ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Solution of Very Large Scale Linear Circuits with Combinatorial Approximation Methods and Graph Theory.

Επίλυση Πολύ Μεγάλης Κλίμακας Γραμμικών Κυκλωμάτων με Μεθόδους Συνδυαστικής Προσέγγισης και Θεωρίας Γράφων.

Διπλωματική Εργασία

Αλεξάνδρα Α. Κούρφαλη

Επιβλέποντες Καθηγητές : Νέστορας Ευμορφόπουλος
Λέκτορας

Γεώργιος Σταμούλης
Καθηγητής

Βόλος, Σεπτέμβριος 2012

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ,
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Επίλυση Πολύ Μεγάλης Κλίμακας Γραμμικών Κυκλωμάτων με Μεθόδους Συνδυαστικής Προσέγγισης και Θεωρίας Γράφων.

Διπλωματική Εργασία

Αλεξάνδρα Α. Κούρφαλη

Επιβλέποντες Καθηγητές : Νέστορας Ευμορφόπουλος
Λέκτορας

Γεώργιος Σταμούλης
Καθηγητής

Εγκρίθηκε από την διμελή εξεταστική επιτροπή την 11η Σεπτεμβρίου 2012

.....	.....
Ν. Ευμορφόπουλος	ΓΣταμούλης
Λέκτορας Καθηγητής	Καθηγητής

Διπλωματική Εργασία για την απόκτηση του Διπλώματος του Μηχανικού Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας, στα πλαίσια του Προγράμματος Προπτυχιακών Σπουδών του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας.

……..

Αλεξάνδρα Κούρφαλη

Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

To my family and friends

Ευχαριστίες

Αλεξάνδρα Κούρφαλη
Βόλος, 2012

Με την περάτωση της παρούσας εργασίας, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα της διπλωματικής μου εργασίας κ. Νέστορα Ευμορφόπουλο για την ευκαιρία που μου έδωσε να διεκπεραιώσω την εν λόγω διπλωματική, την εμπιστοσύνη που επέδειξε στο πρόσωπό μου, την άριστη συνεργασία, την καθοδήγηση και τις ουσιώδεις παρεμβάσεις, που διευκόλυναν την εκπόνηση της διπλωματικής μου εργασίας. Θα ήθελα επίσης να ευχαριστήσω τον δεύτερο επιβλέποντα της διπλωματικής μου εργασίας κ. Γιώργο Σταμούλη για τις συμβουλές του και την υποστήριξη του καθ' όλη τη διάρκεια των σπουδών μου.

Επίσης, θα ήθελα να ευχαριστήσω τους φίλους και συνεργάτες του Εργαστηρίου Ε5 για την υποστήριξη και την δημιουργία ενός πραγματικά ευχάριστου και δημιουργικού κλίματος και ιδιαίτερα τον διδακτορικό φοιτητή και φίλο Κωνσταντή Νταλούκα για τις εύστοχες υποδείξεις του και την συνεχή στήριξή του.

Ακόμη θα ήθελα να ευχαριστήσω τον διδάκτορα του τμήματος Δημήτρη Μπουντά για την καθοδήγηση, τη βοήθεια και τις συμβουλές του σε κάθε μου βήμα καθ' όλη τη διάρκεια των σπουδών μου. Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου και στους φίλους μου και ιδιαίτερα στην αδερφή μου Γιώτα, τον Γρηγόρη, τον Κωστή και τη Χρυσή για την αμέριστη υποστήριξη και την ανεκτίμητη βοήθεια που μου παρείχαν τόσο κατά την διάρκεια των σπουδών μου όσο και κατά την εκπόνηση της διπλωματικής εργασίας.

Contents

List of Tables

List of Figures

List of Acronyms

NA	Nodal Analysis
MNA	Modified Nodal Analysis
KVL	Kirchoffs Voltage Law
KCL	Kirchoffs Current Law
CCVS	Current Controlled Voltage Source
CCCS	Current Controlled Current Source
VCCS	Voltage Controlled Current Source
RHS	Right Hand Side
SDD	Symmetric Diagonally Dominant
SPD	Symmetric Positive Definite
PDE	Partial Differential Equation
HPD	Hermitian Positive Definite
CG	Conjugate Gradient
PCG	Preconditioned Conjugate Gradient
GS	Gauss Seidel
SOR	Successive Overrelaxation Method
GMRES	Generalized Minimal Residual
BiCG	BiConjugate Gradient
MG	Multigrid
CMG	Combinatorial Multigrid
AMG	Algebraic Multigrid
SpMV	Sparse Matrix-Vector multiplication
CAD	Computer Aided Design

Περίληψη

Αρκετοί αλγόριθμοι για προβλήματα συμπεριλαμβανομένου του power grid βάσίζονται σε SDD γραμμικά συστήματα. Αυτοί οι αλγόριθμοι γενικά παράγουν αποτελέσματα υψηλής ποιότητας. Ωστόσο, τα υπάρχοντα εργαλεία για την επίλυσή τους δεν είναι πάντα αποτελεσματικά, και σε πολλές περιπτώσεις λειτουργούν μόνο σε περιορισμένες τοπολογίες. Η μη διαθεσιμότητα αξιόπιστων εργαλείων επίλυσης έχει εμποδίσει την υιοθέτηση προσεγγίσεων και αλγορίθμων με βάση SDD συστήματα, ειδικά σε εφαρμογές που περιλαμβάνουν πολύ μεγάλα αραιά συστήματα.

΄Ενα βασικό θέμα αυτής της διπλωματικής είναι ότι προσεγγίσεις που βασίζονται σε SDD συστήματα μπορούν να θεωρηθούν πρακτικές και αξιόπιστες. Παρουσιάζεται το Combinatorial Multigrid (CMG), που χειρίζεται τα προβλήματα σε γενικές και αυθαίρετα σταθμισμένες τοπολογίες. Ο solver δανείζεται τη δομή και τους τελεστές των multigrid αλγορίθμων και ενσωματώνει σε αυτά ισχυρούς συνδυαστικούς preconditioners, με βάση τα εργαλεία από την θεωρία γραφημάτων.

Για να παρουσιαστεί η παραγωγή του CMG, επανεξετάζονται και αναλύονται βασικές έννοιες των γραφημάτων, επαναληπτικές μέθοδοι επίλυσης γραμμικών κυκλωμάτων και μέθοδοι διαχείρισης αραιών συστημάτων. Τα αποτελέσματα ελέγχονται σε πολύ μεγάλα συστήματα και benchmark κυκλώματα της IBM. Τέλος, ελέγχεται η συμπεριφορά και η λύση τους σε ένα τύπου Spice προσομοιωτή που αναπτύχθηκε για τον σκοπό αυτό, ο οποίος εκτελεί DC ανάλυση.

Λέξεις Κλειδιά:

Επαναληπτικές μέθοδοι, θεωρία γράφων, MNA, Multigrid

Συνδυαστικοί προρυθμιστές

Abstract

Several algorithms for problems including power grid are based on SDD linear systems. These algorithms generally produce results of high quality. However, existing solvers are not always efficient, and in many cases they operate only on restricted topologies. The unavailability of reliably efficient solvers has impeded the adoptability of approaches and algorithms based on SDD systems, especially in applications involving very large sparse systems.

A central claim of this thesis is that SDD-based approaches can be considered practical and reliable. It is presented Combinatorial Multigrid (CMG), that handles problems in general and arbitrary weighted topologies. The solver borrows the structure and operators of multigrid algorithms and embeds into them powerful combinatorial preconditioners, based on tools from support graph theory.

In order to present the derivation of CMG, we review and exemplify key notions of support graphs, iterative solvers and handlers for sparse systems. We validate our claims on very large systems and on IBM benchmark circuits. Finally, we test their behavior and solution in a Spice - like simulator developed for this purpose, who performs DC analysis.

Keywords:

Iterative methods,support graph theory, MNA, Multigrid
Combinatorial preconditioning

Chapter 1

Introduction

1.1 Introduction and Problem Description

Electronic circuit simulation uses mathematical models to replicate the behavior of an actual electronic circuit. Simulation software allows for modeling of circuit operation and is an invaluable analysis tool.

Simulating a circuits behavior before actually building it can greatly improve design efficiency by making faulty designs known as such, and providing insight into the behavior of electronics circuit designs.

In particular, for integrated circuits, the tooling is expensive, breadboards are impractical, and probing the behavior of internal signals is extremely difficult. Therefore almost all IC design relies heavily on simulation.

The most well known analog simulator is SPICE, however in our case we constructed a SPICE-like simulator that performs AC and DC analysis, to parse, solve and simulate the results.

1.2 Overall Flow

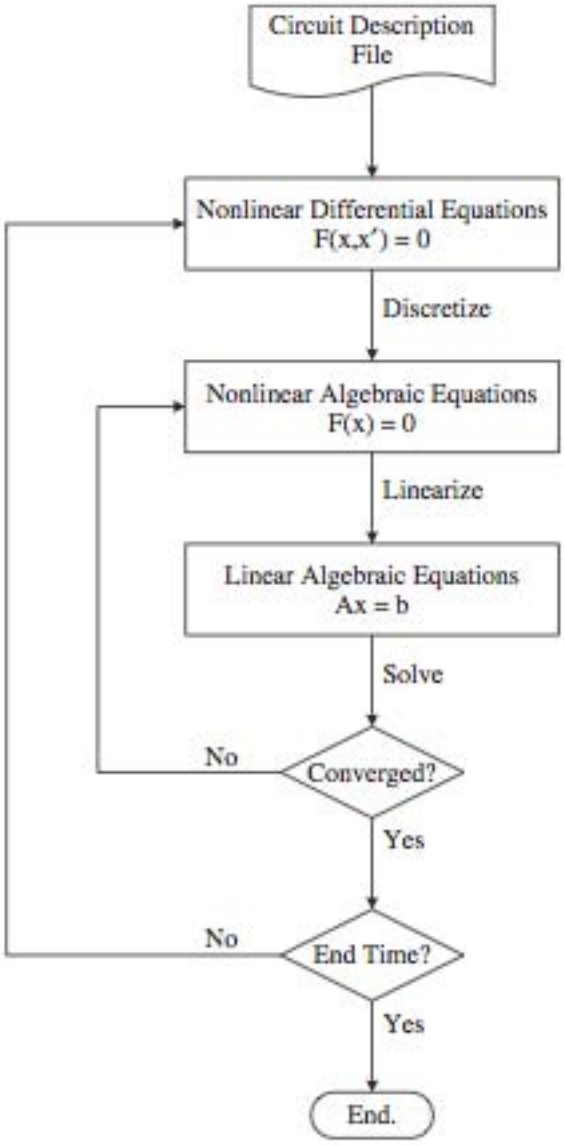

Figure 1.1: Overall Circuit Simulation Flow.

Chapter 2

MNA formulation

In this chapter MNA is described an introduction at modified nodal approach, a widely used method for formulating circuit equations in computer-aided network analysis and design programs.

2.1 NA formulation

As it was described in section **??**, the behavior of a circuit is described by a set of equations that are formulated by combining Kirchoffs Current Law (KCL) and Kirchoffs Voltage Law (KVL) and the element equations. Those results can be represented in a set of simultaneous nonlinear first-order differential equations. For a linear circuit, the equations are a system of simultaneous linear algebraic equations.

$$u_2 = \frac{R_2}{R_1 + R_2} V \tag{2.1}$$

From the figure 2.1 to put a figure here is provided the following equation which is substituted into KCL and KVL .

According to NA the network contains no voltage sources. herefore, it is not usable for general circuit simulation. As a result to the above, the branch equations can be written in the following form:

$$i = iY + s \tag{2.2}$$

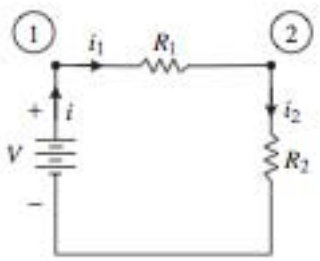

Figure 2.1: Linear Circuit.

Substitute the branch equation $i = iY + s$ into KCL, then use KVL leads to the NA formulation:

$$AYA^T u = -As \tag{2.3}$$

Conclusions and remarks

The nodal formulation emerges as a block elimination process on the tableau formulation. It generates a compact formulation which is not general and is restricted to elements that have the form $-YV_b + I = J$, therefore independent voltage sources, ideal Current Controlled Voltage Source (CCVS), ideal Current Controlled Current Source (CCCS) can not be represented in this manner.

2.2 MNA formulation.

As it was described at 3.1 ,NA shows the way forward to MNA.The widely used nodal method is a method of determining the voltage potential difference between nodes in an electrical circuit in terms of the branch currents. It has been used for formulating circuit equations in computer-aided network analysis and design programs. Although several limitations exist in this method including the inability to process voltage sources and current-dependent circuit elements in a simple and efficient way. Moreover, this approach does not scale efficiently in large circuits. As a result, a systematic equation formulation like MNA approach is needed. This formulation can be applied to any circuit in a methodical manner and the equations can be gathered directly from the circuits specification. dditionally, the coefficient matrix is sparse. MNA splits the circuit elements into groups. The first group contains the elements that have admittance description and the second group the elements that do not have one. MNA is described at Ho et al. (1975), in the following steps:

- Write KCL as Ai = 0.
- Use the element equations to eliminate as many current variables as possible from KCL, leading to equations in terms of mostly branch voltages.
- Use KVL to replace all the branch voltages by nodal voltages to ground.
- Append element equations of those elements whose current variables could not be eliminated as additional equations of the MNA system.

Those steps lead to the following MNA system:

$$\begin{bmatrix} s_u \\ s_i \end{bmatrix} = \begin{bmatrix} Y & B \\ C & Z \end{bmatrix} \times \begin{bmatrix} u \\ i \end{bmatrix}$$

2.3 Element Groups.

All elements whose currents are to be eliminated are referred to as being in group 1, while all other elements are referred to as group 2. As a result, at the vector i the currents of group 1 elements are put into i_1 , and the rest are grouped into i_2 . Also, we partition the branch

voltage vector u, so that all group 1 element voltages are grouped into u_1 , and the rest are grouped into u_2 .

The reduced MNA form is summerized to the following system:

$$\begin{bmatrix} A_{11}Y_{11}A_1^T & A_2 \\ -A_2^T & Z_{22} \end{bmatrix} \times \begin{bmatrix} u \\ i_2 \end{bmatrix} = \begin{bmatrix} -A_1 s_1 \\ s_2 \end{bmatrix}$$

2.4 Assembling the MNA System.

In a simulator, the above matrix equations arent used to construct the MNA system. Instead, it can be built by inspection, on the fly, in linear time, as the circuit description file subjected to parsing. (is being read in) There are two methods to construct the MNA system.

1. as it was described above at 3.2 For every node other than the reference node, write KCL, then:

   a) Eliminate all currents of group 1 elements using branch equations.

   b) Replace all branch voltages in terms of node voltages using KVL.

2. For every group 2 element, write its branch equation, then replace all branch voltages in terms of node voltages using KVL.

The current of every element will appear twice as a KCL current in the top equations. Elements connected to the reference node will appear once. For a group 2 element, its current and its terminal voltages will also appear as part of its element equation in the bottom equations.

2.5 Element Stamps.

The MNA equation can be formulated without using oriented graphs or incidence matrices A_1, A_2Element stamp is called the contribution of every element to the matrix equation. The process is described below:

1. matrix is initialized.

2. Right Hand Side (RHS) vector is set to zero.

3. the element stamps are added to the matrix and RHS vector as the elements are read in.

4. all the elements are read.

At this point the matrix equation is complete and begins one of the solution techniques . symbolization:

	u+	u-	RHS
u+	$\frac{1}{R}$	$-\frac{1}{R}$	
u-	$-\frac{1}{R}$	$\frac{1}{R}$	

Table 2.1: Element stamp for a resistor in group 1.

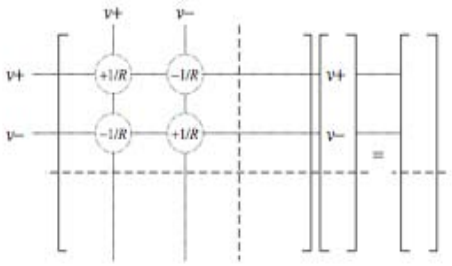

Figure 2.2: Element stamp for a resistor in group 1.

	u+	u-	i	RHS
u+			+1	
u-			-1	
i	+1	-1	-R	

Table 2.2: Element stamp for a resistor in group 2.

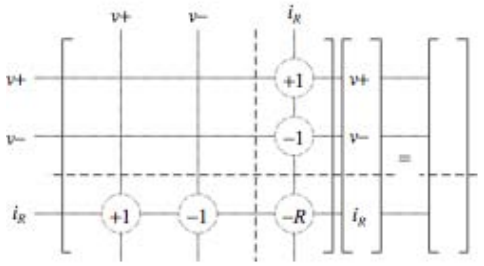

Figure 2.3: Element stamp for a resistor in group 2.

	u+	u-	RHS
u+			+1
u-			-1

Table 2.3: Element stamp for an independent current source in group 1.

	u+	u-	i	RHS
u+			+1	
u-			-1	
i			1	1

Table 2.4: Element stamp for an independent current source in group 2.

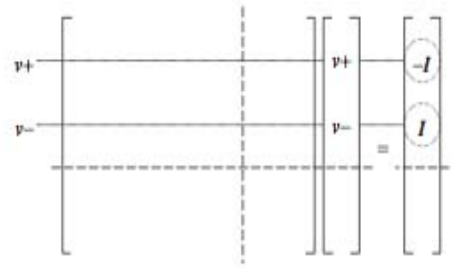

Figure 2.4: Element stamp for an independent current source in group 1.

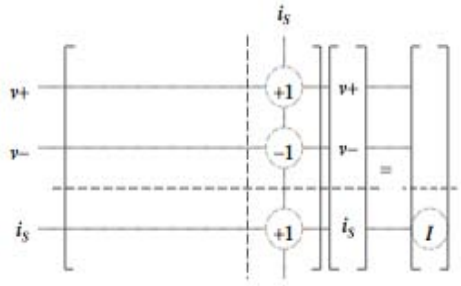

Figure 2.5: Element stamp for an independent current source in group 2.

2.6 MNA Sparsity.

$$1 + \sum_{k=1}^{n-1} 1 + d(k) = n + 2m \tag{2.4}$$

The concept of sparsity is useful in areas which have a low density of significant data or connections. An $(n \times n)$ matrix is sparse if it has $O(n)$ non-zero entries. If $1+\sum_{k=1}^{n-1} 1 + d(k) = n+2m$ the matrix is sparse, because, in a graph, the sum of all vertex degrees is equal to twice the number of edges. The MNA matrix has $n-1$ rows.Also, the element stamps contribute no more than 6 non-zero entries per stamp, then the total number of non-zero entries is no more than 6m. Therefore, if m is $O(n)$, which it usually is, then the MNA matrix is sparse.

The MNA matrix can become singular during the numerical solution process and, therefore, requires careful pivoting.

2.7 Formulation of linear dynamic equations and Element Stamps.

In this section we study the case when the network contains (linear) capacitors and inductors. In that case it is linear dynamic. The equations can be inserted at the system by an extension of the MNA formulation.

$$\begin{bmatrix} A_1 Y_{11} A_1^T & A_2 \\ -A_2^T & Z_{22} \end{bmatrix} \times \begin{bmatrix} u \\ i_2 \end{bmatrix} + \begin{bmatrix} A_1 C_{11} A_1^T & 0 \\ 0 & L_{22} \end{bmatrix} \times \begin{bmatrix} u' \\ i_2' \end{bmatrix} = \begin{bmatrix} -A s_1 \\ s_2 \end{bmatrix}$$

A capacitor C has an admittance of $j\omega C$, can be either in group 1 or 2. An inductor L has an impedance of$j\omega L$, and must be in group 2.

2.8 MNA Sparsity.

- Superposition can be used to solve a network with voltage sources using multiple applications of nodal analysis (although that would be expensive).
- For dynamic circuits, the challenge is to make sure that the order of complexity or index of the differential equations, is no greater than 2.
- Circuits with controlled sources, whether linear or nonlinear, have more problems, and require more constraints, typically of a topological nature.
- In practice, with an accurate circuit model, sufficient parasitics will typically exist in the circuit description file, which helps avoid pathological behavior, so that most practical circuits are found to be uniquely solvable.

Chapter 3

Iterative methods and Preconditioners

In this chapter is described the basic theory and some of the basic techniques for the iterative solution of very large SDD sparse linear systems, especially focused on algebraic methods suitable for sparse matrices and incomplete factorization methods as well. Also, an introduction at multilevel solvers is made. Support preconditioning theory and the most common preconditioners are presented.

3.1 Theory for Iterative Methods for the solution of linear circuits.

Until recently, direct solution methods were often preferred against iterative methods because of their predictable behavior. However, the solution of very large linear systems triggered a shift toward iterative techniques. A significant number of efficient iterative solvers were discovered and are represented in the folowing sections.

Basic Iterative Methods.

The basic iterative methods according to [1] for solving linear systems are Jacobi, Gauss-Seidel, and SOR. The solution of very large sparse linear systems of the form

$$Ax = b \tag{3.1}$$

Where $A = a_{i,j}$, of the equation 3.1 is an $n \times n$ matrix, and b a vector. The equation above remaines the central to numerous numerical simulations in computer-aided network analysis and design programs and is often the most time-consuming part of a computation.

Most of the methods covered in this chapter involve passing from one iterate to the next by modifying one or a few components of an approximate vector solution at a time. The convergence of these methods is rarely guaranteed for all matrices, but a large body of theory exists for the case where the coefficient matrix arises from the finite difference discretization of Elliptic Partial Differential Equation (PDE)s.

Jacobi Method.

$$A = D - L - U \tag{3.2}$$

From the decomposition 3.9 , D is the diagonal of A, L its lower part, and U its upper part, . It is always assumed that the diagonal entries of A are all nonzero. The Jacobi iteration in vector form can be writed as:

$$x^{k+1} = D^{-1}(L + U)x^k + D^{-1}b \tag{3.3}$$

GS Method.

As we proceed as with the Jacobi method, but in this case we assume that the equations are examined one at a time in sequence, and that previously computed results are used as soon as they are available, we obtain the Gauss-Seidel method:

$$x_i^k = \frac{(b_i - \sum_{j<i} a_{i,j}x_j^k - \sum_{j>1} a_{i,j}x_j^{k-1}}{a_{i,i}} \tag{3.4}$$

Interesting facts about the Gauss Siedel method is that the computations in 3.11 appear to be serial and that the new iterate x_k depends upon the order in which the equations are examined. In matrix terms, the definition of the Gauss-Seidel method in 3.11 can be expressed as:

$$x^k = D - L^{-1}(Ux_{k-1} + b), \tag{3.5}$$

where D, L and U represent the diagonal, lower-triangular, and upper-triangular parts of A.

SOR Method.

The SOR method, is devised by applying extrapolation to the GS method. This extrapolation takes the form of a weighted average between the previous iterate and the computed Gauss-Seidel iterate successively for each component. The idea is to choose a value for ω that will accelerate the rate of convergence of the iterates to the solution.

$$x_i^k = \omega\bar{x}_i^k + (1 - \omega)x_i^{k-1}, \tag{3.6}$$

where $\bar{x}$ denotes a Gauss-Seidel iterate, and ω is the extrapolation factor in the 3.6. In matrix terms, the SOR algorithm has the following form:

$$x^k = (D - \omega L)^{-1}(\omega U + (1 - \omega)D)x^{k-1} + \omega(D - \omega L)^{-1}b \tag{3.7}$$

CG Method .

The Conjugate Gradient method is an effective method for Symmetric Positive Definite (SPD) systems. The method proceeds by generating vector sequences of iterates, residuals corresponding to the iterates, and search directions used in updating the iterates and residuals. Only a small number of vectors needs to be kept in memory. In every iteration of the method, two inner products are performed in order to compute update scalars that are defined to make the sequences satisfy certain orthogonality conditions. On a symmetric positive definite linear

system these conditions imply that the distance to the true solution is minimized in some norm.

The CG method is not suitable for nonsymmetric systems because the residual vectors cannot be made orthogonal with short recurrences.

GMRES method.

The GMRES method generates a sequence of orthogonal vectors, but in the absence of symmetry this can no longer be done with short recurrences; instead, all previously computed vectors in the orthogonal sequence have to be retained. For this reason are used restarted versions of the method . The GMRES algorithm has the property that residual norm $\| b - Ax^i \|$ can be computed without the iterate having been formed. Thus, the expensive action of forming the iterate can be postponed until the residual norm is deemed small enough. The GMRES iterates are constructed as:

$$x^i = x^0 + y_1 u^1 + \cdots + y_i u^i \tag{3.8}$$

The GMRES method retains orthogonality of the residuals by using long recurrences, at the cost of a larger storage demand.

BiCG method.

The BiConjugate Gradient method replaces the orthogonal sequence of residuals by two mutually orthogonal sequences, at the price of no longer providing a minimization. The update relations for residuals in the Conjugate Gradient method are augmented in the BiConjugate Gradient method by relations that are similar but based on A^T instead of A. Thus we update two sequences of residuals and two sequences of search directions. The implementation is described **??**:

3.2 Computational Characteristics of the Methods.

Efficient solution of a linear system includes the selection of the proper choice of iterative method. However, to obtain good performance, consideration must also be given to the computational kernels of the method and how efficientlys they can be executed on the target architecture. The performance of direct methods, is largely that of the factorization of the matrix. However, this lower efficiency of execution does not imply anything about the total solution time for a given system. Furthermore, iterative methods are usually simpler to implement than direct methods, and since no full factorization has to be stored, they can handle much larger sytems than direct methods.

3.3 Multigrid method.

Multigrid (MG) method in numerical analysis is defined as a group of algorithms for solving differential equations using a hierarchy of discretizations. They are an example of a class of techniques called multiresolution methods, very useful in problems exhibiting multiple scales of behavior. For example, many basic relaxation methods exhibit different rates of convergence for short- and long-wavelength components, suggesting these different scales be

	Summary of Operations for Iteration i				
Method	Inner Product	SAXPY	Matrix Vector Product	Precond Solve	storage reqmts
JACOBI			1^a		matrix + 3n
G- S		1	1^a		
SOR		1	1^a		matrix + 2n
CG	2	3	1	1	matrix + 6n
GMRES	i+1	i+1	1	1	matrix + (i + 5)n
BiCG	2	5	1/1	1/1	matrix + 10n

Table 3.1: Summary of Operations for Iteration i.

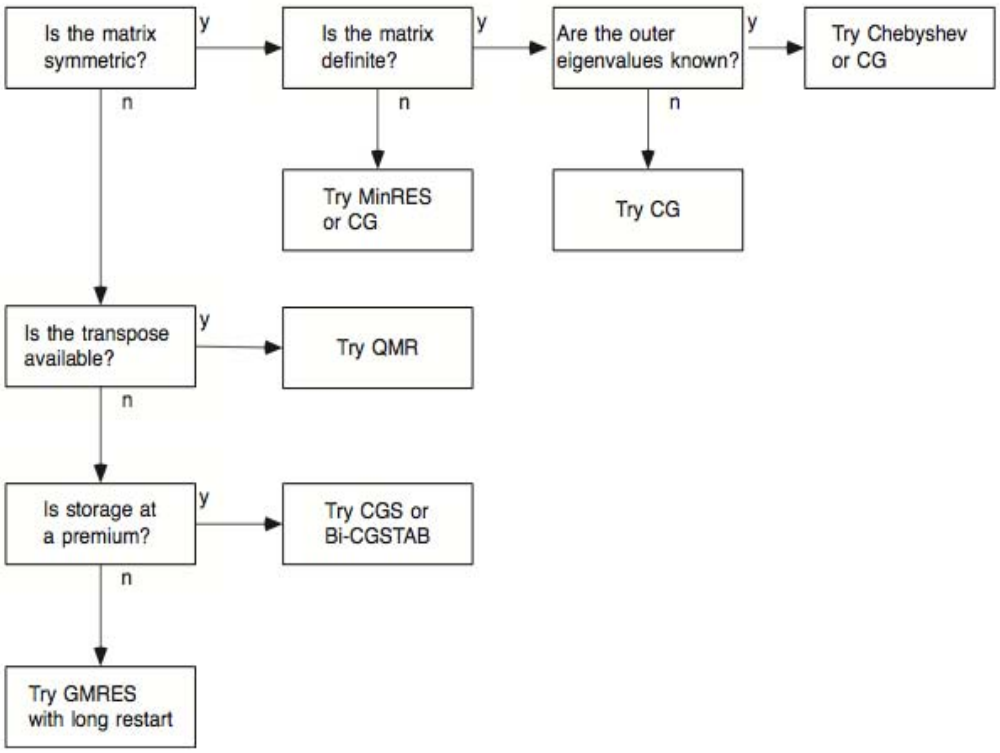

Figure 3.1: Flowchart of iterative methods.

treated differently, as in a Fourier analysis approach to multigrid. MG methods can be used as solvers as well as preconditioners.

The main idea of MG is to accelerate the *convergence* of a basic iterative method by global correction from time to time, accomplished by solving a coarse problem.[1] This principle is similar to interpolation between coarser and finer grids. The typical application for multigrid is in the numerical solution of elliptic partial differential equations in two or more dimensions.

Multigrid can be applied in combination with any of the common discretization techniques. MG methods are among the fastest solution techniques known today. In contrast to other methods, multigrid methods are general in that they can treat arbitrary regions and boundary conditions. They do not depend on the separability of the equations or other special properties of the equation.

3.4 Preconditioning.

Preconditioner is a matrix that transforms the system 3.1 into another system with more preferable properties for iterative solution. Preconditioning generally attempts to improve the spectral properties of the coefficient matrix. For SPD problems, the rate of convergence of the conjugate gradient method depends on the distribution of the eigenvalues of A. The purpose of preconditioning is that the transformed matrix in question will have a smaller spectral condition number, and/or eigenvalues clustered around 1. For nonsymmetric problems the situation is more complicated, and the eigen-values may not describe the convergence of nonsymmetric matrix iterations like GMRES. On parallel machines there is a further trade-off between the efficacy of a preconditioner in the classical sense, and its parallel efficiency. Many of the traditional preconditioners have a large sequential component.

If M is a nonsingular matrix that approximates A, then the linear system 3.9 has the same solution as 3.1 but must be significantly easier to solve.

$$M^{-1}Ax = M^{-1}b \tag{3.9}$$

$$AM^{-1}y = b, x = M^{-1}y \tag{3.10}$$

$$M_1^{-1}AM_2^{-1}y = M_1^{-1}b, x = M_2^{-1}y \tag{3.11}$$

The system 3.9 is preconditioned from the left, 3.10 is preconditioned from the right. At 3.11 is performed split preconditioning where the preconditioner is M = M1 M2.

Preconditioner Construction Basic Goals.

- The linear system M should be easier than A to solve.
- For dynamic circuits, the challenge is to make sure that the order of complexity or index of the differential equations, is no greater than 2.

[1] Coarse problem is an auxiliary system of equations used in an iterative method for the solution of a given larger system of equations. It is basically a version of the same problem at a lower resolution, retaining its essential characteristics, but with fewer variables.

- Circuits with controlled sources, whether linear or nonlinear, have more problems, and require more constraints, typically of a topological nature.
- In practice, with an accurate circuit model, sufficient parasitics will typically exist in the circuit description file, which helps avoid pathological behavior, so that most practical circuits are found to be uniquely solvable.

Convergence.

Preconditioning is a concept created to improve the convergence of iterative methods. One of the biggest problems with preconditioning is that convergence analysis is generally limited to simple model problems. For problems with irregular numerical or topological structure, condition number bounds are generally difficult to obtain.

The convergence of the methods described at 3.1 , 3.1 , 3.1 , (Jacobi, GS and SOR) is rarely guaranteed for all matrices, but a large body of theory exists for the case where the coefficient matrix arises from the finite difference discretization of Elliptic PDEs.

The convergence behavior of the different algorithms seen in this chapter can be analyzed by exploiting optimality properties whenever such properties exist. This is the case for CG and GMRES algorithms.

If A and M are SPD, the convergence of many preconditioned iterative methods (and specifically, PCG) depends on the condition number of the preconditioned operator.

We define the generalized (spectral) condition number by

$$\kappa(A, B) = max_x \frac{x^T Ax}{x^T Bx} \times max_x \frac{x^T Bx}{x^T Ax}, \tag{3.12}$$

where x, is outside the null space of A.

The fact whether or not, or the rate at which, an iterative method approaches the solution of a linear system. Convergence can be:

Linear: Some measure of the distance to the solution decreases by a constant factor in each iteration.

Smooth: The measure of the error decreases in all or most iterations, though not necessarily by the same factor.

Irregular: The measure of the error decreases in some iterations and increases in others. This observation unfortunately does not imply anything about the ultimate convergence of the method.

Stalled: The measure of the error stays more or less constant during a number of iter- ations. As above, this does not imply anything about the ultimate convergence of the method.

The convergence speed of iterative methods may depend on the ordering used, and often the parallel efficiency of a method on a Parallel Computer [2] is strongly dependent on the ordering used.

[2]Computer with multiple independent processing units. If the processors have immediate access to the same memory, the memory is said to be shared; if processors have private memory that is not immediately visible to other processors, the memory is said to be distributed. In that case, processors communicate by message-passing.

Iterative Preconditioners.

Although the methods seen in chapters 3.1 , 3.1 , 3.1 , (BiCG, CG and GMRES) are well founded theoretically, they are all likely to suffer from slow convergence for problems which arise from typical applications such as fluid dynamics or electronic device simulation. Preconditioning is a key ingredient for fast convergence.

$$x_{i+1} = (I - M^{-1}A)x_i + b' \tag{3.13}$$

where 3.13 is the basic method of iterative preconditioning.

Incomplete Factorization Preconditioners.

A broad class of preconditioners is based on incomplete factorizations of the coefficient matrix. We call a factorization incomplete if during the factorization process certain fill elements, nonzero elements in the factorization in positions where the original matrix had a zero, have been ignored. Such a preconditioner is then given in factored form M = LU with L lower and U upper triangular. The efficacy of the preconditioner depends on how well M^{-1} approximates A^{-1}.

When a sparse matrix is factored by Gaussian elimination, fill-in usually takes place. In that case, sparsity-preserving pivoting techniques can be used to reduce it. The triangular factors L and U of the coefficient matrix A are considerably less sparse than A.

Sparse direct methods are not considered viable for solving very large linear systems due to time and space limitations , however, by discarding part of the fill-in in the course of the factorization process, simple but powerful preconditioners can be obtained in the form $M = \bar{L}\bar{U}$, where $\bar{L}$ and $\bar{U}$ are the incomplete (approximate) LU factors.

Summarizing, it can be said that existing solutions to the problem for incomplete factorization preconditioners for general SPD matrices follow one of two cases: simple inexpensive fixes that result in low quality preconditioners in terms of convergence rating, or sophisticated, expensive strategies that produce high quality preconditioners.

Chapter 4

Graph Theory

In this chapter is described the basic graph theory, an overview of the definitions and methods for graph clustering, that is, finding sets of related vertices in graphs. Moreover, it is shown how weighted graph $[\phi,\rho]$ decompositions can be used in the first known linear work parallel construction of provably good preconditioners. Furthermore, it is presented the reduction to the same problem in a sparser, tree-like, spanning subgraph of the given graph. For fixed degree graphs, it is presented a parallel construction of combinatorial preconditioners with a constant condition number.

4.1 Support Theory for Graphs.

Support theory, is a recent methodology for bounding condition numbers of preconditioned systems. More specifically, it is a set of tools and techniques for bounding extremal eigenvalues. For some iterative methods (conjugate gradients in particular), the ratio of largest to smallest eigenvalues provides an upper bound on the number of iterations.

Definitions of graph theory

Graphs are structures formed by a set of vertices (also called nodes) and a set of edges that are connections between pairs of vertices. Graph clustering is the task of grouping the vertices of the graph into clusters taking into consideration the edge structure of the graph in such a way that there should be many edges within each cluster and relatively few between the clusters.

Terminology

Worst-case running time The worst-case running time of an algorithm for a problem instance of size x is the number of computation steps needed to execute the algorithm for the most difficult instance of size x possible.

Worst-case memory consumption The number of memory units that the algorithm will need to simultaneously occupy in the worst possible case for an instance of size x.

Computational complexity The interest is in characterizing how the running time and memory consumption grow when size x grows.

Approximation algorithms They focus is finding efficiently a solution that differs no more than a fixed factor from the exact solution.

A graph G is a pair of sets G=(V,E). V is the set of vertices and the number of vertices $n = |V|$ is the order of the graph. The set E contains the edges of the graph. In an undirected graph, each edge is an unordered pair v, w. In a directed graph, edges are ordered pairs. The vertices v and w are called the end points of the edge. The edge count $|E| = m$ is the size of the graph. In a weighted graph, a weight function $\omega : E \to \mathbb{R}$ is defined that assigns a weight on each edge. A graph is planar if it can be drawn in a plane without any of the edges crossing.

Graph density of a graph G = (V, E) as the ratio of the number of edges present to the maximum possible, $\delta(G) = {}^{m}/_{\binom{n}{2}}$.

A graph of density one is called complete. Also, If $v, u \in E$,then v is a neighbour of u. The set of neighbours for a given vertex v is called the neighbourhood of v and is denoted by $\Gamma(v)$. A vertex v is a member of its own neighbourhood$\Gamma(\ v)$ if and only if the graph contains a reflexive edge v, v. The adjacency matrix A_G of a given graph G = (V, E) of order $n \times n$ matrix $A_G = (a_v^G, u)$, where

$$A_G = (a_v^G, u) = \begin{cases} 1 & \text{if } v, u \in E, \\ 0, & \text{otherwise.} \end{cases}$$

The number of edges incident on a given vertex v is the degree of v and is denoted by $deg(v)$. A graph is regular if all of the vertices have the same degree; if $\forall v \in VinG = (V, E)$ we have $deg(v)$ = k, the graph G is k-regular.

A partition of the vertices V of a graph G = (V, E) into two nonempty sets is called a cut. The length of a path is the number of edges on it, and the distance between v and u is the length of the shortest path connecting them in G. The distance from a vertex to itself is zero: the path from a vertex to itself is an empty edge sequence. A graph is connected if there exist paths between all pairs of vertices. If there are vertices that cannot be reached from others, the graph is disconnected. The minimum number of edges that would need to be removed from G in order to make it disconnected is the edge-connectivity of the graph. A cycle is a simple path that begins and ends at the same vertex. A graph that contains no cycle is acyclic and is also called a forest. A connected forest is called a tree. A subgraph $G_S = (S, ES)$ of G = (V,E) is composed of a set of vertices$S \subseteq V$ and a set of edges $E_S \subseteq E$ such that $v, u \in E_S$ implies $v, u \in S$; the graph G is a supergraph of G_S. A connected acyclic subgraph that includes all vertices is called a spanning tree of the graph. A spanning tree has necessarily exactly n - 1 edges. If the edges are assigned weights, the spanning tree with smallest total weight is called the minimum spanning tree. Note that there may exist several minimum spanning trees that may even be edge-disjoint.

The number of iterations of the conjugate gradient method for the solution of systems of linear equations Ax = b is bounded above by the square root of the spectral condition number $\kappa(A)$ of A. (The actual number of iterations can be significantly smaller in some cases.) The condition number is the ratio of the extreme eigenvalues of A, $\kappa(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}$. The conjugate gradient method can be used to solve consistent linear systems with a singular coefficient matrix A (in floating-point arithmetic, it helps to orthogonalize the search directions against the null space if A is singular). In such cases, the number of iterations is proportional to the square root of the ratio of the extreme positive eigenvalues. When a preconditioner B is used

in the conjugate gradient method, the number of iterations is proportional to the square root of the ratio of the extreme finite generalized eigenvalues of the pencil (A,B), defined below.

Definition of the number λ: is a finite generalized eigenvalue of the matrix pencil (A,B) if there exists a vector$x \neq 0$ such that $Ax = \lambda Bx$ and $Bx \neq 0$. We denote the set of finite generalized eigenvalues by λ_f (A, B).

Generalized eigenvalues: The set of generalized eigenvalues $\Lambda(A, B)$ of a pair of Laplacians and is defined by: $\Lambda(A, B)$ = there is real vector x such that $Ax = \lambda Bx$

Rayleigh quotient characterization of support: If A, B have the same size, we have:

$$\lambda_{max}(A, B) = \sigma(A, B) = max x^T j \neq 0 \frac{(x^T Ax)}{x^T Bc}, \tag{4.1}$$

where j denotes the constant vector.

Schur complement: Let T be a weighted star with n + 1 vertices and edge weights $d_1, \ldots, d_n$. The Schur complement S(T, v) of T withrespect to its root v, is the graph defined by the weights $S_{ij}(T, u) = {}^{d_i d_j}/_D$ where $D = \sum_i d_i$. Let A be any graph, $A[V - v]$ be the graph induced in A by the vertices in Vv, and T_v be the star graph consisting of the edges incident to v in A. The Schur complement $S(A, v)$ of A with respect to vertex v is the graph $A[Vv] + S(T_v, v)$. Let $W \subset V$ and v be any vertex in W. The Schur complement with S(A,W) is recursively defined as:

$$S(A, W) = S(S(A, v), Wv) = S(S(A, Wv), v). \tag{4.2}$$

Let A, B be positive definite matrices. We let $\lambda_1 \leq \ldots \leq \lambda_n$ denote the eigenvalues of A and $\mu_1 \leq \ldots \leq \mu_n$ denote the eigenvalues of B. Let κ_{max} and κ_{min} denote $\lambda_{max}(A, B)$ and $\lambda_{min}(A, B)$.

We therefore have $\lambda_{max}(B, A) = {}^1/_{\kappa_{min}}$ and $\lambda_{min}(B, A) = \frac{1}{\kappa_{max}}$.

Splitting Lemma: If $A = \sum_{i=1}^m A_i$ and $M = \sum_{i=1}^m M_i$ where A_i, M_i are Laplacians, then $\sigma(A, M) \leq max_i \sigma(A_i, M_i)$.

The Splitting Lemma allows bounding of the support of A by M, by splitting the power dissipation in A into small local pieces, and supporting them by also local pieces in M.

If we take M as the maximal weight spanning tree of A, it is easy to show that $\sigma(M, A) \leq 1$, intuitively because more resistances always dissipate more power. In order to bound $\sigma(A_i, M_i)$, the basic idea to let the A_i be edges on A (the ones not existing in M), and let M_i be the unique path in the tree that connects the two end-points of A_i. Then one can bound separately each $\sigma(A_i, M_i)$. In fact, it can be shown that any edge in A that doesnt exist in M, can be supported only by the path M_i. Steiner preconditioners, introduced in 3.4 introduce external nodes into preconditioners. CMG is based on a partitioning of the n vertices in V into m vertex-disjoint clusters V_i. For each V_i, the preconditioner contains a star graph S_i with leaves corresponding to the vertices in V_i rooted at a vertex r_i. The roots r_i are connected and form the quotient graph Q.

$$S = \begin{pmatrix} D' & -D'R \\ -R^T D' & Q + R^T D' R \end{pmatrix}$$

D' is the total degree of the leaves in the Steiner preconditioner S. R is the restriction of an $n \times m$ matrix, where $R(i,j) = 1$ if vertex i is in cluster j and 0 otherwise. Then, the Laplacian of S has n + m vertices, and the algebraic form is the above.

Every time a system of the form $Bz = y$ is solved in an usual preconditioned method, the system:

$$S \begin{pmatrix} z \\ z' \end{pmatrix} = \begin{pmatrix} y \\ 0 \end{pmatrix}$$

should be solved instead, for a set of dont care variables z'. Also the operation is equivalent to preconditioning with the dense matrix

$$B = D' - V(Q + D_Q)^{-1} V^T, \tag{4.3}$$

where $V = D'R$ and $D_Q = R^T D' R$ and B is the Laplacian which is also called the Schur complement of S with respect to the elimination of the roots r_i .

The analysis of the support $\sigma(A, S)$), is identical to that for the case of subgraph preconditioners. For two roots r_i, r_j we should have

$$w(r_i, r_j) = \sum_{i' \in V_i, j' \in V_j} w_{i,j}. \tag{4.4}$$

Thus, he algebraic form of the quotient Q is $Q = R^T AR$.

Graph Conductance.

The *conductance* of the graph is the minimum sparsity value over all possible cuts. It controls how fast a random walk on G converges to a uniform distribution.

P partitions the vertices of a graph G = (V,E,w) into disjoint sets V_i, $i = 1, \ldots, m$ and G_i denotes the graph induced by the vertices in V_i. The n/m is the vertex reduction factor of P and it is denoted by ρ. P is set as a (ϕ,γ) decomposition if the conductance of each G_i is bounded below by ϕ and for each vertex v $\in V_i$, $\frac{cap(v,V_i v)}{vol(v)} \geq \gamma$.

It is considered a variant of (ϕ,γ) decompositions. For each G_i in the partition, is set a vertex on each edge leaving G_i. If W_i is the set of newly set vertices forG_i , then P is [ϕ,γ] decomposition if the closure graph G_i^o induced by the vertices in $V_i \cup W_i$ has conductance bounded below by ϕ and the vertex reduction factor of P is at least ρ. By definition, G_i^o is G_i with additional degree one vertices hanging off of it. Therefore, any edge cut in G_i induces a sparser cut in G_i^o , and thus the conductance of G_i must be lower bounded by ϕ. Besides, if G_i contains two vertices v_1 , v_2 such that $\frac{cap(v_j,V_i v_j)}{vol(v_j)} \leq \phi$ for $1, 2, \cdots$ the conductance of G_i^o is less than ϕ.

It turns out that the parallel computation of [ϕ,ρ] decompositions is not trivial even for trees, for which is needed machinery from parallel tree contraction algorithms.

Theorem 1 Trees have a [1/2,6/5] - decomposition that can be computed with linear work in $O(\log n)$ parallel time.

Theorem 2 Planar graphs have a [ϕ,ρ]-decomposition such that $\phi\rho$ is constant. The decomposition can be constructed with linear work in $O(\log n)$ parallel time.

Theorem 3 Graphs with no K_s minor or s^2 genus have a $[(1/(\log^3 ns^2 logs)), O(1)]$ decomposition which can be computed in $O(n \log 2n)$ time.

Theorem 4 The support $\sigma(S/A)$ is bounded by a constant c independent from n, if and only if for all i the conductance of the graph $A^o[V_i]$ induced by the nodes in V_i augmented by the edges leaving V_i is bounded by a constant c'.

Thus, the conductance $\phi(A)$ of graph A = (V,E,w) is defined as

$$\phi(A) = min_{S \subseteq V} \frac{w(S, V-S)}{min(w(S), w(V-S))}, \tag{4.5}$$

where w(S, V-S) denotes the total weight connecting the sets S and V - S, and w(S) denotes the total weight incident to the vertices in S.

4.2 Graph Clustering.

Any nonuniform data contains underlying structure due to the heterogeneity of the data. The process of identifying this structure in terms of grouping the data elements is called *clustering*. Graph clustering in the sense of grouping the vertices of a given input graph into clusters[1] The goal of clustering is to divide the data set into clusters such that the elements assigned to a particular cluster are similar or connected in some predefined sense.

Clustering is an important issue in the analysis and exploration of data. There is a wide area of applications as e.g. VLSI, data mining, design, computer graphics and gene analysis. A natural notion of graph clustering is the separation of sparsely connected dense subgraphs from each other.

Matrix diagonalization in itself is an important application of clustering algorithms, as there are efficient computational methods available for processing diagonalized matrices, for example, in our case, to solve linear systems. Such computations enable efficient algorithms for graph partitioning, *as the graph partitioning problem can be written in the form of a set of linear equations*.

The goal in graph partitioning is to minimize the number of edges that cross from one subgroup of vertices to another, usually posing limits on the number of groups as well as to the relative size of the groups.

Weighted graph decomposition occurs, with n vertices into a collection P of vertex disjoint clusters such that, for all clusters C $\in$ P, the graph induced by the vertices in C and the edges leaving C, has conductance bounded below by ϕ. It is shown that for planar graphs a decomposition P can be computed such that $\mid P \mid < n/\rho$, where ρ is a constant, in $O(\log n)$ parallel time with O(n) work. Slightly worse guarantees can be obtained in nearly linear time for graphs that have fixed size minors or bounded genus. It is shown how these decompositions can be used in the first known linear work parallel construction of provably good preconditioners for the important class of fixed degree graph Laplacians.

Partitioning weighted graphs into disjoint and dissimilar clusters of similar vertices is one of the most important algorithmic problems with applications in web clustering, text

[1] and must not be confused with the clustering of sets of graphs based on *structural similarity*.

retrieval,computer aided diagnosis and computational biology. The importance of good clusterings with as few clusters as possible, defined as the number of vertices in the given graph over the number of clusters, is critical.

The quality of a clustering is characterized by the minimum conductance ϕ over the clusters and the ratio γ of the weight going between clusters over the total weight of the edges in the graph. An interesting property of this bicriteria mea- sure, that we will subsequently call ϕ,γ^{aug} decomposition, is its connection to the sparsest cut problem. Assuming a two-way algorithm returns a cut of sparsity at most when there is cut of sparsity ϕ , its recursive application returns $a((\phi/\sigma)1/,[(\sigma\gamma)^v]^{avg})$ decomposition when the graph has $a(\phi,\gamma^{avg})$ decomposition. The complexity of the recursive algorithm is at least a logarithmic factor slower than the two-way algorithm, but it can be considerably slower because in general the two-way algorithm is not expected to return balanced cuts.

A stronger type of clusterings, (ϕ,γ) decompositions, is implicit in the *laminar decompositions* , that usually describe a tree, constructed in the context of low congestion oblivious routing. In a (ϕ,γ) decomposition all clusters have minimum conductance *phi*, but now the for every vertex v the total weight incident to v that stays within vs cluster is at least a fraction γ of the total weight incident to v.

The vertex reduction factor is constant in average, but there are no guarantees for the reduction factor between subsequent levels of the decomposition.

The local partitioning algorithm as well as other heuristic variants exploit the connection of (ϕ,γ) decompositions with random walks. A particle doing a random walk tends to get trapped in clusters of high conductance when the vertices of the cluster are connected to the exterior with relatively light edges; then the probability distribution P_v^t after a small number t of steps of the random walk starting at a given vertex v is expected to provide information about the cluster where v belongs. While this local intuition can be captured mathematically, obtaining a multi-way decomposition by computing independently several such probability distributions , is a quite complicated task when the running time must be nearly linear. In contrast, computing arbitrary distribution mixtures of the form $\sum_{v\in V} w_v P_u^t$ is straightforward and can be done in time linear in t and the number of edges in the graph.

4.3 Planar Decompositions.

Tree decompositions of graphs are of fundamental importance in structural and algorithmic graph theory. Planar decompositions generalise tree decompositions by allowing an arbitrary planar graph to index the decomposition. Every graph that excludes a fixed graph as a minor has a planar decomposition with bounded width. Planar decompositions are intimately related to the crossing number. In particular, a graph with bounded degree has linear crossing number if and only if it has a planar decomposition with bounded width and linear order. It follows from the above result about planar decompositions that every graph with bounded degree and an excluded minor has linear crossing number.

$$cap(U,V) = \sum_{u\in U, v\in V} w(u,v) \tag{4.6}$$

$$\frac{cap(V,V'-V)}{min(vol(V'),vol(V-V'))} \tag{4.7}$$

$$Ax = b \iff \begin{pmatrix} D + A_n & -A_p \\ -A_p & D + A_n \end{pmatrix} \begin{pmatrix} x \\ -x \end{pmatrix} = \begin{pmatrix} b \\ -b \end{pmatrix}$$

The total weight connecting the nodes of the disjoint sets U,V is modeled at 4.6 , where, $G = (V, E, w)$ a weighted graph. Also, the *sparsity* of an edge cut into $VandV - V$ is defined as the ratio at 4.7 . The total incident weight f vertex v is denoted by *vol(v)*.

4.4 Graphs as electric networks.

There is a fairly well known analogy between graph Laplacians and resistive networks. If G is seen as an electrical network with the resistance between nodes i and j being $1/w_{i,j}$, then in the equation $Av = i$ is v is the vector of voltages at the node, i is the vector of currents. Also, the quadratic form $v^T Av = \sum_{i,j} w_{i,j}(v_i - v_j)^2$ expresses the power dissipation on G, given the node voltages v. In view of this, the construction of a good preconditioner M amounts to the construction of a simpler resistive network (for example by deleting some resistances) with an energy profile close to that of A.

The support of A by M, defined as $\sigma(A/M) = max_v {}^{v^T Av}/_{v^T M}$, where v is the number of copies of M that are needed to support the power dissipation in A, for all settings of voltages. The principal reason behind the introduction of the notion of support, is to express its local nature, captured by the Splitting Lemma.

4.5 Graphs as SDD linear systems.

SDD linear systems can be viewed entirely as graphs. CMG advocates a principled approach to the solution of linear systems. The core of CMG and all other solvers designed in the context of combinatorial preconditioning is in fact a solver for a special class of matrices, graph Laplacians. The Laplacian A of a graph G = (V, E, w) with positive weights, is defined by:

$$A_{i,j} = A_{j,i} = -w_{i,j} and \tag{4.8}$$

$$A_{i,i} = -\sum_{i \neq j} A_{i,j}. \tag{4.9}$$

More general systems are solved via light-weight transformations to Laplacians. Consider for example the case where the matrix A has a number of positive off-diagonal entries, and the property $A_{i,i} = \sum_{i \neq j} |A_{i,j}|$. Positive off-diagonal entries have been a source of confusion for Algebraic Multigrid (AMG) solvers, and various heuristics have been proposed. Instead, CMG uses the double- cover reduction. Let $A = A_p + A_n + D$, where D is the diagonal of A and A_p is the matrix consisting only of the positive off-diagonal entries of A. Then from:
In this way, the original system is reduced to a Laplacian system, while at most doubling the size. In practice it is possible to exploit the obvious symmetries of the new system, to solve it with an even smaller space and time overhead.

Matrices of the form $A + D_e$, where A is a Laplacian and D_e is a positive diagonal matrix have also been addressed in various ways by different AMG implementations. In CMG, we

again reduce the system to a Laplacian. If d_e is the vector of the diagonal elements of D, then:

$$Ax = b \iff \begin{pmatrix} A + D_e & 0 & -d_e \\ 0 & A + D_e & -d_e \\ -d_e^T & -d_e^T & \sum_i d_e(i) \end{pmatrix} \begin{pmatrix} x \\ -x \\ 0 \end{pmatrix} = \begin{pmatrix} b \\ -b \\ 0 \end{pmatrix}$$

It is possible to implement the reduction in a way that exploits the symmetry of the new system, and with a small space and time overhead work only implicitly with the new system.

A symmetric matrix A is SDD, if $A_{i,i} \geq \sum_{i \neq j} |A_{i,j}|$. The two reductions above can reduce any SDD linear system to a Laplacian system. Those matrices with non-positive off diagonals are known as M-matrices. It is well known that if A is an M-matrix, there is a positive diagonal matrix D such that A = DLD, where L is a Laplacian. Assuming D is known, an M-system can also be reduced to a Laplacian system via a simple change of variables. In many application D is given, or it can be recovered with some additional work.

There is a one-to-one correspondence between Laplacians and graphs, so the terms are often used interchangeably.

Chapter 5

Support Graphs

In this chapter is analyzed the Support Theory which is a fairly recent methodology for bounding condition numbers of preconditioned systems and extremal eigenvalues. It is also covered the relation between CMG and support graph theory.

5.1 Support Graph Preconditioning.

Support-graph preconditioning is to first construct a graph according to a given matrix A, and then compute the preconditioner M as a reduced matrix based on the support of the graph. The support is usually a specific subgraph extracted from the full graph.

Formally, support-graph preconditioning is to compute a preconditioner M such that the generalized eigenvalues and the condition number of the matrix pencil (A, M) are bounded. If both A and M are SPD matrices, the convergence depends on the condition number (A,P) computed as follows:

$$\kappa(A, M) = \frac{\lambda_{max}(A, M)}{\lambda_{min}(A, M)} \tag{5.1}$$

where $\lambda(A, M)$ denotes the generalized eigenvalue. A stronger theoretical result on convergence can be derived as follows. Define the support of (A,M), denoted by $\sigma(A, P)$, as follows:

$$\sigma(A, M) = \{\tau \in \mathbb{R} | xT(\tau P - A)x \geq 0 \forall x \in \mathbb{R}^n\} \tag{5.2}$$

To more effectively trade off the the memory consumption and convergence rate of iterative algorithms, support-graph preconditioner can be naturally applied to accelerate large-scale power grid analysis by extracting the maximum spanning tree from the original power grid structure. Existing direct matrix solvers can be applied to generate the support-graph preconditioner based upon the spanning- tree theory. Since the support graph maintains a tree-structure, during the matrix factorization process, the number of new fill-ins can be very well controlled, especially when appropriate node ordering techniques are used. As a result, compared with existing black-box incomplete matrix factorization-based preconditioners such as incomplete Cholesky preconditioner, the matrix factor associated with the support graph can serve as a more efficient preconditioner with guaranteed convergence. To achieve higher efficiency in large-scale power grid analysis, *hierarchical support graphs* are used.

Hierarchical Support Graph.

Instead of using the low-stretch spanning tree, it is used a maximum spanning tree to approximate the low-stretch spanning tree. The procedure is the following:

Arbitrarily pick a starting node, add it to the visited node list, choose the maximum weighted outgoing edge from the visited node list, and add the new node in. Repeat the above procedure until all nodes are visited.

5.2 Preconditioning for fixed degree graphs.

A simple and fully parallel implementation can be made in the case of fixed degree graphs. The decomposition is computed by performing the following simple steps:

I From the given graph A, form the graph $\hat{A}$ by independently perturbing each edge by a random constant in (1, 2).

II For each vertex u keep in A the heaviest incident edge of u in $\hat{A}$, to form a subgraph B of A, which is a forest of trees.

III Independently split each tree in B into clusters of size at most k for some constant k.

To see why step 1 generates a forest B, consider the graph $\hat{B}$ consisting of the edges of B with their weighting in $\hat{A}$. The graph $\hat{B}$ is unimodal, i.e. for each path $u_1, \ldots, u_k$ there is no edge u_i, u_{i+1} which is lighter than its two adjacent edges. This happens because u_i, u_{i+1} is the heavier incident edge of either u_i or u_{i+1} . From this it also follows that $\hat{B}$ and thus B are forests of trees. If the maximum degree in the graph is d, this simple process generates a $[2d^2k, 2]$ decomposition. This occurs because the conductance of the closure of each cluster in $\hat{B}$ is at least $\frac{1}{dk}$ by the unimodality property. This implies that the conductance of the closure of every class in A is at most $\frac{1}{2d^2k}$. The reduction factor is at least 2 because every vertex is assigned to a cluster. Decomposition can be used to construct a preconditioner with a constant condition number and at most $\frac{n}{2}$ Steiner vertices.

5.3 Combinatorial Preconditioning.

Combinatorial preconditioning which is motivated by the problem of simplifying linear systems and is defined by a technique that relies on graph algorithms to construct effective preconditioners. It studies the approximation of graphs by other simpler graphs with respect to the condition number metric.

The simplest applications of combinatorial preconditioning target a class of matrices that are isomorphic to a weighted undirected graph. The coefficient matrix A is viewed as its isomorphic graph G_A. A specialized graph algorithm constructs another graph G_B such that the isomorphic matrix B is a good preconditioner for A. The graph algorithm aims to achieve two goals: the inverse of B should be easy to apply, and the spectrum of $B^{-1}A$ should be clustered. It turns out that the spectrum of $B^{-1}A$ can be bounded in terms of properties of the graphs G_A and G_B; in particular, the quality of embeddings of G_A in G_B (and sometimes vice versa) plays a fundamental role in these spectral bounds.

5.4 Steiner Preconditioners.

The fast construction of preconditioners that use extra vertices.The laminar decomposition can be used for the construction of provably good Steiner trees which can be extended from Steiner trees to more general Steiner graphs. At 5 is shown that graphs can be used as preconditioners. Steiner preconditioners implement that theory with the use of multi-way clusterings as well.

According to [4] , given a graph A with n vertices, a Steiner support graph S for A is a graph with n vertices corresponding to the vertices of A and m extra or Steiner vertices. The analysis of their quality can be reduced to the analysis of the support of the pair (A, B) where B is the Schur complement with respect to the Steiner vertices of S.

Quotient and Steiner graphs.

Let P be an edge cut, i.e. a partitioning of the vertices of the graph A into disjoint sets Vi, $i = 1, \ldots, m$. Let T_i be a tree with: (i) leaves corresponding to the vertices in V_i, (ii) root r_i, and (iii) for each $u \in V_i, w(r_i, u)$ is the total incident weight of u in A. We define the quotient graph Q on the set of the roots of the trees T_i, by letting $w(r_i, r_j) = cap(V_i, V_j)$. We define the Steiner graph with respect to P, as $S_P = Q + \sum_{i=1}^{m} T_i$.

The main result of this section is a Theorem that characterizes the support $\sigma(S_P, A)$ with respect to the parameters ϕ,γ of the decomposition P . Before we get there we need to show some Lemmas.

Theorems and Lemmas

Lemma1: If S is a Steiner graph for A and B_S is Schur complement with respect to the elimination of the Steiner vertices of S, we have

$$\sigma(B_S, A) = max_x min_y \frac{\frac{x!}{y!(x-y)!}^T}{x^T A x} \tag{5.3}$$

where $y \in \mathbb{R}^m$, and x is orthogonal to the constant vector.

Lemma2: Steiner support transitivity Let S, S be Steiner graphs for A, with the same number of vertices. Also, let B_S, B_Sbe the Schur complements with respect to the elimination of the Steiner vertices of S, S .

$$\sigma(B_S, A) \leq \sigma(S', S)\sigma(B_{S'}, A) \tag{5.4}$$

Lemma3: Star complement support Let A be a graph with n vertices of volumes $\alpha_1 \leq \cdots \leq \alpha_n$ and S be the star graph with n edges corresponding to the vertices of A. Assume that $\forall i \leq n-1$ the weight c_i of the i^{th} edge of S satisfies $c_i \leq \gamma^{-1}\alpha_i$. Then if $c_n \leq \gamma^{-1}\alpha_n$ or $\alpha_n \geq \sum_{k \leq n-1} \alpha_k$ we have $\sigma(S, A) \leq \frac{2}{\gamma\phi_A^2}$, where ϕ_A the conductance of A.

Theorem: If P is a (ϕ,γ) decomposition of A then $\sigma(S_P, A) \leq 3(1 + \frac{2}{\gamma\phi^2})$. If P is a [ϕ,ρ] decomposition of A then $\sigma(S_P, A) \leq 3(1 + \frac{2}{\phi^3})$.

From steiner preconditioners to multigrid.

Algebraically, any of the classic preconditioned iterative methods, such as the Jacobi and GS iteration, is nothing but a matrix S, which gets applied implicitly to the current error vector e, to produce a new error vector $e' = Se$. For example, in the Jacobi iteration we have $S = (I - D^{-1}A)$. This has the effect that it reduces effectively only part of the error in a given iterate, namely the components that lie in the low eigenspaces of S.

The main idea behind CMG is that the current smooth residual error r = b - Ax, can be used to calculate a correction $R^T Q^{-1} Rr$, where Q is a smaller graph and R is an $m \times n$ restriction operator. The correction is then added to the iterate x. The hope here is that for smooth residuals, the low-rank matrix $R^T Q^{-1} R$ is a good approximation of A^{-1}. Algebraically, this correction is the application of the operator $T = (I - R^T Q^{-1} RA)$ to the error vector e. The choice of Q is most often not independent from that of R. Also,

$$Q = RAR^T \tag{5.5}$$

Steiner preconditioners, introduced in 3.4 introduce external nodes into preconditioners. CMG is based on a partitioning of the n vertices in V into m vertex-disjoint clusters V_i. For each V_i, the preconditioner contains a star graph S_i with leaves corresponding to the vertices in V_i rooted at a vertex r_i. The roots r_i are connected and form the quotient graph Q.

5.5 Multigrid Preconditioning.

An MG method, mentioned at 3.2 with an intentionally reduced tolerance can be used as an efficient preconditioner for an external iterative solver. The solution may still be obtained in time as well as in the case where the multigrid method is used as a solver. Multigrid preconditioning can be used in linear systems.

Generalized multigrid methods.

Multigrid methods can be generalized in many different ways. They can be applied naturally in a time-stepping solution of parabolic PDEs, or they can be applied directly to time-dependent PDEs. Multigrid methods can also be applied to integral equations, or for problems in statistical physics.

Other extensions of multigrid methods include techniques where no PDE nor geometrical problem background is used to construct the multilevel hierarchy. Such algebraic multigrid methods (AMG) construct their hierarchy of operators directly from the system matrix, and the levels of the hierarchy are simply subsets of unknowns without any geometric interpretation. Thus, AMG methods become true black-box solvers for sparse matrices. However, AMG is regarded as advantageous mainly where geometric multigrid is too difficult to apply.

CMG method.

Combinatorial Multigrid (CMG), is a variant of Algebraic Multigrid (AMG) providing strong convergence guarantees for SDD linear systems. It is fundamental the potential for immediate impact on the design of industrial strength code for important applications. In contrast to AMG, CMG offers strong convergence guarantees for the class of SDD matrices and under

certain conditions for the even more general class of symmetric M-matrices. The convergence guarantees are based on recent progress in spectral graph theory and combinatorial preconditioning 5.3. At the same time, linear systems from these classes play an increasingly important role in various fields [2].

MG algorithms are commonly used as preconditioners to other iterative methods, as a result the main advantage of the CMG solver comparing to other iterative methods is that it can be used as a preconditioner to CG.

CMG Description and Parallel Implementation Details.

At a high level, the key idea behind CMG is that the provably small condition number $\kappa(A, B)$, where B is given in 4.3, is equal to the condition number $\kappa(\hat{A}, \hat{B})$. We have the two-level algorithm, which is described in the appendix and can be extended into a full multigrid algorithm, by recursively calling the algorithm when the solution to the system with Q is requested. This produces a hierarchy of graphs. The full multigrid algorithm, after simplifications in the algebra of the two-level scheme is described in the appendix as well.

More detailed, the CMG algorithm consists of the setup phase which computes a multigrid hierarchy, and the solve phase. The setup phase constructs a hierarchy of SDD matrices $A = A_0, \ldots, A_i$. As with most variants of AMG, CMG uses the Galerkin condition 5.5 to construct the matrix A_{i+1} from A_i. This amounts to the computation of a restriction operator $R_i \in R^{dim(A_i)dim(A_{i+1})}$, and the construction of A_{i+1} via the relation $A_{i+1} = R_i^T A_i R_i$. CMG constructs the restriction operator R_i by grouping the variables/nodes of A_i into $dim(A_{i+1})$ disjoint clusters and letting $R(i, j) = 1$ if node i is in cluster j, and $R(i, j) = 0$ otherwise. This simple approach is known as aggregate-based coarsening, and it has recently attracted significant interest due to its simplicity and advantages for parallel implementations. Classic AMG constructs more complicated restriction operators that can be viewed as (partially) overlapping clusters. The main difference between CMG and other AMG variants is the algorithm for clustering, which in the CMG case is combinatorially rather than algebraically driven. The running time of the CMG setup phase is negligible comparing to the actual MG iteration.

The solve phase of CMG , which is dominated by Sparse Matrix-Vector multiplication (SpMV) operations, is quite similar to the AMG solve phase. the pseudo- code is given in the appendix. When $t_i = 1$, the algorithm is known in the MG literature as the V-cycle, while when $t_i = 2$, it is known as the W-cycle. It has been known that the aggregate- based AMG does not exhibit good convergence for the V-cycle. More complicated cycles are expected to converge fast, without blowing up the total work performed by the algorithm. t_i is as follows:

$$t_i = max \left\{ \lceil \frac{nnz(A_i)}{nnz(A_i + 1)} - 1 \rceil, 1 \right\}. \tag{5.6}$$

where, $nnz(A)$ denotes the number of nonzero entries of A. This choice for the number of recursive calls, combined with the fast geometric decrease of the matrix sizes, targets a geometric decrease in the total work per level.

In the parallel Matlab implementation, CMG solve phase has been optimized by using different SpMV implementations for different matrix sizes. When the matrix size is larger than 1K, is used the blocked version of SpMV, and when it is smaller than that, it is resorted to the plain parallel implementation, where the matrix is stored in full and each row can be computed in parallel. The reason is that the blocked version of SpMV has higher overhead

than the simple implementation for smaller matrices.

The CMG solve phase is the implicit inverse of a symmetric positive operator B. The condition number $\kappa(A, B)$ can therefore be defined, and it characterizes the rate of convergence of the preconditioned CG iteration. The CMG core works with the assumption that the system matrix A is SDD.

In summary, CMG exploits both symmetry and precision reduction, in applications that are well suited for the diagonal hierarchical blocking approach, where a decomposition of a matrix into a tree of submatrices occurs. Moreover, the performance improvements we see for CMG are expected to carry over to other flavors of multigrid.

The CMG algorithm is described at the Appendix 6.2 . At 6.2 is also presented the structure of the Spice-like simulator including the CMG preconditioner.

Chapter 6

Results and conclusions

In this chapter is made a comparative analysis between CMG preconditioner and incomplete cholesky preconditioner.

6.1 Results

IBM Power Grid Benchmarks for DC Analysis

i for current source

n for nodes (total number, does not take shorts into account)

r for resistors (include shorts)

s for shorts (zero value resistors and voltage sources)

v for voltage sources (include shorts). 0 for SPD system

IBM Power Grid Benchmarks										
DC Analysis-tolerance 1e-3										
Name	i	n	r	s	v	l	CMG time	CMG iter	incChol time	incChol ite
ibmpg2	37926	127238	208325	1298	0	5	1.413250	20	32.606550	1284
ibmpg4	276976	953583	1560645	11682	0	6	5.361745	17	508.570326	2254
ckt1	1.2	6.29	12.57	0	0	5	36.9	40	640	850

Table 6.1: IBM Power Grid Benchmark for DC Analysis with tolerance 1e-3

IBM Power Grid Benchmarks										
DC Analysis-tolerance 1e-6										
Name	i	n	r	s	v	l	CMG time	CMG iter	incChol time	incChol ite
ibmpg2	37926	127238	208325	1298	0	5	1.119179	32	39.246141	1485
ibmpg4	276976	953583	1560645	11682	0	6	9.756959	27	582.352378	2500
ckt1	1.2	6.29	12.57	0	0	5	56.6	62	1054	1400

Table 6.2: IBM Power Grid Benchmark for DC Analysis with tolerance 1e-6

l for metal layers

6.2 Graph vertices, density and reduction

IBM Power Grid Benchmark number 2

Level 1: Graph vertices:127235
Level 2. Graph vertices: 42599
Density : 5.11413
Reduction factor in vertices : 2.98681
Reduction factor in edges : 2.49652
****
Level 3. Graph vertices: 10582
Density : 5.38896
Reduction factor in vertices : 4.02561
Reduction factor in edges : 3.82031
****
Level 4. Graph vertices: 2572
Density : 6.23639
Reduction factor in vertices : 4.11431
Reduction factor in edges : 3.55524
****
Level 5. Graph vertices: 676
Density : 6.38462
Reduction factor in vertices : 3.80473
Reduction factor in edges : 3.7164
****
Level 6. Graph vertices: 161
Density : 6.10559
Reduction factor in vertices : 4.19876
Reduction factor in edges : 4.39064
****

IBM Power Grid Benchmark number 3

Level 2. Graph vertices: 283993
Density : 5.3102
Reduction factor in vertices : 2.9986
Reduction factor in edges : 2.42311
****
Hierarchy constructed in 0.260000 seconds
Level 3. Graph vertices: 82313
Density : 5.4089
Reduction factor in vertices : 3.45016
Reduction factor in edges : 3.3872
****
Level 4. Graph vertices: 22790

Density : 5.58605
Reduction factor in vertices : 3.6118
Reduction factor in edges : 3.49727
****
Level 5. Graph vertices: 5832
Density : 6.21399
Reduction factor in vertices : 3.90775
Reduction factor in edges : 3.51286
****
Level 6. Graph vertices: 1339
Density : 6.53249
Reduction factor in vertices : 4.35549
Reduction factor in edges : 4.14313
****
Level 7. Graph vertices: 343
Density : 6.34111
Reduction factor in vertices : 3.90379
Reduction factor in edges : 4.02161
****
Hierarchy constructed in 1.440000 seconds

IBM Power Grid Benchmark number 4

Level 1: Graph vertices:953580
Level 2. Graph vertices: 325447
Density : 5.63642
Reduction factor in vertices : 2.93006
Reduction factor in edges : 2.21164
****
Level 3. Graph vertices: 99139
Density : 5.29665
Reduction factor in vertices : 3.28273
Reduction factor in edges : 3.49331
****
Level 4. Graph vertices: 24226
Density : 6.04408
Reduction factor in vertices : 4.09226
Reduction factor in edges : 3.58619
****
Level 5. Graph vertices: 5894
Density : 6.54496
Reduction factor in vertices : 4.11028
Reduction factor in edges : 3.79573
****
Level 6. Graph vertices: 1477
Density : 6.63981
Reduction factor in vertices : 3.99052

Reduction factor in edges : 3.93352
****

Level 7. Graph vertices: 363
Density : 6.43251
Reduction factor in vertices : 4.06887
Reduction factor in edges : 4.2
****

Hierarchy constructed in 1.550000 seconds

IBM Power Grid Benchmark number 5

Level 2. Graph vertices: 515791
Density : 5.07972
Reduction factor in vertices : 3.13716
Reduction factor in edges : 2.26217
****

Level 3. Graph vertices: 148103
Density : 6.43242
Reduction factor in vertices : 3.48265
Reduction factor in edges : 2.75027
****

Level 4. Graph vertices: 32774
Density : 5.96876
Reduction factor in vertices : 4.51892
Reduction factor in edges : 4.86996
****

Level 5. Graph vertices: 6996
Density : 6.0223
Reduction factor in vertices : 4.68468
Reduction factor in edges : 4.64303
****

Level 6. Graph vertices: 199
Density : 2.15578
Reduction factor in vertices : 35.1558
Reduction factor in edges : 98.2098
****

Hierarchy constructed in 2.390000 seconds

Bibliography

[1] Richard Barrett. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. siambooks.

[2] Guy E. Blelloch, Ioannis Koutis, Gary L. Miller, and Kanat Tangwongsan. Hierarchical diagonal blocking and precision reduction applied to combinatorial multigrid. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–12, Washington, DC, USA, 2010. IEEE Computer Society.

[3] Ioannis Koutis and Gary L. Miller. Graph partitioning into isolated, high conductance clusters: theory, computation and applications to preconditioning. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, SPAA '08, pages 137–145, New York, NY, USA, 2008. ACM.

[4] Ioannis Koutis, Gary L. Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Comput. Vis. Image Underst.*, 115(12):1638–1646, December 2011.

[5] Dilip Krishnan and Richard Szeliski. Multigrid and multilevel preconditioners for computational photography. *ACM Trans. Graph.*, 30(6):177:1–177:10, December 2011.

[6] Bruce M. Maggs, Gary L. Miller, Ojas Parekh, R. Ravi, and Shan Leung Maverick Woo. Finding effective support-tree preconditioners. In *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '05, pages 176–185, New York, NY, USA, 2005. ACM.

[7] Gary L. Miller and Peter C. Richter. Lower bounds for graph embeddings and combinatorial preconditioners. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '04, pages 112–119, New York, NY, USA, 2004. ACM.

[8] Farid N. Najm. *Circuit Simulation*. JOHN WILEY and SONS, 2010.

[9] Yousef Saad. *Iterative Methods fos Sparse Linear Systems*. Second edition, 2003.

[10] Markus Wagner, Karl Rupp, and Josef Weinbub. A comparison of algebraic multigrid preconditioners using graphics processing units and multi-core central processing units. In *Proceedings of the 2012 Symposium on High Performance Computing*, HPC '12, pages 2:1–2:8, San Diego, CA, USA, 2012. Society for Computer Simulation International.

[11] Xueqian Zhao and Zhuo Feng. Towards efficient spice-accurate nonlinear circuit simulation with on-the-fly support-circuit preconditioners. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 1119–1124, New York, NY, USA, 2012. ACM.

Appendix

Overall Circuit Simulation Flow

Algorithms

In the appendix are presented the basic algorithms that have been mentioned in the previous sections and are implemented in the thesis.

Algorithm 1 Preconditioned Conjugate Gradient (PCG)

$r^{(0)} = b - Ax^{(0)}$ `for some initial guess` $x^{(0)}$
for $i = 1, 2, \ldots$ **do**
  `solve` $Mz^{i-1} = r^{i-1}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $(i = 1)$ **then**
    $p^{(1)} = z^{(0)}$
  else
    $\beta^{i-1} = \frac{\rho^{i-1}}{\rho^{i-2}}$
    $p^{i-1} = z^{i-1} + \beta_{i-1} p^{i-1}$
  end if
  $q^{(i)} = Ap^{(i)}$
  $\alpha_i = \frac{\rho_{(i-1)}}{{p^{(i)}}^T} q^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_{(i)} q^{(i)}$
  `check convergence; continue if necessary`
end for

Code implementation structure

Input: Initial time t_0 and final time T.
{Perform `DC analysis` at t_0:}
Disable all dynamic elements.
Choose an initial candidate solution at t_0.
while ($x(t_0)$ has not converged and not timed-out) **do** {Newton loop}
For every (resistive) element in the network, evaluate its
linearized element stamp and construct the MNA system.
Solve the resulting MNA system using LU factorization.
endwhile
if (Newton loop has timed-out) **then**
Abort! {Unable to find initial DC solution}
endif

{Perform `Transient Analysis`:}
Reinstate all dynamic elements.
Set $n = 0$, and choose an initial time-step h_1.
while ($t_n < T$) **do** {Time discretization loop}
$t_{n+1} = t_n + h_{n+1}$
Use $x(t_n)$ as the initial candidate solution at t_{n+1}.
while ($x(t_{n+1})$ has not converged and not timed-out) **do** {Newton loop}
For every element in the network, evaluate its discretized,
linearized element stamp and construct the MNA system.
Solve the resulting MNA system using LU factorization.
endwhile
if (Newton loop has timed-out) **then** {Need to reduce the time-step}
$h_{n+1} = h_{n+1}/2$
else
Compute the LTE at t_{n+1}.
if (LTE is too large) **then** {Need to reduce the time-step}
$h_{n+1} = h_{n+1}/2$
else {Accept the solution $x(t_{n+1})$ and move forward in time}
$n = n + 1$
if (LTE is too small) **then** {We can increase the time-step}
$h_{n+1} = 2h_n$
else {Keep the same time-step}
$h_{n+1} = h_n$
endif
endif
endif
if (h_{n+1} is too small) **then**
Abort! {Time-step too small}
endif
endwhile

Figure 1: Overall Circuit Simulation Pseudo-Code.

Algorithm 2 Preconditioned BiCG

$r^{(0)} = b - Ax^{(0)}$ `for some initial guess` $x^{(0)}$
`choose` $\tilde{r}^{(0)}$ `for example:` $\tilde{r}^{(0)} = r^{(0)}$
for $i = 1, 2, \ldots$ **do**
  `solve` $Mz^{i-1} = r^{i-1}$
  `solve` $M^T \tilde{z}^{i-1} = \tilde{r}^{i-1}$
  $\rho_{i-1} = \tilde{r}^{(i-1)^T} z^{(i-1)}$
  if $(i = 1)$ **then**
    $p^{(1)} = z^{(0)}$
    $\tilde{p}^{(1)} = \tilde{z}^{(0)}$
  else
    $\beta^{i-1} = \frac{\rho^{i-1}}{\rho^{i-2}}$
    $p^{i-1} = z^{i-1} + \beta_{i-1} p^{i-1}$
    $\tilde{p}^{i-1} = \tilde{z}^{i-1} + \beta_{i-1} \tilde{p}^{i-1}$
  end if
  $q^{(i)} = Ap^{(i)}$
  $\tilde{q}^{(i)} = A^T \tilde{p}^{(i)}$
  $\alpha_i = \frac{\rho_{(i-1)}}{\tilde{p}^{(i)^T}} q^{(i)}$
  $x^{(i)} = x^{(i-1)} - \alpha_{(i)} p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_{(i)} q^{(i)}$
  $\tilde{r}^{(i)} = \tilde{r}^{(i-1)} - \alpha_{(i)} \tilde{q}^{(i)}$
  `check convergence; continue if necessary`
end for

Algorithm 3 Two-level CMG

Input: Laplacian A=(V,E,w),vector b, approximate solution x, $n \times m$ restriction matrix R

Output: Updated solution x for Ax = b
1. $D : diag(A); \hat{A} := D^{-1/2} A D{-}1/2$
2. $z := (I - \hat{A}) D^{-1/2} x + D^{-1/2} A D^{-1/2}$
3. $r := D^{-1/2} b - \hat{A} z; w := RD^{1/2}$
4. $Q := RAR^T; SolveQy = w;$
5. $z := z + D^{1/2} R^T y;$
6. $x := D^{-1/2}((I - \hat{A})z + D^{-1/2} b)$

Algorithm 4 CMG

function $x := CMG(A_i, B_i)$

1. $D : diag(A)$;
2. $x := D^{-1}b$
3. $r_i := b_i - A_i D^{-1}b$
4. $b_{i+1} := Rr_i$
5. $z := CMG(A_{i+1}, B_{i+1})$

for $i = 1, t_i - 1$ **do**

7. $r_{i+1} := b_{i+1} - A_{i+1}z$
8. $z := z + CMG(A_{i+1}, r_{i+1})$

end for

10. $x := x + R^T z$
11. $x := r_i - D^{-1}(A_i x - b)$

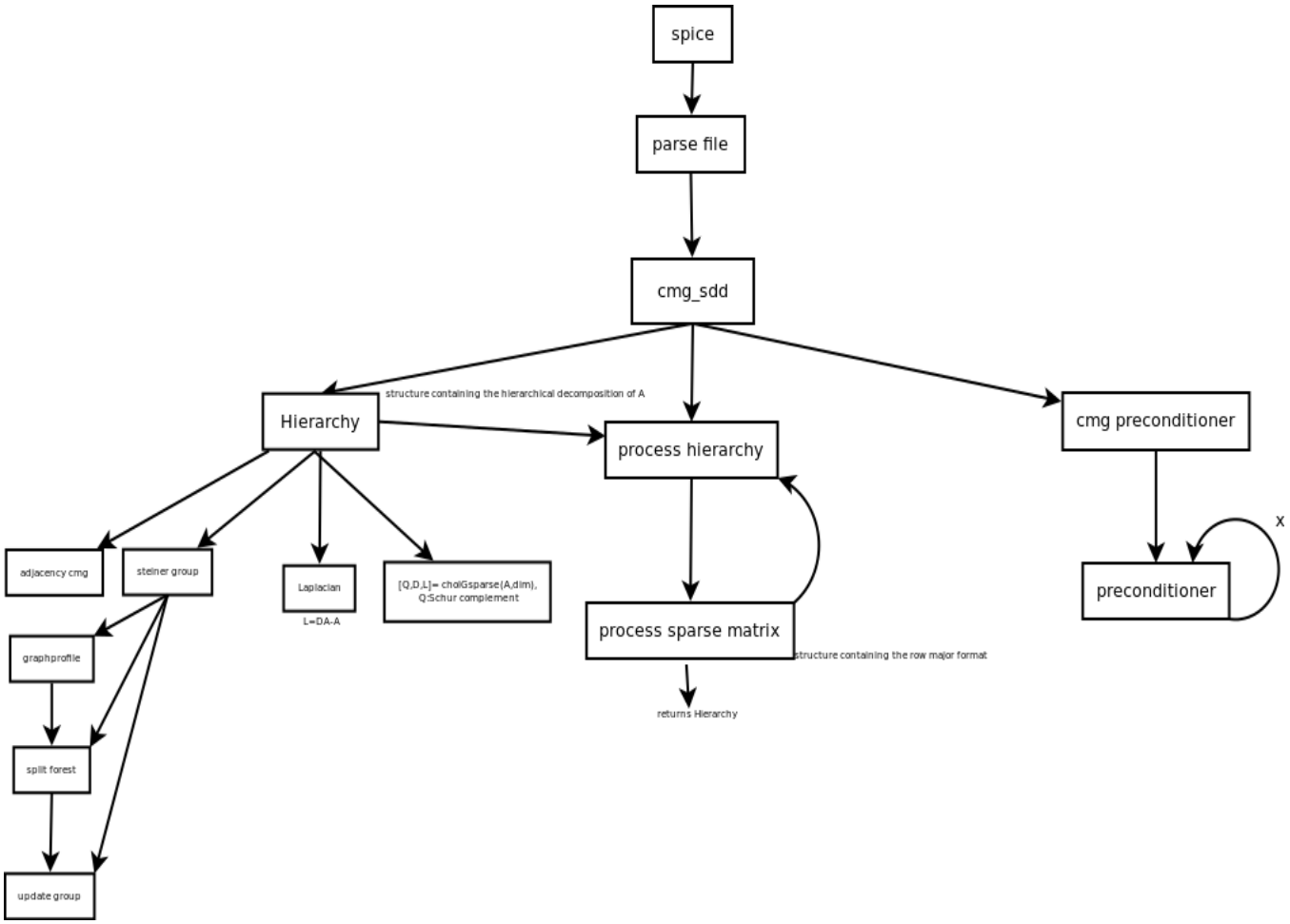

Figure 2: Structure of preconditioner.