*
2^{nd}	9818	4787	1.67	3582	379	3221	*1.3*
3^{rd}	10108	2191	3.42	-	447	1762	*2.1*
4^{th}	10131	1237	5.69	-	608	1564	*3.2*
5^{th}	10056	786	8.44	-	826	1362	*3.4*

[1] The shorthand itrs refers to iterations.

[2] The speedup is relative to sequential FGaBP.

5.8.3 FGaBP Convergence Trends

Fig. 5.7 shows the convergence behavior of the FGaBP for 2D and 3D meshes of quadrilateral and hexahedral elements. The FEM problem is solving Laplace on the L-shaped domain Fig. 5.6(b). The FEM assembly function was performed using the open-source library deal.II [27]. Clearly, the FGaBP displays near linear convergence with increasing problem size. Note that, the slight variations in the trends are mainly due to the automatic relaxation algorithm. In addition, the FGaBP showed similar trends of lower number of iterations as the complexity of the local factors increased either in terms of space dimension or FEM order.

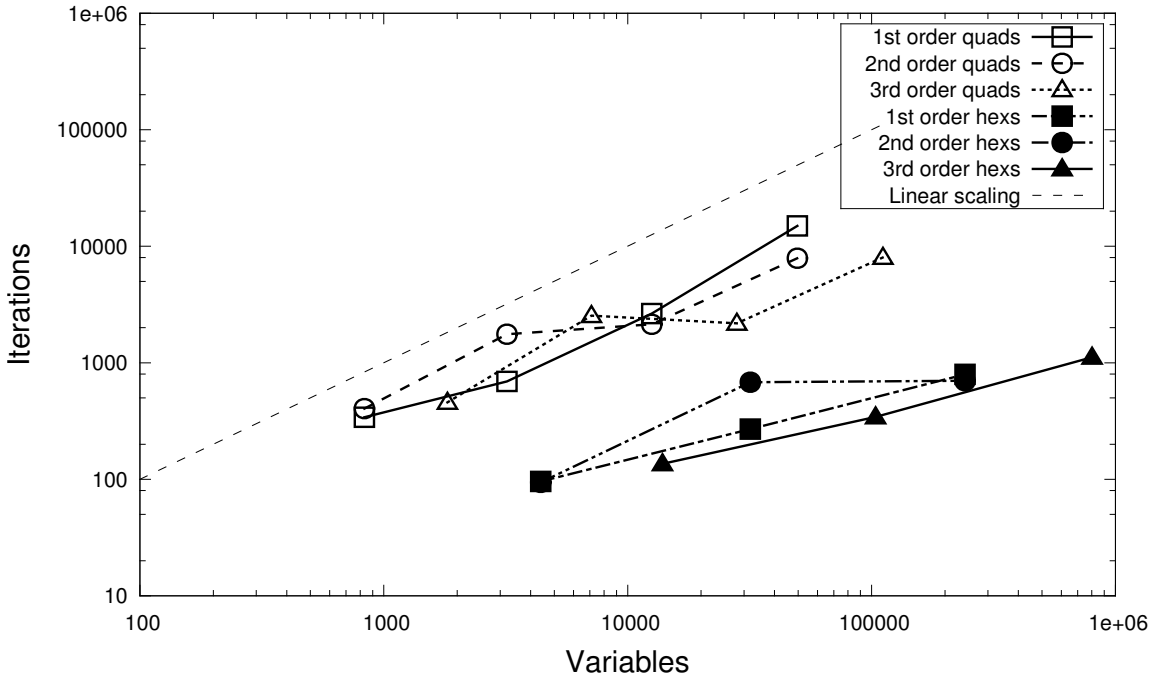


Fig. 5.7 The FGaBP algorithm executed on quadrilateral and hexahedral meshes of different orders.

5.8.4 Element Merging Performance

The EM feature implemented with the FGaBP algorithm is demonstrated using a structured triangular mesh on a unit square domain. The Laplace equation is solved in the domain

using zero Dirichlet on the boundary. The unit square is subdivided into equally spaced sub-squares where each square is further divided into two right triangles. We perform two levels of merges by merging each two adjacent triangles, and then each four adjacent triangles. The original mesh has $N_f = 200000$ triangular element FNs. Relaxation was not used in order to isolate the effect of merging on the iteration count. The algorithm iterations were terminated when the message relative l^2-norm reached $< 10^{-9}$. Table 5.4 shows the speedup results of the merges. Execution was performed on an Intel Core2 Quad CPU with clock frequency of 2.83 GHz.

The merge results in increasing speedups. Noting the overall computational complexity of the measured algorithm is approximately $O\big(TN_f(n^3+n^2)\big)$, where T is the total number of iterations, the overall computational complexity of the merged algorithms has actually increased as can be noted by the complexity ratio column. However, the execution time was actually faster which was mainly due to the improved locality of the algorithm. This improved locality results in better trade-offs of cache and memory bandwidth for cheaper CPU flops.

Table 5.4 AU-FGaBP with Element Merge Speedups.

Merge	N_f	n	Iteration ratio[1]	Complexity ratio[2]	*SU*
un-merged	200000	3	1.0	1.0	*1.0*
2-triangle	100000	4	1.08	0.972	*1.34*
4-triangle	50000	6	1.35	0.771	*2.89*

[1] Iterations ratio = iterations of un-merged / merged.

[2] Complexity ratio = complexity of un-merged / merged.

5.9 Conclusion

In this chapter, we presented the novel FGaBP algorithm which performs inherently parallel FEM computations element-by-element. We presented a variational inference reformulation for the FEM in order to facilitate the use of the BP computational inference algorithms. The new probabilistic FEM-FG model was introduced to address general FEM problems which facilitates the derivation of various forms of the FGaBP algorithm. The approximate update and element merging variants of the FGaBP algorithm were introduced which illustrates the FEM-FG model flexibility in reducing computational complexity and improving the memory bandwidth utilization. The FGaBP algorithm was illustrated to solve the FEM problem in parallel, element-by-element, eliminating the need for large sparse matrix operations. The scale of the tested problems demonstrates that the algorithm is also suited for large-scale FEM implementations on emerging parallel architectures. The FGaBP algorithm demonstrated significant scalability advantages in terms of both computation and memory bandwidth for increasing FEM element order in both 2D and 3D domains. In the following chapter, we will introduce a multigrid scheme for the FGaBP that will significantly reduce its iteration count, which makes it competitive with state-of-the-are solvers such as the Multigrid-Preconditioned Conjugate Gradient (MG-PCG).

CHAPTER 6

Parallel Multigrid Adaptation for the Finite Element Gaussian Belief Propagation Algorithm

In this chapter we introduce a novel parallel multigrid algorithm, referred to as the Finite Element Multigrid Gaussian Belief Propagation (FMGaBP) algorithm, to accelerate the convergence of the previously introduced FGaBP algorithm. The FMGaBP reduces the iteration count required for convergence to a competitive level while maintaining all the parallelism features of the original FGaBP algorithm. The FMGaBP algorithm processes the FEM computation in a fully distributed and parallel manner, with stencil-like element-by-element operations, while eliminating all global algebraic operations. To our knowledge, this is the first multigrid formulation for continuous domain Gaussian BP type algorithms. Similar to the FGaBP algorithm, the FMGaBP algorithm finds the solution to the variational formulation of the FEM problem in a distributed manner. In comparison with MG-PCG solvers, the FMGaBP algorithm demonstrates considerable iteration reductions as tested by Laplace benchmark problems. In addition, the parallel implementation of the FMGaBP algorithm shows a speedup of up to 2.9 times over the parallel implementation of MG-PCG using eight CPU cores.

6.1 Introduction

The PW-GaBP algorithm, introduced by [33,54], was discussed in Chapters 3 and 4 illustrating great potential to parallelize the solving stage of the FEM [55,57]. In addition, PW-GaBP has been shown to outperform classical iterative solvers such as GS and Jacobi [54]. However such algorithms, which are derived based on pairwise interconnect assumptions on the underlying GM, suffer mostly from a lack of convergence when diagonal dominance properties are not met. In Chapter 5, the FGaBP algorithm was introduced which addresses the convergence shortcomings of PW-GaBP and improves the parallel efficiency of the FEM computation. While the FGaBP was demonstrated to efficiently solve FEM problems in parallel, element-by-element; however the FGaBP's convergence rate tends to stall when executed on fine meshes.

In this work we address this issue by introducing the novel FMGaBP algorithm. Our results for various scheduling versions of FMGaBP demonstrate high convergence rates independent of the scale of discretization on the finest mesh. In addition, we show for a benchmark Laplace problem that the FMGaBP requires, in certain cases, less iterations than the MG-PCG solver. The eight thread multicore implementation of FMGaBP demonstrates speedups of up to 2.9 times over the MG-PCG solver based on sparse algebraic operations as implemented by the multithreaded library deal.II [27].

The chapter is organized as follows. In Section 6.2 we present the detailed formulation of the FMGaBP algorithm. Then in Section 6.3 we present the implementation details of the FMGaBP algorithm. Finally in Section 6.4 we conclude with numerical and performance results and discussions.

6.2 The FMGaBP Algorithm

Multigrid acceleration schemes [79, 80] provide optimal convergence speeds resulting in a number of iterations almost independent of the discretization level of the FEM mesh. It is expected that the FGaBP algorithm can benefit greatly from multigrid schemes mainly because, BP communications on coarser levels can serve as bridges to communications between far away nodes on finer levels. This has the effect of considerably improving the propagation of information in the FEM-FG model and hence speeding up the overall convergence. In this section, we introduce an efficient adaptation of multigrid schemes to the FGaBP algorithm resulting in the FMGaBP algorithm. As will be shown later, the resulting FMGaBP algorithm demonstrates convergence rates independent of the FEM domain's discretization level. In addition, the FMGaBP retains the distributed nature of computations, which has important implications for HPC implementations. Mainly, the FMGaBP level transfer operations are computationally decoupled and localized without requiring any global sparse algebraic operations.

The FMGaBP algorithm considerably improves the FGaBP convergence rate while maintaining high parallel efficiency of stencil-like, element-by-element, operations. Our results will later demonstrate that not only the FMGaBP shows considerable advantage in parallel efficiency, but also, compared to the MG-PCG, the FMGaBP shows iteration reductions in certain cases. The distinguishing feature of the FGaBP algorithm is solving the FEM in parallel, element-by-element, without assembling a large sparse matrix or performing any global algebraic operations such as SMVMs. This key advantage, which has important impacts on parallel hardware implementations, is maintained by the new multigrid formulation for FMGaBP.

6.2.1 Hierarchical Mesh Refinement

A brief overview of multigrid schemes was presented in Section 2.1.4. In our derivation of the FMGaBP algorithm, we will be considering GMG schemes with hierarchical meshes. In hierarchical mesh refinement schemes, local element splitting is used to generate a hierarchy of multigrid levels. As an example, Fig. 2.3(a) shows hierarchical refinement by starting from an initial coarse triangular mesh and then splitting each triangle into four geometrically similar child triangles by inserting nodes at midpoints of the parent triangle's edges. Similar refinement can be used for quadrilateral and hexahedral meshes, where each quadrilateral (parent) is split into four child-quadrilaterals and each hexahedron (parent) is split into eight child-hexahedrons. Each level of the hierarchy will be represented by a unique FEM-FG graph; as will be shown later, the transfer operations will only occur between the factor nodes of the corresponding parent-child elements. In addition, this hierarchical refinement scheme will provide key advantages for element-by-element parallel behavior of the FMGaBP. Also, by utilizing semi-irregular mesh hierarchies, more adaptation to arbitrary domains can be achieved than by using only structured meshes hierarchies.

However, and more importantly, the choice of the hierarchical refinement is not due to any imposed limitations from the FMGaBP algorithm, but rather, it is used as an initial proof-of-concept phase to better illustrate the new FMGaBP algorithm. One approach to handle geometries with fine and complex features is to use non-conforming adaptive refinement [117,118] which will require special care to handle the non-conforming grid points. In addition, adaptive refinement schemes rely on computing error estimations from a local element-by-element perspective which makes formulating the FMGaBP for such refinement schemes an interesting future research direction.

6.2.2 The Factor Node Belief

To illustrate the FMGaBP formulation, we start by defining a multivariate distribution, associated with each individual FN_a, referred to as the factor node belief $b_a(U_a)$, denoted for short as b_a. The belief b_a function takes the form:

$$b_a^{(t)}(U_a) \propto \Psi_a(U_a) \prod_{i \in \mathcal{N}(a)} \eta_{ia}^{(t)}(U_i), \tag{6.1}$$

where U_a is the vector of random unknowns linked to FN_a. In essence the belief b_a, in contrast with the nodal belief b_i as defined in (2.24), represents the marginal distribution as seen from FN_a in a particular iteration t. When we use the FEM-FG setting, the factor belief b_a takes a Gaussian multivariate form as:

$$b_a^{(t)}(U_a) \propto \exp\left[-\frac{1}{2} U_a^T W_a^{(t)} U_a + (K_a^{(t)})^T U_a\right] \tag{6.2}$$

where the parameters W_a and K_a represent the inverse covariance and the source vector of the factor node respectively. Note that the function b_a takes an iterative form, where the matrix W_a and the vector K_a are updated each iteration according to the FGaBP update rules in (5.16).

By observing the dynamics of the message update rules of the BP algorithm in (2.22), (2.23), and (2.24), we can show that at message convergence the joint mean vector of b_a, given by $W_a^{-1} K_a$ as computed locally from the factor node's perspective, will be equal to the marginal means of U_a as computed from the nodal perspective by (5.22). That is, if we form a vector of nodal marginal means as computed by the FGaBP rule (5.22) in the vector

$\bar{\mu}_a$ for the set of nodes connected to FN_a, then at message convergence we obtain:

$$\bar{\mu}_a = \left[\mu_{\mathcal{L}(j)}\right], \text{ where } j \in \mathcal{N}(a) \tag{6.3}$$

$$= W_a^{-1} K_a \tag{6.4}$$

and $\mathcal{L}(\cdot)$ is the mapping of global variable node indices to local factor node indices.

In order to demonstrate this result, we start by formulating the marginal for U_i, denoted as $\tilde{b}_i(U_i)$, from the defined b_a using the general BP message communication as follows:

$$\tilde{b}_i^{(t)}(U_i) \propto \int_{U_{\mathcal{N}(a)\setminus i}} b_a^{(t)}(U_a)\, \mathrm{d}U_{\mathcal{N}(a)\setminus i} \tag{6.5}$$

$$\propto \int_{U_{\mathcal{N}(a)\setminus i}} \Psi_a(U_a) \prod_{j\in\mathcal{N}(a)} \eta_{ja}^{(t)}(U_j)\, \mathrm{d}U_{\mathcal{N}(a)\setminus i}. \tag{6.6}$$

Since at $j = i$, $\eta_{ia}(U_i)$ is a function of the single variable node U_i, hence we can take it outside the integral as:

$$\tilde{b}_i^{(t)}(U_i) \propto \eta_{ia}^{(t)}(U_i) \int_{U_{\mathcal{N}(a)\setminus i}} \Psi_a(U_a) \prod_{j\in\mathcal{N}(a)\setminus i} \eta_{ja}^{(t)}(U_j)\, \mathrm{d}U_{\mathcal{N}(a)\setminus i}. \tag{6.7}$$

Here, we are considering the case where all the BP messages are stationary, then we can substitute the BP update rules (2.22), (2.23) and (2.24) which results in:

$$\tilde{b}_i^{(t)}(U_i) \propto \left[\prod_{k\in\mathcal{N}(i)\setminus a} m_{ki}^{(t)}(U_i)\right] m_{ai}^{(t)}(U_i) \tag{6.8}$$

$$\propto \prod_{k\in\mathcal{N}(i)} m_{ki}^{(t)}(U_i) \tag{6.9}$$

$$\propto b_i^{(t)}(U_i). \tag{6.10}$$

The above result states that, at message convergence the marginal of a particular variable node computed using a factor belief function is equivalent to the marginal distribution of that node computed from the nodal perspective incorporating global information as in (5.22). Now, since for multivariate Gaussian distributions the marginal mean of a variable is equal to its corresponding value in the joint mean vector; therefore, at message convergence, the Gaussian means as computed from the local belief perspective will be in agreement with the marginal means as computed from global perspective as stated in (6.4).

6.2.3 Local Belief Residual-Correction

Given the belief condition in (6.4), then for each factor on a fine grid we can formulate a quantity referred to as the belief residual r_a given by:

$$r_a^{(t)} = K_a^{(t)} - W_a^{(t)} \bar{\mu}_a^{(t)}. \tag{6.11}$$

When hierarchical mesh refinement is used, the belief residuals of each group of child elements can locally be restricted into the parent element as:

$$K_a^H = \mathcal{R}_l r_a^h \tag{6.12}$$

where K_a^H is the source vector of the parent element, r_a^h is the accumulated local residual of child elements, and $\mathcal{R}_l$ is the child-parent local restriction operator. Notice that we dropped the iteration count (t) since we are operating in the same iteration for both sides of the equations.

Similarly, we can use local interpolation operations in order to apply the corrections from

the coarse elements as follows:

$$\bar{\mu}_a^h \leftarrow \bar{\mu}_a^h + \mathcal{I}_l \bar{\mu}_a^H. \tag{6.13}$$

Using the level updated $\bar{\mu}_a^h$ we can reinitialize the corresponding level local messages using again (6.11) but with $r_a \to 0$, then solving for the local factor messages as follows:

$$K_a^h = W_a \bar{u}_a^h \tag{6.14}$$

$$\mathcal{B}_a^h = K_a^h - B_a^h. \tag{6.15}$$

Since we are only considering linear problems, the α messages are only dependent on the factor matrices which are not affected by the residual-correction multigrid scheme. Therefore, the α messages retain their previous value on each level transfer. It was found by experimentation on first order FEM problems, that the $\mathcal{R}_l$ is typically the transpose of $\mathcal{I}_l$ up to a constant factor as follows:

$$\mathcal{R}_l = c\,(\mathcal{I}_l)^T \tag{6.16}$$

where c was found to equal to 1 for structured quadrilateral and hexahedral meshes, while it was equal to $\frac{1}{3}$ for semi-irregular triangular meshes.

6.2.4 Local Transfer Operations

The FMGaBP local transfer operations can be enhanced by the definition of local node numbering schemes between each set of parent-child elements. The parent-child numbering scheme facilitates the transfer of residuals from child nodes to parent nodes using logical index computations. As a result, the FMGaBP transfer operations take the form of local stencil-like operation. Fig. 6.1 shows the residual transfer operations from one child node to the corresponding parent nodes using a particular local numbering scheme. The local

transfer scheme can be generalized to 3D meshes. For example, when refining a hexahedron into eight hexahedrons, the nodes on the faces of the refined hexahedron will have weightings similar to the ones illustrated in Fig. 6.1, while the node in the interior will have a weighting equal to $\frac{1}{8}$. In the case of triangular and tetrahedral meshes, the nodal weights will consist of 1 and $\frac{1}{2}$ only.

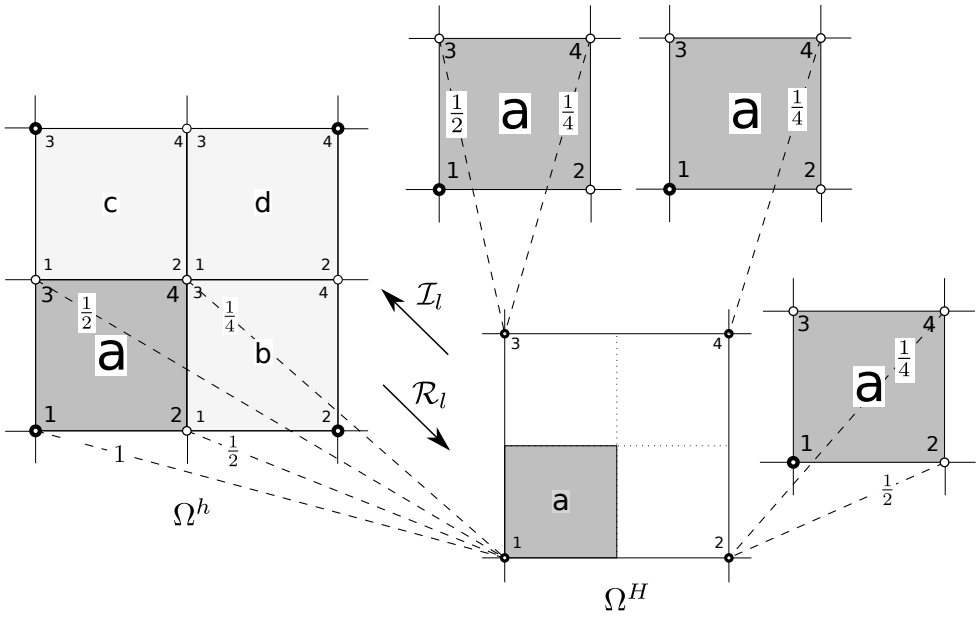
Finally, It is important to note that the local FMGaBP transfer operations between each set of parent-child elements create parallel operations that are also thread-safe. This means that these local transfer operations can be implemented in an embarrassingly parallel stencil-like operations.

6.2.5 The Global Residual and the Local Belief Residuals

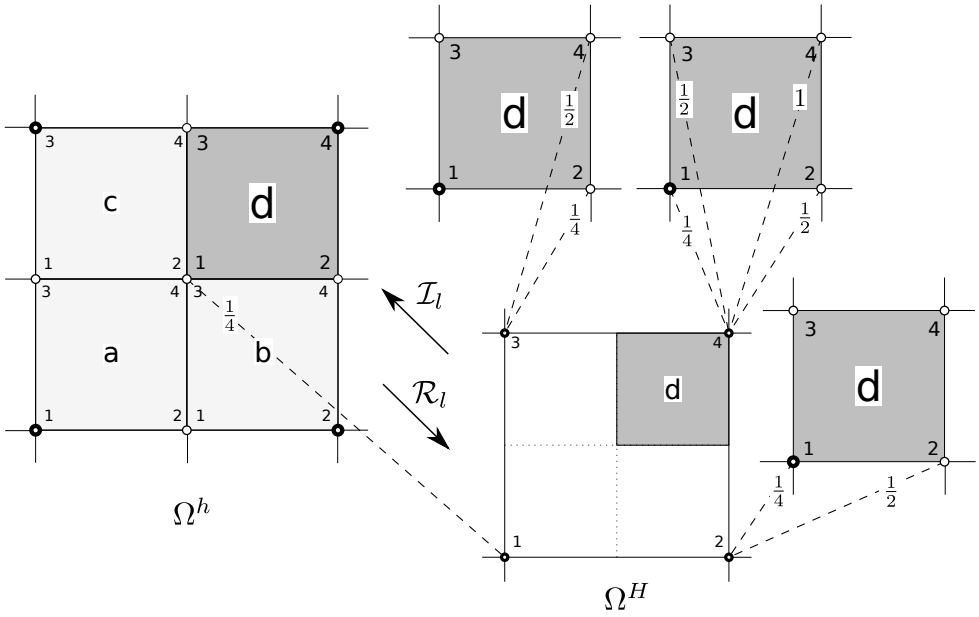
To illustrate the relationship between the local belief residuals and the global residual, we start by assembling a global residual vector $r_{\mathcal{G}}$ by summing all the local residual components from each factor node as follows:

$$r_{\mathcal{G}} = \sum_{a \in \mathcal{S}} r_a \tag{6.17}$$

where $\mathcal{S}$ is the set of all factor nodes. Note that the sum in (6.17) is carried out by resolving mappings from local factor indices to global indices. In the following formulas, we consider this mapping to be implicitly stated in the sum form. Also in order to define the residual for a particular BP iteration, we assume a synchronous schedule for message communications. This does not lead to any loss of generality, but rather it greatly simplifies the formulation since other forms of schedules may not have a clearly definable iteration. Alternatively, one way to represent iterations is to define an iteration for every single message update. For such a case, our formulation will also hold.



(a) Child-a



(b) Child-d

Fig. 6.1 FMGaBP local transfer weights between two example child elements and the corresponding parent element. (a) Local transfer weights for child-a. (b) Local transfer weights for child-d.

Next we substitute the local belief residual (6.11) into (6.17) to get:

$$r_{\mathcal{G}} = \sum_{a \in \mathcal{S}} [K_a - W_a \bar{\mu}_a] \tag{6.18}$$

$$= \sum_{a \in \mathcal{S}} [B_a + \mathcal{B}_a - M_a \bar{\mu}_a - \mathcal{A}_a \bar{\mu}_a] \quad \text{substituting (5.16).} \tag{6.19}$$

Note that the terms $\sum_{a \in \mathcal{S}} B_a$ and $\sum_{a \in \mathcal{S}} M_a \bar{\mu}_a$ represent the assembly of the global system f and $A\bar{\mu}$ correspondingly, then we have:

$$r_{\mathcal{G}} = f - A\bar{\mu} + \sum_{a \in \mathcal{S}} [\mathcal{B}_a - \mathcal{A}_a \bar{\mu}_a] \tag{6.20}$$

where the dimensions of the system A and f are N, where N is the total number of unknowns; and $\bar{\mu}$ is a vector of the same dimension representing the solution estimate obtained from the current FGaBP level iteration. Recall that $\mathcal{B}_a$ and $\mathcal{A}_a$ represent the factor node's incoming messages as in (5.16) where the incoming messages α_{ia} and β_{ia} can each be substituted by (5.14) and (5.15) correspondingly, then we can assemble $\mathcal{B}_a$ and $\mathcal{A}_a$ as follows:

$$\sum_{a \in \mathcal{S}} \mathcal{A}_a = C\mathcal{A}_{\mathcal{G}} \tag{6.21}$$

$$\sum_{a \in \mathcal{S}} \mathcal{B}_a = C\mathcal{B}_{\mathcal{G}} \tag{6.22}$$

where $\mathcal{A}_{\mathcal{G}}$ is a diagonal matrix with dimension N-by-N and off-diagonals equal to zero, where N is the total number of unknowns; $\mathcal{B}_{\mathcal{G}}$ is a global vector of the same dimension; and C is a diagonal matrix of the same dimension with constant diagonals that will be clarified shortly. The diagonal of $\mathcal{A}_{\mathcal{G}}$ contains the nodal values α_i which are basically the sum of incoming α factor messages as in (5.12); and similarly, the vector $\mathcal{B}_{\mathcal{G}}$ contains the sum of incoming

β factor messages as in (5.13). Each diagonal element of C is equal to the corresponding nodes' number of links minus one.

It is easier to illustrate the assembly of $\mathcal{A}_{\mathcal{G}}$, $\mathcal{B}_{\mathcal{G}}$, and C using the example diagram shown in Fig. 6.2. If we are to assemble node index i of $\mathcal{A}_{\mathcal{G}}$, then we would effectively sum all the corresponding diagonal values of $\mathcal{A}_a$, $\mathcal{A}_b$, ... up to $\mathcal{A}_f$, which results in:

$$[C\mathcal{A}_{\mathcal{G}}]_i = \sum_{k\in\mathcal{N}(i)} [\mathcal{A}_k]_i \tag{6.23}$$

$$= c\alpha_i - \sum_{k\in\mathcal{N}(i)} \alpha_{ik} \tag{6.24}$$

$$= l\alpha_i - \alpha_i, \text{ substituting (5.12)} \tag{6.25}$$

$$= (l-1)\alpha_i \tag{6.26}$$

where $l = 6$ is the number of FEM-FG links around node i. Similarly, we can perform the assembly for $\mathcal{B}_{\mathcal{G}}$.

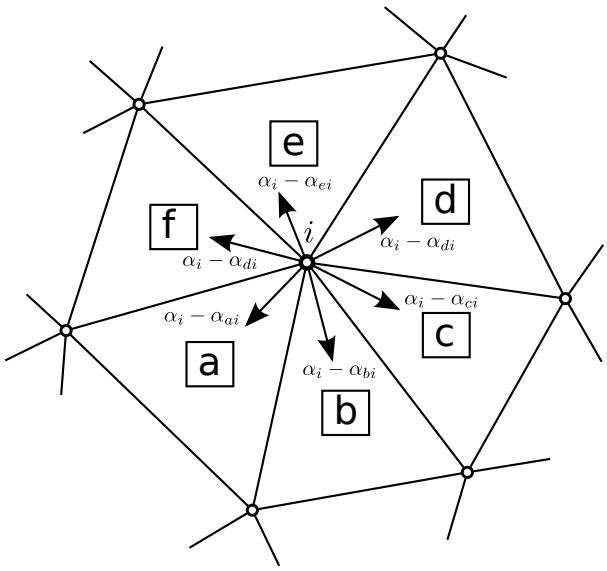


Fig. 6.2 The outgoing messages around an interior node i in FEM-FG.

Now substituting (6.21) and (6.22) into (6.20) we obtain:

$$r_{\mathcal{G}} = f - A\bar{\mu} + C\mathcal{B}_{\mathcal{G}} - C\mathcal{A}_{\mathcal{G}}\bar{\mu}. \tag{6.27}$$

Noting that $C\mathcal{A}_{\mathcal{G}}\bar{\mu} = C\mathcal{B}_{\mathcal{G}}$, since $[\bar{\mu}]_i = \beta_i/\alpha_i \ \forall i$, then (6.27) reduces to:

$$r_{\mathcal{G}} = f - A\bar{\mu} + C\mathcal{B}_{\mathcal{G}} - C\mathcal{B}_{\mathcal{G}} \tag{6.28}$$

$$= f - A\bar{\mu}. \tag{6.29}$$

This result illustrates that minimizing the local belief residuals is equivalent to minimizing the global residual for the assembled linear system $Au = f$, albeit computed in a distributed way without needing to assemble the sparse matrix A. The result also shows that the local distributed computations performed by the FMGaBP algorithm amounts to solving the same problem from global perspectives. The reader at this point might raise the question whether it will be possible to decompose the global residual into local quantities. In fact, such a decomposition would be hard; since without any prior knowledge of the local messages per each iteration, the decomposition would not be unique.

6.2.6 The FMGaBP Fixed-Point

Clearly from (6.11), once all messages converge on the fine grid, that is once $\bar{\mu}_a^h \rightarrow [u_0]_a$ where $[u_0]_a$ is the true solution vector of the local nodes of a, then $r_a \rightarrow 0$. Furthermore, by uniqueness of the FEM solution on coarser grids, at stationarity the β messages on each subsequent coarse grid will also be zero, hence the correction $\bar{u}_a^H$ from the coarser grids will consequently be zero. Therefore, the FMGaBP is a fixed-point algorithm for the true solution on the fine grid.

6.2.7 The FMGaBP Algorithm Listing

Algorithm 9 illustrates a high-level listing of the FMGaBP algorithm. The FMGaBP algorithm can be seen as executing instances of the FGaBP algorithm on each level in the multigrid hierarchy. The loops on lines 9 and 17 traverse all the levels on the V-cycle except the coarsest level, first going down the cycle and then up the cycle. Each level visited, the FGaBP executes v_1 iterations down the cycle and v_2 iterations up the cycle. It was found that $v_1 = v_2 = 1$ is typically sufficient in most cases. We iterate on the coarsest level using the FGaBP algorithm, that is in contrast to conventional multigrid methods where a direct solver is used on the coarsest level. Since the coarsest level contains a very small number of factors, within the order of few hundreds, its execution can be quite fast. Also as can be noted on Line 16, there was no need to use a low tolerance on the coarsest level. In many cases also, a tolerance of 10^{-6} was found to be sufficient on the coarsest level. One important note to make is that the FMGaBP algorithm can solve the problem down to machine floating-point precision, as can be noted by the low tolerance for convergence on Line 21.

6.3 Implementation

6.3.1 Data-Structures

The FMGaBP data-structures are mostly based on the FGaBP data-structures with the addition of another dense matrix per multigrid level. The added matrix stores the index associations of parent-child FNs for each hierarchical pair of coarse-fine levels. The total size

Algorithm 9 The FMGaBP algorithm.

1: Obtain FEM hierarchical mesh
2: **loop** {For each level in the mesh hierarchy}
3: Perform partition-color
4: Generate element matrices M_s and source vectors B_s
5: Incorporate boundary conditions
6: *Initialize:* $\alpha_{ij} = 1,\ \beta_{ij} = 0,\ \forall i, j$
7: **end loop**
8: **repeat** {FMGaBP V-cycle iteration: $t = 1, 2, \cdots$}
9: **loop** {Traverse levels down the V-cycle}
10: **if** Coarsest level reached **then**
11: Exit
12: **end if**
13: Execute v_1 iterations of FGaBP (Algorithm 8)
14: Restrict residuals from child-factors into parent-factors
15: **end loop**
16: Execute FGaBP on coarsest level for tolerance $< 10^{-9}$
17: **loop** {Traverse levels up the V-cycle}
18: Prolongate corrections from parent-factors into child-factors
19: Execute v_2 iterations of FGaBP (Algorithm 8)
20: **end loop**
21: **until** Convergence check on finest level messages for tolerance $< 10^{-16}$
22: *Output:* $\mu_i = \frac{\beta_i}{\alpha_i}$

of the FMGaBP data-structure can be obtained by:

$$\text{FMGaBP Memory} \approx O\left[(Z + cN_f)\sum_{l=0}^{L-1}\frac{1}{c^l} - cN_f\right] \tag{6.30}$$

$$= O\left[(Z + cN_f)\frac{1-(1/c)^L}{1-(1/c)} - cN_f\right] \tag{6.31}$$

where l is the level index; L is the total number of levels; $Z = 2N_v + N_f(n^2 + 4n)$ which is the FGaBP memory on the finest level as detailed in Section 5.7.1; and c is the number of children, e.g. $c = 4$ for 2D quadrilateral meshes or $c = 8$ for 3D hexahedral meshes. Clearly, the overall memory complexity is linear in N_v as $L \to \infty$.

6.3.2 Multicore CPU Implementation

Similar to the FGaBP, the FMGaBP code was designed using C++ OOP [114, 115]. This facilities the integration of the FGaBP code as a level solver into the FMGaBP with minimal recoding. In addition, different FEM open-source libraries, such as deal.II [27] and GetFEM++ [28], can be integrated easily which enables thorough testing of the FMGaBP algorithm. The library GetFEM++ was used to test our code using irregular triangular meshes that are generated by Gmsh [113], while deal.II was used to test our code on quadrilateral and hexahedral meshes. However out of the two, only deal.II provides parallel implementation of its solver and therefore it will be used for our performance analysis. While there are other high quality C++ FEM libraries such as Dune [119, 119] and libMesh [120], deal.II and GetFEM++ were the most up-to-date with extensive documentation and active support teams.

Another advantage of using OOP style coding is that the FMGaBP implementation performance can be tested using different Basic Linear Algebra (BLAS) libraries and dense solvers [105, 121, 122]. These libraries are needed by the FNs for the efficient execution of its dense operations, such as the Cholesky inversion or the GS iteration required by the AU-FGaBP scheme. For that purpose, we used different dense routines from the libraries deal.II, gmm [93], and Eigen [104]. Eigen, however, seemed to produce the best overall results.

Similar to the FGaBP code, the FMGaBP code was parallelized using OpenMP [116]. Each thread is assigned to handle the transfer operations of each parent-child FNs set. The transfer operations are fully decoupled; and hence, there was no need for the use of elaborate partitioning schemes such as coloring in order to ensure thread-safety. A single thread synchronization directive is used at the end of each transfer operation either restriction or prolongation. Finally, the output solution was visualized using Paraview [123].

6.3.3 Manycore GPU Implementation

In the past decade, architectures with many (tens or hundreds of) integrated cores have been introduced and used in the scientific computing community. GPUs are a popular class of manycore processors offering greater opportunities of parallelism on a single chip than multicore CPUs. It is expected that adeptly parallel algorithms such as the FMGaBP can benefit from the increased parallelism of GPU architectures. In this section, we will detail the implementation techniques used to evaluate the FMGaBP performance on GPU architectures.

Background on GPUs

Attempts to port parts or all of the FEM computations to manycore architectures have been presented in previous works. Most of these works are aimed at accelerating either the global sparse matrix assembly or the global system solve stage. Constrained by the special characteristics of their applications, the works in [124,125] present simple assembly strategies suited for manycore architectures. Their proposed techniques are not applicable for general FEM applications but, rather, are mostly suited to the sparsity structure of their specific applications. Graph coloring schemes in [126–128] are used to partition the FEM elements to non-overlapping sets, in order to enable a parallel thread-safe execution of the assembly routines for the global sparse matrix. Assembling a global sparse matrix, to be solved in a separate stage, based on graph coloring and partitioning can be costly. For the FGaBP algorithm, coloring adds little overhead since the global sparse matrix is never assembled and coloring is used, rather, to compute the FEM solution in parallel. Other work [129–131] propose strategies and novel sparse storage formats to reduce the memory footprint of the assembled sparse matrix. Since, FGaBP eliminates the need for assembling a large sparse

matrix, many of the optimizations proposed in previous work are, thus, not applicable to our work.

Significant research has been aimed at improving the solve stage for sparse systems using GPUs. Most these works focus on accelerating the execution of the compute intensive kernels in the Krylov solvers. As shown in [19,25] such efforts are mainly communication bound and are limited by the maximum performance achieved from parallelizing the compute intensive kernels. The assembled matrix is usually large and sparse and in many cases does not fit in the small and fast access memory levels of CPU and the co-processor memory.

FGaBP avoids assembling a global sparse matrix and, thus, is a promising candidate for manycore architectures. Instead of solving a large sparse matrix, each FGaBP iteration needs to compute the incoming and outgoing messages for each factor node. The compute intensive kernel in a FGaBP iteration involves computing the inverses of many small dense matrices; an operation that is embarrassingly parallel and well suited for manycore architectures. Coloring the FEM-FG models improves the underlying parallelism significantly and allows FNs of the same color to be processed in parallel by hundreds of threads without causing any synchronization or memory collisions when accessing the VN data. To demonstrate the embarrassingly parallel nature of both the FGaBP and FMGaBP algorthims, we have implemented them on the NVIDIA Tesla C2075 GPU.

The GPU architecture

The NVIDIA Tesla C2075 GPU, which belongs to the Fermi generation, is used to illustrate the performance of the FMGaBP implementation on manycore architectures. The Tesla C2075 has a 6 GB DRAM memory, 448 CUDA cores, 48 KB of shared memory, and a memory bandwidth of 144 GB/s.

Current NVIDIA GPUs consist of Streaming Multiprocessors (SMs) each containing a

few Scalar Processor Cores (SPs), an on-chip shared memory, a data cache and a register file. Threads executing on each SM have access to the same shared memory and register file, which enables communication and data synchronization with little overhead. Threads executing on different SMs can only synchronize through the GPUs off-chip global Dynamic Random-Access Memory (DRAM) which incurs high latency of several hundred clock cycles. To parallelize an algorithm on GPUs, all of these architectural features have to be taken into account to efficiently use the available GPU resources.

The most popular APIs used to implement algorithms on GPUs are the Compute Unified Device Architecture (CUDA) and the Open Computing Language (OpenCL). CUDA 5.0 was used along with the library CUBLAS 5.0 [132] to implement the FMGaBP algorithm on the GPU. Initially, data has to be explicitly transferred from the CPU memory to GPU followed by the instantiation of a collection of kernels executing the parallel segments of the program on the GPU. Threads are grouped into blocks that are scheduled for execution on the GPU's SM. Groups of 32 threads in a block, called warps, are scheduled to execute the same instruction at the same time. Threads in the same block can communicate via on-chip shared memory with little latency while threads in different blocks have to go through the GPU global memory for any kind of data synchronization [132].

GPU implementation details

The FMGaBP algorithm is fully implemented on the NVIDIA Tesla C2075. The FNs, VNs, and level matrix data are transferred to the GPU once, thus no GPU-CPU memory transfers are required during the algorithm's execution. The following details the GPU implementation of the FMGaBP algorithm:

Multigrid restriction and prolongation kernels: The restriction and prolongation stages are implemented in two different kernels. The parent-child mappings in the FMGaBP

are loaded into shared memory to reduce global memory references. The compute intensive operation in the multigrid computations is the dense matrix vector multiply for each parent FN in the coarser grid. The number of parent FNs assigned to each warp is computed by dividing the number of threads per warp (32) by the number of children for each parent. For example, in a 2D problem using quadrilateral meshes, each warp in the interpolation kernel applies corrections from eight FNs in the coarse grid to their children; thus allocating four threads to each FN in the coarse grid to parallelize the compute intensive kernels involved in the restrict operations.

Batch of inverses on GPUs: Computing the inverse of local matrices in the smoother iterations is the most time consuming operation in the FMGaBP algorithm. Depending on the problem size, the number of matrices to be inverted can be very large. Various heuristics could be used to compute a batch of inverses on the GPU. Depending on the size of the local matrices, each inverse could be computed via one thread block, one warp or even one thread. For example, if the rank of each matrix is 256 then allocating one thread block (with 256 threads) to each matrix inverse would be efficient.

A batch of inverses can be computed using the NVIDIA's CUBLAS library [132] for matrices up to rank 32. An in-place LU decomposition should first be performed and then the cublasDgetriBatched kernel is called to compute an out-of-place batch inversion. Since each warp computes the inverse of one matrix, the aforementioned kernel does not perform well for the low rank matrices in the FMGaBP kernel. For 2D problems using quadrilateral meshes or 3D problems using tetrahedral meshes, our matrices are only of rank 4, thus when using the CUBLAS matrix inversion, the GPU resource will be underutilized and threads in a warp will not have enough work. Our matrix inversion kernel is customized to the matrix's dimension. The number of inverses computed via one warp is obtained through dividing the number of threads per warp (32) by the rank of the local dense matrices. For example,

for a 2D problem with rank 4 local matrices, each warp computes 8 matrix inversions. We outperform the CUBLAS matrix inversion kernel by 2× when inverting a batch of 10 million rank 4 matrices. Another major advantage of our matrix inverse kernel is that it performs the inverse in-place on shared memory. As a result, the computed inverses do not have to be stored in global memory and the outgoing messages can be computed in the same kernel. Not storing the matrix inverses in the global memory enables the GPU to solve larger problems more readily.

Kernel fusion in FGaBP: The FGaBP iterations involve computing the incoming and outgoing messages and updating the VN's running sums. Instead of calling three separate GPU kernels one for each stage, we fuse these kernels and only call one GPU kernel for each iteration. Key advantages resulting from the fusion process are: First, data can be loaded into shared memory in order to be used by a single FGaBP kernel call reducing communication within the GPU's memory hierarchy. Second, the local matrix inverses can be generated on the fly and used to compute the running sum without the need to be stored in global memory. Lastly, kernel call overheads are also reduced by only calling one kernel for each FGaBP iteration.

6.4 The FMGaBP Numerical Results and Discussions

The performance of the FMGaBP algorithm is demonstrated using two Laplace benchmark problems as shown in Fig. 5.6. The first problem is the SSC using two different material properties as shown in Fig. 5.6(a). The second problem is the L-shaped corner of the Square Coaxial Conductor (LSC) as shown in Fig. 5.6(b). Both problems employ Dirichlet and Neumann boundary conditions.

The SSC problem was formulated using the library GetFEM++ [28] with semi-irregular

triangular meshes. The LSC problem was formulated using the library deal.II [27] with hierarchical quadrilateral meshes. Since the library deal.II inherently supports hierarchical meshes and multigrid solvers using parallel implementations, it was used for performance comparisons. A V-cycle multigrid scheme is used in all our experiments. All solvers are terminated when the residual l^2-norm is dropped to 10^{-15}. The FMGaBP algorithm uses the iterative FGaBP as the coarsest level solver; whereas, a Householder direct solver was used by deal.II on the coarse level for the MG-PCG solver.

6.4.1 Semi-irregular Mesh Hierarchy

Table 6.1 shows the FMGaBP convergence results for the SSC problem. The results demonstrate a convergence performance almost independent of the number of unknowns in the finest level. However, a trend of slightly increasing number of iterations is observed. This increase is expected as a result of the strongly varying material coefficients (20:1) as shown in Fig. 5.6(a).

Table 6.1 The FMGaBP algorithm performance for the SSC problem using five levels of refinement.

Refinement Level	Unknowns	Triangles	*Iterations* $v_1 = 1,\ v_2 = 1$ *
1-coarse	222	382	—
2	825	1,528	*9*
3	3,177	6,112	*11*
4	12,465	24,448	*13*
5	49,377	97,792	*15*
6	196,545	391,168	*16*

* The parameters v_1 and v_2 are pre and post V-cycle iterations.

6.4.2 Structured Mesh Hierarchy

Table 6.2 shows the FMGaBP results for the LSC problem compared with the results from deal.II. The basic FGaBP algorithm was used as the level solver for the FMGaBP algorithm. The library deal.II implements the MG-PCG solver using multithreading. The symbols t_{st} and t_{sv} refer to the setup and solve time respectively reported in seconds; the acronym "itr" refers to iterations. The timing results were obtained using a compute node from the supercomputing cluster Colesse [133]. The computing node contains two quadcore Xeon Nehalem CPUs @ 2.8 GHz with 8 MB cache and 48 GB DRAM. Problem sizes up to to 12.6 million unknowns are solved. A coloring algorithm is used to guarantee thread safety. The coloring algorithm required minimal overhead, by virtue of utilizing the hierarchical structure of the mesh provided by deal.II. For all timing cases, the best of forty runs is reported.

Table 6.2 The FMGaBP speedup over deal.II on a 2D domain using 8-threads on 2×quadcore processors.

Unknowns (million)	Quadrilaterals (million)	FMGaBP			MG-PCG			*Speedup*	
		itr	t_{st}	t_{sv}	itr	t_{st}	t_{sv}	*setup*	*solve*
0.788	0.786	6	2.6	0.98	10	6.14	2.56	*2.4*	*2.6*
3.15	3.15	6	10.5	3.74	10	27.5	10.4	*2.6*	*2.8*
12.6	12.6	6	43.1	14.2	10	108	41.6	*2.5*	*2.9*

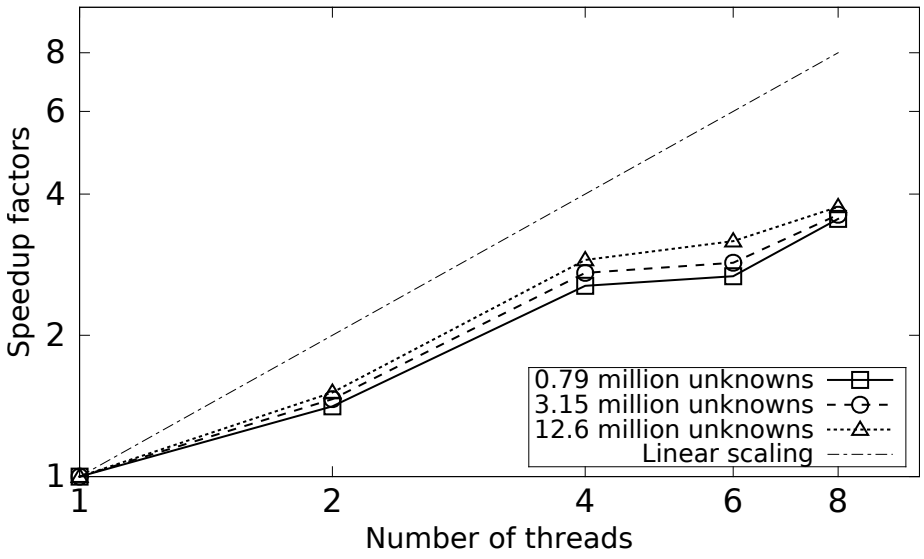
As shown in Table 6.2, the FMGaBP algorithm required a considerably lower number of iterations than the MG-PCG solver on this particular problem. In addition, the FMGaBP algorithm demonstrates considerable time speedups for both the setup time and the solve time. Since setup operations are not parallelized in deal.II at the time of using the library, sequential execution time is only reported for the setup phase. The major reductions in setup time was due to the fact that the FMGaBP, unlike to the library deal.II, does not require the

setup of globally large sparse matrices, or level transfer matrices. The FMGaBP algorithm also demonstrated parallel solve time speedups of up to 2.9 times over deal.II's MG-PCG solver. The speedups were due to mainly two factors, the considerable iteration reductions and the higher parallel efficiency of FMGaBP. As a key factor, the FMGaBP implements the level transfer operations using embarrassingly parallel local stencil-like operations.

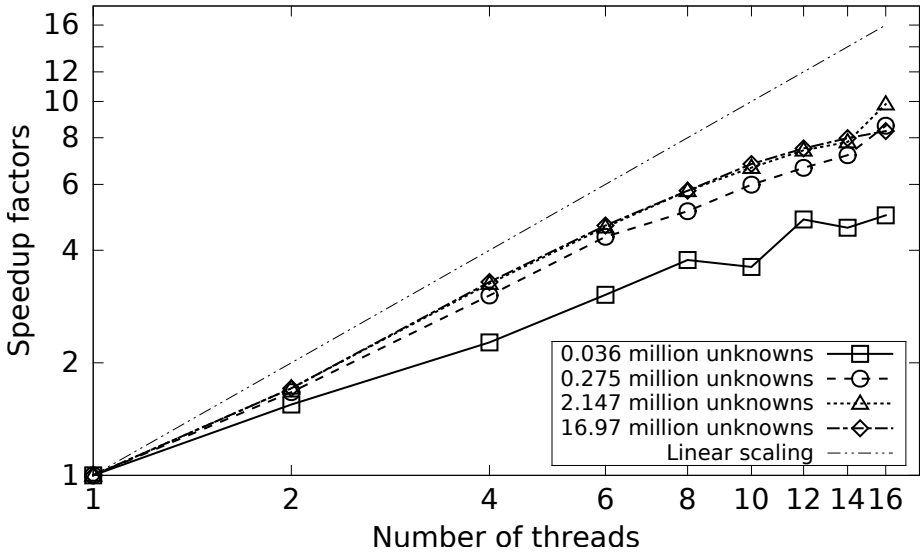
Table 6.3 shows the FMGaBP results for the Helmhotlz problem on a 3D domain as introduced in Section 5.8.1. The AU-FGaBP algorithm was used as the level solver for the FMGaBP algorithm configured with 2 GS iterations and 10^{-1} tolerance for the α messages. The FMGaBP is compared with the MG-PCG solver implemented with multithreading by deal.II. The similarity in the iteration counts between the two solvers is simply coincidental. The timing results were obtained using a compute node from the supercomputing cluster Sandybride on SciNet [134]. The node contains 2×8-core Xeon CPUs @ 2.5 GHz with 8 MB cache and 64 GB DRAM. Problem sizes up to 16.97 million unknowns are solved. A coloring algorithm is used to guarantee thread safety. For all timing cases, the best of forty runs is reported.

The FMGaBP algorithm demonstrated similar timing speedups over the multithreaded implementation of MG-PCG by the library deal.II. These speedups are attributable to the FMGaBP's efficient utilization of parallel CPU's resources. While the library deal.II requires the buildup of globally shared sparse-data structures for each multigrid level, the FMGaBP algorithm uses dense data-structures. This also explains the speedups obtained by the setup time which can be considered high compared to the solve time.

Fig. 6.3 shows the speedup factors of FMGaBP for the different problem sizes on 2D and 3D domains. FMGaBP demonstrates approximately linear trends of speedups with increasing number of threads. In addition, the graph trends show increased parallel efficiency as the problem size increases.



(a)



(b)

Fig. 6.3 (a) The FMGaBP speedup factors for the 2D SSC problem on 2×quadcore processors node. (b) The FMGaBP speedup factors for the 3D Helmholtz problem on 2×8-core processors node.

Table 6.3 The FMGaBP speedup over deal.II on a 3D domain using 24-threads on 2×6-core processors.

Unknowns (million)	Quadrilaterals (million)	FMGaBP			MG-PCG			*Speedup*	
		itr	t_{st}	t_{sv}	itr	t_{st}	t_{sv}	*setup*	*solve*
0.036	0.033	8	0.54	0.15	8	0.90	0.17	*1.67*	*1.13*
0.275	0.262	8	3.19	0.78	8	7.48	1.51	*2.34*	*1.94*
2.147	2.097	8	25.0	5.56	8	62.2	11.5	*2.5*	*2.07*
16.97	16.78	8	212.	52.9	– *	–	–	–	–

* deal.II crashed for the experiment with 16.97 million unknowns.

To address problems of larger scale, the use of multi-node clusters is required. For such implementations, the MPI programming model is typically used along with sophisticated partitioning algorithms. Implementations of FMGaBP using MPI are proposed as a future research direction.

6.4.3 Manycore GPU Performance

The FMGaBP is implemented on an NVIDIA Tesla C2075 for the 2D Helmhotlz problem, as previously defined in Section 5.8.1, with the number of unknowns ranging from 26K to 4.1M. Larger problems should be executed on a cluster of GPUs because of the GPU's global memory size limits. Fig. 6.4 shows the speedup achieved by implementing FMGaBP on a single GPU versus the proposed parallel CPU implementation of the method on the SciNet Sandybride cluster node [134]. The SciNet node contains 2×8-core Xeon 2.0 GHz CPUs with 64 GB DRAM. The speedup scalability is also presented in the figure by altering the number of threads for the CPU runs. As shown in the figure, the Tesla C2075 outperforms the CPU with up to 12 cores for all problem sizes. Larger problems are able to utilize the GPU resources more efficiently thus the GPU is faster than the 16-core CPU node for the largest problem with 4.1M unknowns. The only case where the GPU did not demonstrate

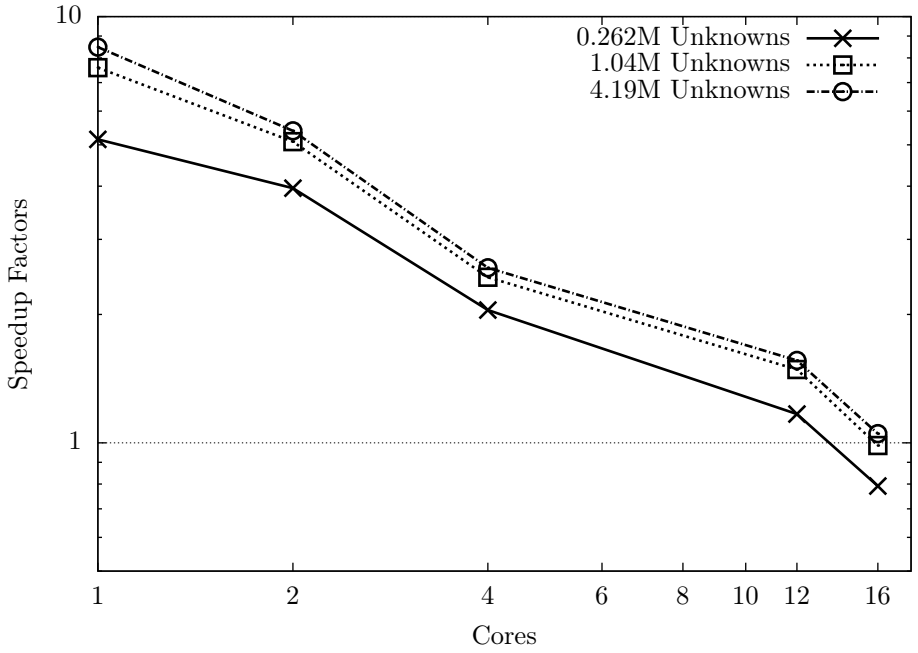


Fig. 6.4 The speedup achieved from accelerating FMGaBP on NVIDIA Tesla C2075 compared to the parallel implementation of the method on 1-16 CPU cores.

speedups were for the smaller problem sizes (26K and 1M unknowns). The average (speedup over all problem sizes) achieved from the GPU implementation compared to the dual-core, quad-core and 12-core CPU settings are 4.8×, 2.3× and 1.5× respectively.

6.5 Conclusion

The novel FMGaBP algorithm was introduced and was shown to achieve high parallel performance due to its matrix-free approach and localized computations. In addition, we showed for a benchmark Laplace problem that the FMGaBP algorithm requires less iterations than the MG-PCG solver. The threaded multicore implementation of FMGaBP demonstrates speedups of up to 2.9× over deal.II's implementation of MG-PCG. The GPU implementation showed considerable speedups over the multithreaded CPU implementation.

CHAPTER 7

Future Directions

The FGaBP and the FMGaBP algorithms are newly developed solvers that were implemented with only one purpose in mind, which is to efficiently accelerate the FEM problem and demonstrate high parallel scalability. Therefore, there may be many possibilities to advance the new algorithms just by looking at the wealth of literature developed in the field of numerical methods in the past decades. One would find a multitude of methods used to improve the efficiency and the robustness of conventional solvers which could also be applicable here. In this chapter we survey some of these ideas and discuss their applicability to the new algorithms, as well as the challenges that may be encountered.

7.1 Krylov-Subspace Method Preconditioners

For certain applications, the FMGaBP algorithm can be very efficient in achieving high convergence; however, many solvers including multigrid can be used as preconditioners in order to improve their robustness especially for complicated applications. In using the FMGaBP or the FGaBP as a preconditioner, one would be interested in obtaining a robust and efficient convergence rate and, most importantly, increasing the parallel scalability of the conventional

Krylov-Subspace Methods (KSMs). By observing Algorithm 1, which is presenting the PCG solver, we can see that the preconditioner step can be implemented by either the FGaBP or the FMGaBP. If the FMGaBP is used, the fast convergence rate should be expected. In each PCG iteration, the residual is used as the right hand side of the preconditioner system. This can be accomplished by distributing the global residual vector among the FNs as local source vectors. It is important to note here that this distribution of the residual vector to the FNs may not be unique. For example, one choice is to consider to distribute the residual vector elements evenly to all FNs connected to that residual element. Another approach would be to use the α messages as weighting factors, since after each consistent FGaBP iteration the sum of each VN edge α messages is equal to the nodal α sum as expressed by the update rule (5.12). The β messages are then reinitialized from zero for each PCG iteration. Note that the α messages need not be reinitialized each PCG iteration since they do not depend on the right hand side, or the FNs source vectors, and their computation can progressively advance each PCG iteration without being restarted. The FGaBP, or the FMGaBP, will then be allowed to iterate a number of times based on the desired preconditioning recipe.

A difficulty may arise in how to compute the SMVM operation efficiently as in Step 2 of Algorithm 1. In this setting we can still avoid assembling the sparse matrix A. The effect of the SMVM operation can be obtained by operating on the FGaBPs data-structure directly. In other words, we reuse the FGaBP data-structure for both the SMVMs and the precondition operations. The unknown vector would be distributed to the FNs, and then each FN will perform a local dense matrix vector multiplications using its local characteristic matrix and source vector. The local results are then gathered using the local to global index information in order to produce the global residual vector. This SMVM parallel operation can be performed in a thread-safe manner when element coloring is used.

7.2 FMGaBP for Adaptive Refinement

The FMGaBP algorithm demonstrated high computational parallel efficiency compared to state-of-the-art solvers; however, the FMGaBP algorithm was developed based on global mesh refinement, which can result in unnecessary computations when the local FEM solution accuracy is considered. The FMGaBP formulation can be extended to supporting adaptive FEMs refinement by exploiting the local information already present within the FNs. The resulting adaptive FMGaBP algorithm would exhibits distributed computations and high parallel efficiency for increasing FEM solution accuracy.

If the adaptive mesh refinement scheme produces conformal meshes, then the FMGaBP algorithm can readily be used. However, if the adaptive mesh refinement scheme produces level meshes that contain hanging-nodes, then the FMGaBP algorithm would require some reformulation to address this case. This scheme of adaptive meshing results in special nodes located at the interface between the refined and the non-refined patches of the mesh. Such nodes are referred to as hanging nodes. Hanging nodes are termed so because they belong to one side of the mesh, which is the refined side. In other words, they have the associated basis functions on one side and have none on the other. The library deal.II uses adaptive mesh refinement with hanging-nodes for quadrilateral and hexahedral meshes. The approach used by deal.II involves introducing constraints that ensures the FEMs solution's continuity across the hanging nodes [135]. The FMGaBP can incorporate similar constraints by applying them on a per FN basis similar to the defined local belief system as in (6.2).

7.3 FMGaBP MPI Implementation

In Chapter 6 the FMGaBP algorithm was scaled for large FEM problems where the number of elements reaches millions, which is limited only by the memory capacity of the multicore

node. However, for very large scale FEM simulations involving billions of unknowns the use of clusters of CPU nodes connected by a network is needed. Each node in the cluster contains a multicore CPU and its own memory space which can be accessed by each of the CPU cores as shared memory. Accessing data from one node to another however, has to go through the network which incurs high latency delays. The FMGaBP algorithm, due to its high parallel efficiency, demonstrated great performance when executed within one multicore node. It is also expected that the FMGaBP can demonstrate even more competitive performance when executed on cluster HPC systems, given that a good partitioning algorithm is used to distribute the FEMs problem across the nodes efficiently.

Cluster HPC systems are very scalable and can handle FMGaBP problems in the order of billions of unknowns. Obtaining linear scaling on these systems however is very challenging due to its dependence on problem partitioning schemes that impacts memory communication and load-balancing. Specialized open-source libraries such as ParMETIS [107] and P4EST [136] can be used to efficiently partition the problem. These libraries can be integrated within the FGaBP algorithm in order to enhance its message communication pattern. The FGaBP software can utilize a hybrid model of multithreading and MPI [65] in order to target the cluster of multicore CPUs. The multithreading features will enable the FGaBP software to efficiently exploit parallelism within the multicore node while the MPI interface will exploit parallelism between different nodes.

APPENDIX A

Gaussian Probability Distribution Functions

A.1 Univariate Gaussian Distribution

A random variable u that assumes values u_o drawn from a Gaussian distribution $G(\mu, \sigma)$ is referred to as a Gaussian random variable and is denoted as $u \sim G(\mu, \sigma)$. The univariate Gaussian distribution $G(\mu, \sigma)$, also referred to as the normal distribution, is defined as:

$$G(u\,;\,\mu,\sigma) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}(u-\mu)^2\right] \tag{A.1}$$

where μ and σ^2 are referred as the mean and the variance of the Gaussian distribution correspondingly, while σ is referred to as the standard deviation. The term $1/\sqrt{2\pi\sigma^2}$ is the normalization constant that ensures the property $\int_{-\infty}^{\infty} G(u\,;\,\mu,\sigma)\,\mathrm{d}u = 1$. As a result, the Gaussian distributions, in their basic form, are fully characterized by their mean and variance parameters. Fig. A.1 illustrates different Gaussian distribution plots for different μ and σ parameters. Notice that the Gaussian distribution has a maximum value at its mean point $u = \mu$.

The Gaussian distribution used in the derivation of the FGaBP algorithm in Section 5

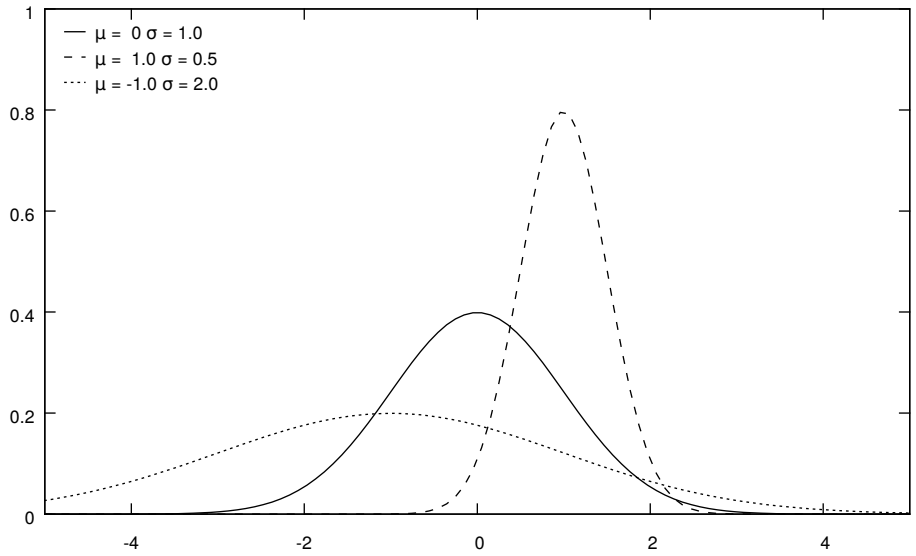


Fig. A.1 Sample univariate Gaussian distribution plots for different μ and σ values.

can more conveniently be represented using a parameterization referred to as the canonical parameterization, or the information form. The Gaussian distribution can also be used in an un-normalized form by ignoring the normalizing constant. This does not constitute any loss of information since the Gaussian distribution is fully identified by its mean and variance parameters or, correspondingly, the canonical β and α parameters. The un-normalized canonical representation of the univariate Gaussian distribution is defined as:

$$G(u\,;\,\beta,\alpha) \propto \exp\left[-\frac{1}{2}\alpha u^2 + \beta u\right] \tag{A.2}$$

where $\alpha = 1/\sigma^2$ is the reciprocal of the variance, also referred to as the precision; and $\beta = \mu/\sigma^2$ is the second canonical parameter.

A.2 Multivariate Gaussian Distribution

A collection of n random variables $U = (u_1, u_2, \cdots, u_n)$ have a joint Gaussian probability distribution ($U \sim \mathcal{G}(\bar{\mu}, \Sigma)$) defined as:

$$\mathcal{G}(U\,;\,\bar{\mu}, \Sigma) \triangleq \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(U-\bar{\mu})^T\Sigma^{-1}(U-\bar{\mu})\right] \tag{A.3}$$

where $\bar{\mu}$ is the mean vector of size n for the joint Gaussian random variables U; Σ is the covariance matrix; and $|\cdot|$ is the determinant operator. The covariance matrix Σ must be Symmetric Positive Definite (SPD) in order for $\mathcal{G}$ to be a valid Gaussian probability distribution. Fig. A.2 illustrates a sample bivariate Gaussian distribution plot with $\bar{\mu} = (0.5, 0.5)$ and $\Sigma = I$, where I is the identity matrix. A key property of Gaussian distributions is that they hold their maximum value at $U = \bar{\mu}$.

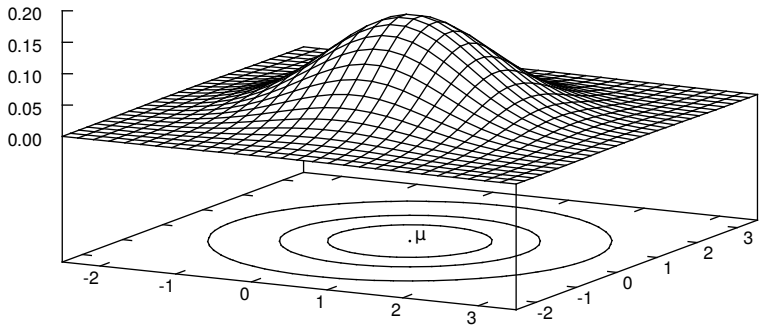


Fig. A.2 Sample bivariate Gaussian distribution plot with $\bar{\mu} = (0.5, 0.5)$ and $\Sigma = I$.

Similar to the univariate case, the multivariate Gaussian distribution can be represented in the un-normalized canonical form as:

$$\mathcal{G}(U\,;\,\bar{\mu}, \Sigma) \propto \exp\left[-\frac{1}{2}U^TWU + K^TU\right] \tag{A.4}$$

where $W = \Sigma^{-1}$ is the inverse covariance matrix, also referred to as the precision matrix;

and $K = \Sigma^{-1}\bar{\mu}$ is the scaled mean parameter.

The original form of the Gaussian distribution in (A.1) and (A.3) can be obtained from (A.2) and (A.4) correspondingly by completing the algebraic square of the exponent and adjusting the normalizing constant accordingly. The canonical form is used in the FGaBP algorithm in order to produce more convenient formulation when multiplying and marginalizing Gaussian distributions as will be illustrated below.

A.3 Multiplication of Gaussian Distributions

Multiplication of any number of Gaussian distributions results in a Gaussian distribution up to a constant. Using un-normalized canonical forms, the resulting distribution has canonical parameters equal to the sum of the canonical parameters of the multiplied distributions. The multiplication of M multivariate Gaussian distributions that represent the same Gaussian random variable vector U can be formulated as:

$$\mathcal{G}(U\,;\,W,K) \propto \prod_{m=1}^{M} \mathcal{G}_m(U\,;\,W_m,K_m) \tag{A.5}$$

the resulting distribution $\mathcal{G}$ is Gaussian with canonical parameters computed by:

$$W = \sum_{m=1}^{M} W_m \tag{A.6}$$

$$K = \sum_{m=1}^{M} K_m. \tag{A.7}$$

This result can readily be applied to scalar Gaussian distributions by replacing the multidimensional parameters with their scalar counterparts.

A.4 Marginalization of Gaussian Distributions

Marginalization of a multivariate Gaussian distribution is integrating away a subset of variables, sometimes referred to as nuisance variables, so that the resulting distribution is a function of the variables of interest. An important property of Gaussian distributions is that any marginalization of a Gaussian distribution results in a Gaussian distribution. To further illustrate, suppose that the Gaussian variable vector U is partitioned into two subsets labeled a and b such that $U = (U_a, U_b)$. Then, to marginalize the Gaussian distribution for U_a, we formulate the following:

$$\mathcal{G}(U_a\,;\,\bar{W}_a, \bar{K}_a) \propto \int_{U_b} \mathcal{G}(U\,;\,W, K)\,\mathrm{d}U_b. \tag{A.8}$$

To evaluate this integral, we first partition W and K according to the U_a and U_b partitions as follows:

$$W = \begin{bmatrix} W_{aa} & W_{ab} \\ W_{ba} & W_{bb} \end{bmatrix}, \quad K = \begin{bmatrix} K_a \\ K_b \end{bmatrix}. \tag{A.9}$$

We then manipulate (A.8) by separating the U_a and U_b variables, completing the algebraic square, and using the Gaussian integral identity stated as follows:

$$\int_U \exp\left[-\frac{1}{2}(U-\mu)^T\Sigma^{-1}(U-\mu)\right]\,\mathrm{d}U = (2\pi)^{n/2}|\Sigma|^{1/2}. \tag{A.10}$$

Finally the integral (A.8) is evaluated by finding the $\bar{W}_a$ and $\bar{K}_a$ parameters as follows:

$$\bar{W}_a = W_{aa} - W_{ab}W_{bb}^{-1}W_{ba} \tag{A.11}$$

$$\bar{K}_a = K_a - W_{ab}W_{bb}^{-1}K_b. \tag{A.12}$$

References

[1] J. Jin, *The Finite Element Method in Electromagnetics*. New York, USA: Wiley-IEEE Press, 2nd ed., 2002.

[2] Intel Corporation, "Intel® Xeon Phi™ product family." http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html, 2013.

[3] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelpha, PA: SIAM, 2nd ed., 2003.

[4] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," in *In Proc. ACM/IEEE Conf. Supercomputing, SC '07*, pp. 1–12, 2007.

[5] E.-J. Im, K. Yelick, and R. Vuduc, "Sparsity: Optimization framework for sparse matrix kernels," *International Journal of High Performance Computing Applications*, vol. 18, no. 1, pp. 135–158, 2004.

[6] R. Vuduc, J. Demmel, K. Yelick, S. Kamil, R. Nishtala, and B. Lee, "Performance optimizations and bounds for sparse matrix-vector multiply," in *ACM/IEEE 2002 Conference Supercomputing*, pp. 26–26, Nov. 2002.

[7] K. Kourtis, G. Goumas, and N. Koziris, "Optimizing sparse matrix-vector multiplication using index and value compression," in *Proceedings of the 5th conference on Computing frontiers*, pp. 87–96, ACM, 2008.

[8] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Philadelphia, PA: SIAM, 2nd ed., 1994.

[9] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson, "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks," in *Proceedings of the Twenty-first Annual Symposium on Parallelism in*

Algorithms and Architectures, SPAA '09, (New York, NY, USA), pp. 233–244, ACM, 2009.

[10] J. Mellor-Crummey and J. Garvin, "Optimizing sparse matrix–vector product computations using unroll and jam," *International Journal of High Performance Computing Applications*, vol. 18, no. 2, pp. 225–236, 2004.

[11] D. Fernandez, D. Giannacopoulos, and W. Gross, "Multicore acceleration of CG algorithms using blocked-pipeline-matching techniques," *IEEE Transactions on Magnetics*, vol. 46, pp. 3057–3060, Aug. 2010.

[12] J. Willcock and A. Lumsdaine, "Accelerating sparse matrix computations via data compression," in *Proceedings of the 20th annual international conference on Supercomputing*, pp. 307–316, ACM, 2006.

[13] R. Geus and S. Röllin, "Towards a fast parallel sparse matrix-vector multiplication.," in *PARCO*, pp. 308–315, Citeseer, 1999.

[14] Y. Shan, T. Wu, Y. Wang, B. Wang, Z. Wang, N. Xu, and H. Yang, "FPGA and GPU implementation of large scale SpMV," in *2010 IEEE 8th Symposium on Application Specific Processors (SASP)*, pp. 64–70, IEEE, 2010.

[15] S. Kestur, J. D. Davis, and E. S. Chung, "Towards a universal FPGA matrix-vector multiplication architecture," in *2012 IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 9–16, IEEE, 2012.

[16] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, p. 18, ACM, 2009.

[17] J. Sun, G. Peterson, and O. Storaasli, "Mapping sparse matrix-vector multiplication on FPGAS," in *Proceedings of Supercomputing*, vol. 99, 1999.

[18] L. Zhuo and V. K. Prasanna, "Sparse matrix-vector multiplication on FPGAs," in *In Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*, pp. 63–74, ACM Press, 2005.

[19] M. MehriDehnavi, Y. El-Kurdi, J. Demmel, and D. Giannacopoulos, "Communication-avoiding Krylov techniques on graphic processing units," *IEEE transactions on magnetics*, vol. 49, no. 5, pp. 1749–1752, 2013.

[20] M. Dehnavi, D. Fernandez, and D. Giannacopoulos, "Enhancing the performance of conjugate gradient solvers on graphic processing units," *IEEE Transactions on Magnetics*, vol. 47, pp. 1162–1165, May 2011.

[21] M. Dehnavi, D. Fernandez, and D. Giannacopoulos, "Finite-element sparse matrix vector multiplication on graphic processing units," *IEEE Transactions on Magnetics*, vol. 46, pp. 2982–2985, Aug. 2010.

[22] Y. El-Kurdi, D. Giannacopoulos, and W. Gross, "Hardware acceleration for finite-element electromagnetics: Efficient sparse matrix floating-point computations with fpgas," *IEEE Transactions on Magnetics*, vol. 43, no. 4, pp. 1525–1528, 2007.

[23] Y. El-Kurdi, W. Gross, and D. Giannacopoulos, "Sparse matrix-vector multiplication for finite element method matrices on FPGAs," in *FCCM '06. 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2006.*, pp. 293–294, April 2006.

[24] R. Vuduc, J. W. Demmel, and K. A. Yelick, "OSKI: A library of automatically tuned sparse matrix kernels," in *Proceedings of SciDAC 2005*, Journal of Physics: Conference Series, (San Francisco, CA, USA), Institute of Physics Publishing, June 2005.

[25] M. F. Hoemmen, *Communication-avoiding Krylov subspace methods*. PhD thesis, EECS Department, University of California, Berkeley, April 2010.

[26] D. Fernandez, M. Dehnavi, W. Gross, and D. Giannacopoulos, "Alternate parallel processing approach for FEM," *IEEE Transactions on Magnetics*, vol. 48, pp. 399–402, Feb. 2012.

[27] W. Bangerth, R. Hartmann, and G. Kanschat, "`deal.II` – a general purpose object oriented finite element library," *ACM Trans. Math. Softw.*, vol. 33, no. 4, pp. 24/1–24/27, 2007.

[28] Y. Renard and J. Pommier, "GetFEM++ – an open-source finite element library." `http://download.gna.org/getfem/html/homepage/`.

[29] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, "An overview of the trilinos project," *ACM Trans. Math. Softw.*, vol. 31, no. 3, pp. 397–423, 2005.

[30] M. Kronbichler and K. Kormann, "A generic interface for parallel cell-based finite element operator application," *Computers & Fluids*, vol. 63, no. 0, pp. 135 – 147, 2012.

[31] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[32] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Constructing free energy approximations and generalized belief propagation algorithms," *IEEE Trans. Inf. Theory*, vol. 51, pp. 2282–2312, 2004.

[33] Y. Weiss and W. T. Freeman, "Correctness of belief propagation in Gaussian graphical models of arbitrary topology," *Neural Computation*, vol. 13, no. 10, pp. 2173–2200, 2001.

[34] M. J. Wainwright, T. Jaakkola, and A. S. Willsky, "Tree-based reparameterization framework for analysis of sum-product and related algorithms," *IEEE Transactions on Information Theory*, vol. 49, no. 5, pp. 1120–1146, 2003.

[35] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, pp. 498–519, Feb. 2001.

[36] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends in Machine Learning*, vol. 1, pp. 1–305, January 2008.

[37] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 585–598, 2001.

[38] R. J. Mceliece, D. J. C. Mackay, and J. fu Cheng, "Turbo decoding as an instance of pearls belief propagation algorithm," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 140–152, 1998.

[39] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, 2001.

[40] W. Freeman and E. Pasztor, "Learning low-level vision," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, pp. 1182–1189 vol.2, 1999.

[41] J. Sun, N.-N. Zheng, and H.-Y. Shum, "Stereo matching using belief propagation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, pp. 787–800, July 2003.

[42] B. J. Frey, R. Koetter, and N. Petrovic, "Very loopy belief propagation for unwrapping phase images," in *Advances in Neural Information Processing Systems 14*, pp. 737–743, 2001.

[43] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1568–1583, 2006.

[44] T. Meltzer, C. Yanover, and Y. Weiss, "Globally optimal solutions for energy minimization in stereo vision using reweighted belief propagation.," in *International Conference on Computer Vision*, pp. 428–435, 2005.

[45] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods for markov random fields with smoothness-based priors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 6, pp. 1068–1080, 2008.

[46] E. B. Sudderth and W. T. Freeman, "Signal and image processing with belief propagation," *IEEE Signal Processing Magazine*, vol. 25, no. 2, p. 114, 2008.

[47] A. Mitrofanova, V. Pavlovic, and B. Mishra, "Integrative protein function transfer using factor graphs and heterogeneous data sources," in *Bioinformatics and Biomedicine, 2008. BIBM'08. IEEE International Conference on*, pp. 314–318, IEEE, 2008.

[48] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. Adaptive computation and machine learning, MIT Press, 2009.

[49] J. Yedidia, W. Freeman, and Y. Weiss, "Generalized belief propagation," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 13, pp. 689–695, Dec. 2000.

[50] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," *Exploring artificial intelligence in the new millennium*, vol. 8, pp. 236–239, 2003.

[51] Y. Weiss, C. Yanover, and T. Meltzer, "Map estimation, linear programming and belief propagation with convex free energies," *arXiv preprint arXiv:1206.5286*, 2012.

[52] J. K. Johnson, D. M. Malioutov, and A. S. Willsky, "Walk-sum interpretation and analysis of Gaussian belief propagation," in *Advances in Neural Information Processing Systems 18*, pp. 579–586, MIT Press, 2006.

[53] D. M. Malioutov, J. K. Johnson, and A. S. Willsky, "Walk-sums and belief propagation in gaussian graphical models," *The Journal of Machine Learning Research*, vol. 7, pp. 2031–2064, 2006.

[54] O. Shental, P. Siegel, J. Wolf, D. Bickson, and D. Dolev, "Gaussian belief propagation solver for systems of linear equations," in *IEEE Int. Symp. on Inform. Theory (ISIT)*, pp. 1863–1867, June-Nov. 2008.

[55] Y. El-Kurdi, W. J. Gross, and D. Giannacopoulos, "Efficient implementation of Gaussian belief propagation solver for large sparse diagonally dominant linear systems," *IEEE Trans. Magn.*, vol. 48, pp. 471 –474, Feb. 2012.

[56] Y. El-Kurdi, W. Gross, and D. Giannacopoulos, "Parallel solution of the finite element method using Gaussian belief propagation," in *The 15th Biennial IEEE Conference on Electromagnetic Field Computation*, p. 141, 2012.

[57] Y. El-Kurdi, D. Giannacopoulos, and W. Gross, "Relaxed Gaussian belief propagation," in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pp. 2002 –2006, July 2012.

[58] D. Dolev, D. Bickson, and J. Johnson, "Fixing convergence of Gaussian belief propagation," in *IEEE Int. Symp. on Inform. Theory (ISIT)*, pp. 1674–1678, June 2009.

[59] N. Ma, Y. Xia, and V. K. Prasanna, "Data parallelism for belief propagation in factor graphs," in *2011 23rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 56–63, IEEE, 2011.

[60] Y. Xia and V. K. Prasanna, "Scalable node-level computation kernels for parallel exact inference," *Computers, IEEE Transactions on*, vol. 59, no. 1, pp. 103–115, 2010.

[61] J. E. Gonzalez, Y. Low, C. Guestrin, and D. O'Hallaron, "Distributed parallel inference on large factor graphs," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, (Arlington, Virginia, United States), pp. 203–212, AUAI Press, 2009.

[62] K.-Y. Hsieh, C.-H. Lai, S.-H. Lai, and J. K. Lee, "Parallelization of belief propagation on cell processors for stereo vision," *ACM Trans. Embed. Comput. Syst.*, vol. 11S, pp. 13:1–13:15, June 2012.

[63] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A new parallel framework for machine learning," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, (Catalina Island, California), July 2010.

[64] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[65] MPI Forum, "Message Passing Interface (MPI) Forum Home Page." `http://www.mpi-forum.org/` (Dec. 2009).

[66] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," in *Proceedings of the Twenty-second Conference on Uncertainty in AI (UAI)*, (Boston, Massachussetts), July 2006.

[67] J. Gonzalez, Y. Low, and C. Guestrin, "Residual splash for optimally parallelizing belief propagation," in *International Conference on Artificial Intelligence and Statistics*, pp. 177–184, 2009.

[68] "Matrix Market: Harwell Boeing Collection." http://math.nist.gov/MatrixMarket/index.html.

[69] Octave community, "GNU/Octave," 2012.

[70] T. A. Davis, "Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method," *ACM Trans. Math. Softw.*, vol. 30, pp. 196–199, June 2004.

[71] T. A. Davis, "A column pre-ordering strategy for the unsymmetric-pattern multifrontal method," *ACM Trans. Math. Softw.*, vol. 30, pp. 165–195, June 2004.

[72] X. S. Li, "An overview of SuperLU: Algorithms, implementation, and user interface," *ACM Trans. Mathematical Software*, vol. 31, pp. 302–325, Sept. 2005.

[73] X. Li, J. Demmel, J. Gilbert, iL. Grigori, M. Shao, and I. Yamazaki, "SuperLU Users' Guide," Tech. Rep. LBNL-44289, Lawrence Berkeley National Laboratory, Sept. 1999. http://crd.lbl.gov/~xiaoye/SuperLU/. Last update: August 2011.

[74] P. R. Amestoy, A. Buttari, I. S. Duff, A. Guermouche, J.-Y. L'Excellent, and B. Uçar, "MUMPS," in *Encyclopedia of Parallel Computing* (D. Padua, ed.), Springer, 2011.

[75] A. George and W. H. Liu, "The evolution of the minimum degree ordering algorithm," *SIAM Rev.*, vol. 31, pp. 1–19, March 1989.

[76] A. George, "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 345–363, 1973.

[77] J. W. Demmel, *Applied Numerical Linear Algebra*. SIAM, 1997.

[78] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD, USA: Johns Hopkins University Press, 3rd ed., 1996.

[79] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*. Philadelphia, PA, USA: SIAM, 2nd ed., 2000.

[80] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*. Academic Press, 2001.

[81] J. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," tech. rep., Pittsburgh, PA, USA, 1994.

[82] G. H. Golub and D. P. O'Leary, "Some history of the conjugate gradient and Lanczos algorithms: 1948–1976," *SIAM Review*, vol. 31, pp. 50–102, March 1989.

[83] Y. Zhu and A. Cangellaris, *Multigrid Finite Element Methods for Electromagnetic Field Modeling*. IEEE Press Series on Electromagnetic Wave Theory, Wiley, 2006.

[84] V. E. Henson and U. M. Yang, "Boomeramg: a parallel algebraic multigrid solver and preconditioner," *Applied Numerical Mathematics*, vol. 41, pp. 155–177, 2002.

[85] R. Kettler, "Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods," in *Multigrid Methods* (W. Hackbusch and U. Trottenberg, eds.), Lecture Notes in Mathematics 960, pp. 502–534, Springer Berlin Heidelberg, 1982.

[86] P. P. Silvester and R. L. Ferrari, *Finite Elements For Electrical Engineers*. New York, USA: Cambridge University Press, 3rd ed., 1996.

[87] D. Fernández, D. Giannacopoulos, and W. Gross, "Multicore acceleration of cg algorithms using blocked-pipeline-matching techniques," *IEEE Trans. Magn.*, vol. 46, no. 8, pp. 3057–3060, 2010.

[88] Y. El-Kurdi, D. Fernández, E. Souleimanov, D. Giannacopoulos, and W. J. Gross, "FPGA architecture and implementation of sparse matrix-vector multiplication for the finite element method," *Computer Physics Communications*, vol. 178, no. 8, pp. 558–570, 2008.

[89] "PETSc - portable, extensible toolkit for scientific computation." http://www.mcs.anl.gov/petsc/petsc-as/. retrieved 2013.

[90] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th national conference*, ACM, (New York, NY, USA), pp. 157–172, ACM, 1969.

[91] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, April 1965.

[92] D. Bertsekas, "Distributed asynchronous computation of fixed points," *Mathematical Programming*, vol. 27, no. 1, pp. 107–120, 1983.

[93] "Gmm++: a generic template matrix c++library." http://download.gna.org/getfem/html/homepage/gmm.html. retrieved 2013.

[94] "The University of Florida Sparse Matrix Collection." http://www.cise.ufl.edu/research/sparse/matrices. submitted to ACM Transactions on Mathematical Software.

[95] MATLAB, *Release R2013b*. Natick, Massachusetts: The MathWorks Inc., 2013.

[96] A. Chronopoulos and C. Gear, "s-step iterative methods for symmetric linear systems," *Journal of Computational and Applied Mathematics*, vol. 25, no. 2, pp. 153 – 168, 1989.

[97] H. A. Bethe, "Statistical theory of superlattices," *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 150, no. 871, pp. pp. 552–575, 1935.

[98] R. Kikuchi, "A theory of cooperative phenomena," *Phys. Rev.*, vol. 81, pp. 988–1003, March 1951.

[99] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of ACM*, vol. 16, pp. 575–577, Sept. 1973.

[100] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Computer Science*, vol. 363, no. 1, pp. 28–42, 2006.

[101] E. A. Akkoyunlu, "The enumeration of maximal cliques of large graphs," *SIAM Journal on Computing*, vol. 2, no. 1, pp. 1–6, 1973.

[102] S. Szabó, "Parallel algorithms for finding cliques in a graph," *Journal of Physics: Conference Series*, vol. 268, no. 1, p. 012030, 2011.

[103] M. Darehmiraki, "A new solution for maximal clique problem based sticker model.," *Bio Systems*, vol. 95, pp. 145–9, Feb. 2009.

[104] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3." `http://eigen.tuxfamily.org`, 2010.

[105] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 3rd ed., 1999.

[106] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Santa Clara, CA, USA: Springer-Verlag TELOS, 3rd ed., 2008.

[107] D. Lasalle and G. Karypis, "Multi-threaded graph partitioning," *Parallel and Distributed Processing Symposium, International*, vol. 0, pp. 225–236, 2013.

[108] G. Karypis and K. Schloegel, *PARMETIS - Parallel Graph Partitioning and Sparse Matrix Ordering Library Version 4.0*. University of Minnesota, Department of Computer Science and Engineering, Minneapolis, MN 55455, March 2013.

[109] G. Karypis and V. Kumar, "MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0." `http://www.cs.umn.edu/~metis`, 2009.

[110] G. Karypis and V. Kumar, "Parallel multilevel k-way partitioning scheme for irregular graphs," in *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, Supercomputing '96, (Washington, DC, USA), IEEE Computer Society, 1996.

[111] T. Chan, J. Xu, L. Zikatanov, and E. C. of Science. Department of Mathematics, *An Agglomeration Multigrid Method for Unstructured Grids*. PSU applied mathematics report series, Penn State Department of Mathematics, 1998.

[112] J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 5th ed., 2012.

[113] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.

[114] B. Stroustrup, *The C++ Programming Language*. Pearson Education, 2013.

[115] S. Prata, *C++ Primer Plus*. Pearson Education, 2004.

[116] OpenMP Architecture Review Board, "OpenMP application program interface," 2013.

[117] R. Becker and R. Rannacher, "An optimal control approach to a posteriori error estimation in finite element methods," *Acta Numerica 2001*, vol. 10, pp. 1–102, May 2001.

[118] W. Bangerth and R. Rannacher, *Adaptive Finite Element Methods for Differential Equations*. Birkhäuser Verlag, 2003.

[119] A. Dedner, R. Klöfkorn, M. Nolte, and M. Ohlberger, "DUNE-FEM Web page," 2011. http://dune.mathematik.uni-freiburg.de.

[120] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey, "`libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations," *Engineering with Computers*, vol. 22, no. 3–4, pp. 237–254, 2006. `http://dx.doi.org/10.1007/s00366-006-0049-3`.

[121] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, "An updated set of basic linear algebra subprograms (blas)," *ACM Transactions on Mathematical Software*, vol. 28, pp. 135–151, 2001.

[122] J. Dongarra, "Basic Linear Algebra Subprograms Technical Forum Standard," *International Journal of High Performance Applications and Supercomputing*, vol. 16, no. 1, pp. 1–111, 2002.

[123] A. Squillacote, *The ParaView Guide*. Kitware, 2007. http://www.paraview.org/.

[124] J. Bolz, I. Farmer, E. Grinspun, and P. Schröoder, "Sparse matrix solvers on the GPU: Conjugate gradients and multigrid," *ACM Trans. Graph.*, vol. 22, pp. 917–924, July 2003.

[125] J. Rodríguez-Navarro and A. Susín, "Non structured meshes for cloth GPU simulation using FEM," in *VRIPHYS*, pp. 1–7, 2006.

[126] D. Komatitsch, D. Michéa, and G. Erlebacher, "Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA," *J. Parallel Distrib. Comput.*, vol. 69, pp. 451–460, May 2009.

[127] J. S. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Incorporated Springer Publishing Company, 1st ed., 2007.

[128] C. Cecka, A. J. Lew, and E. Darve, "Assembly of finite element methods on graphics processors," *International Journal for Numerical Methods in Engineering*, vol. 85, no. 5, pp. 640–669, 2011.

[129] A. Dziekonski, P. Sypek, A. Lamecki, and M. Mrozowski, "Generation of large finite-element matrices on multiple graphics processors," *International Journal for Numerical Methods in Engineering*, vol. 94, no. 2, pp. 204–220, 2013.

[130] A. Dziekonski, P. Sypek, A. Lamecki, and M. Mrozowski, "Finite element matrix generation on a GPU," *Progress In Electromagnetics Research*, vol. 128, 2012.

[131] Z. Fu, T. James Lewis, R. M. Kirby, and R. T. Whitaker, "Architecting the finite element method pipeline for the GPU," *Journal of Computational and Applied Mathematics*, vol. 257, pp. 195–211, 2014.

[132] N. Corporation, *NVIDIA CUDA C Programming Guide*, 2013.

[133] "Colosse supercomputing cluster." http://www.calculquebec.ca/en/resources/compute-servers/colosse. retrieved 2014.

[134] C. Loken, D. Gruner, L. Groer, R. Peltier, N. Bunn, M. Craig, T. Henriques, J. Dempsey, C.-H. Yu, J. Chen, L. J. Dursi, J. Chong, S. Northrup, J. Pinto, N. Knecht, and R. Van Zon, "SciNet: Lessons learned from building a power-efficient Top-20 system and data centre," *Journal of Physics: Conference Series*, vol. 256, no. 1, 2010.

[135] B. Janssen and G. Kanschat, "Adaptive multilevel methods with local smoothing for H^1-and H^{curl}-conforming high order finite element methods," *SIAM Journal on Scientific Computing*, vol. 33, no. 4, pp. 2095–2114, 2011.

[136] C. Burstedde, L. C. Wilcox, and O. Ghattas, "`p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees," *SIAM Journal on Scientific Computing*, vol. 33, no. 3, pp. 1103–1133, 2011.