Master's Thesis

Parallel Deflated CG Method to Model Groundwater Flow in a Layered Grid

Raju Ram

to obtain the degree of Master of Science
at the Delft University of Technology,
Defended publicly on Thursday August 24, 2017 at 10:00 AM.

Student number:	4592476	
Project duration:	October 2016 – August 2017	
Thesis committee:	Prof. Dr. Ir. Kees Vuik,	TU Delft Supervisor
	Ir. Jarno Verkaik,	Deltares Supervisor
	Prof. Dr. Ir. Hai-Xiang Lin,	TU Delft

24 August, 2017

TU Delft Delft University of Technology

Deltares Enabling Delta Life

Contents

Abstract

Groundwater, present beneath the earth's surface in soil pore spaces, is the primary source of fresh water that we use in day to day life. Hydrologists at Dutch research institute Deltares are developing large groundwater models to support water managers in their decision-making process. For example, these models simulate effects such as water availability during periods of drought. These models use a Deltares accelerated version of MODFLOW called iMODFLOW.

Together with the United State Geological Survey (USGS), Deltares has developed the Parallel Krylov Solver (PKS) package, which has recently been incorporated into iMODFLOW. It was observed that for the larger number of subdomains the Preconditioned Conjugate Gradient (PCG) solver in PKS deteriorates the number of iterations.

We have implemented the deflation preconditioner with constant and linear deflation vectors in the PCG solver. These vectors approximate the eigenvectors that are slowing down convergence. The groundwater simulation time can be reduced by a factor of 4 in iMODFLOW. This speed up is achieved due to a decrease in PCG iterations. The iteration drop is highest using linear deflation vectors.

Preface

This Master's thesis is written for the degree of Master of Science in Applied Mathematics, faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology for the Joint Masters Program, Computer Simulation for Science and Engineering (COSSE). Friedrich-Alexander-Universität (FAU) Erlangen Germany was the first university in the COSSE program. This Master's project has been carried out at Groundwater Management Department of Dutch Research Institute Deltares by spending 5 months in Delft and the remaining 6 months in Utrecht.

Before I go on with the report, I would like to thank my Deltares supervisor Jarno Verkaik and TU Delft supervisor Kees Vuik for excellent supervision. Kees showed me correct research direction from the beginning and had an answer to almost all of my questions. Jarno helped me in parallel FORTRAN coding and kept me focused on the linear deflation implementation. Jarno's master project done 13 years ago gave me the ideas to work on my project. I have seen a tremendous growth in my professional life such as proper planning and communication in the meetings. I am grateful to Edwin Sutanudjaja who arranged 5000 System Billing Unit (SBU) on Cartesius at the crucial time of my project. I had a great discussion with Timo Kroon regarding the business side of my research. I am grateful to online latex editor Overleaf, on which this thesis was compiled.

The courses taken at FAU Erlangen gave me a strong background in parallel numerical linear algebra. I am thankful to Prof. Gerhard Wellein and Prof. Ulrich Rüde and Dr. Harald Köstler for the motivation to do research in this field.

I would like to thank Utrecht and Delft Art of Living family who provided me a home away from home. I am grateful to have a friend like Nico who took care of me in Utrecht like a big brother: be it cooking or yoga. Vikrant's family made sure that I do not miss Indian food. Divaye helped me in the last days of my project to stay calm. Special thanks go to my teacher Sri Sri Ravi Shankar who propelled my growth as a person and provided me the stimulation and opportunity to enhance my skills. I would like to thank my lunch buddies, all the interns at Deltares for fun and laughs during tough times. Tobias gave a good feedback on the report and background information about Hydrology.

Lastly, I would like to thanks my parents and family members who allowed me to study abroad.

Raju Ram
Delft, August 2017

1

Introduction

1.1. Thesis Objective

Water is a vital element in our lives. About 2.5% of total water present on earth is fresh water. The fresh water is present as ice, liquid water, and water vapor. About 70% of the freshwater is present in the form of ice, therefore, is not drinkable. About 98% of the earth's available fresh water is present beneath the earth's surface in soil pore spaces and rocks [16], called groundwater. To understand the effects of projected increased demands on groundwater to supply water, groundwater simulation is getting increasingly important.

Scientists at Dutch research institute `Deltares` are developing large groundwater models for water boards [1], drinking water companies and municipalities to simulate the effects of climate change, such as drought. The simulation code used is MODFLOW written in FORTRAN, and this code is the worldwide standard for groundwater computation. The first version of the MODFLOW has been developed by the United States Geological Survey (USGS) in 1984, and the current core version is MODFLOW 6. It computes the hydraulic head for the groundwater equation representing Darcy flow on a cell-centered finite volume grid. Discretization results into a large system of equations. The problem becomes nonlinear when incorporating the effect of a river or drains since a Cauchy boundary condition is imposed. Therefore, in each outer iteration, a linear system is being solved. Typically, the outer iteration is done by Picard iteration and the inner iteration by Preconditioned Conjugate Gradient (PCG) solver. Deltares has developed an accelerated version of MODFLOW called `iMOD`.

To support the decision makers in addressing hydrological problems, high-resolution models are often needed. These models typically consist of a vast number of computational cells and therefore have significant memory requirements and long run

[1] Regional government bodies charged with managing water barriers, water levels, and water quality.

times. Simulating such large scale models on a serial computer is impractical. Also, solving the system of equation that arises after discretization costs major time in the whole simulation. Therefore, Deltares is developing a new module in iMOD, called Parallel Krylov Solver (PKS) package together with the USGS, Utrecht University and Delft University of Technology. PKS incorporates Message Passing Interface (MPI) and OpenMP as parallel processing paradigms. PKS includes Conjugate Gradient (CG) and Bi-Conjugate Gradient Stabilized (BiCGSTAB) solvers to solve symmetric and non-symmetric Linear system of equations (LSE) respectively. Additive Schwarz (AS) method is used as a preconditioner because it is suitable for parallel computations.

The physical domain is divided into various subdomains, and the computations are carried on these subdomains in parallel. iMOD with PKS package is capable of computing the hydraulic head for high resolution models such as, seven layered Dutch national Nederlands Hydrologisch Instrumentarium (NHI) [13] groundwater model. The computations are carried on a Dutch national super-computer Cartesius [7] (figure 1.1).

Figure 1.1: Dutch National Supercomputer Cartesius

Due to an increase in the number of subdomains, the information about the head from cells in other subdomains takes more iterations to reach the current subdomain. Therefore, we notice an increase in the number of iterations with increasing number of subdomains in the PCG solver [26]. We have implemented the deflation preconditioner with constant and linear deflation vectors in the PKS solver. These vectors approximate the eigenvectors that are slowing down convergence. We denote PCG solver using constant deflation vectors by CDPCG; and using linear and constant deflation vectors by LDPCG.

The primary goal of this master's project research is to improve the performance of the PKS solver by preventing the PCG iteration increase with increasing number

of subdomains and therefore reducing the computational time in PKS enabled iMOD.

1.2. Thesis Outline

The content of the chapters in this report is as follows:

Chapter 2 We give a short introduction of the Geohydrology. Further, we derive the system of equations involving porous media flow. We also describe the PCG solver in MODFLOW.

Chapter 3 We give the scientific background information about the iterative solvers and preconditioners. In particular, we focus on the PCG method.

Chapter 4 We discuss Schwarz domain decomposition methods. In particular, we concentrate on the Additive Schwarz method due to its inherent parallelism.

Chapter 5 We give mathematical theory of the Deflation method and present the algorithm. We suggest two ways to choose the deflation vectors. We also give implementation aspects by illustrating with examples for a toy problem.

Chapter 6 We present results for two test problems: Two dimensional (2D) Poisson problem and iMOD two-layer unit problem.

Chapter 7 We present results for two real case models: Dutch national NHI models with cell size 250 m, 100 m and 50 m; and California miamore model with cell size 100 m and 50 m.

Chapter 8 We conclude from the master's project research. We provide further recommendations, open questions and propose directions of future research.

2

Problem Description

2.1. Introduction

In this chapter, we describe the problem, which we want to address and improve in our research. We start by presenting the theory and mathematics to model the Ground water flow (GWF) in section 2.2 and 2.3. In section 2.4 and 2.5, we define how the system of equations is formed in the transient and steady-state simulation. In the last sections, we give an overview of MODFLOW code which is a worldwide standard for groundwater computation. The models used in groundwater computation are three dimensional. However, we formulate the system of equations in section 2.4 and 2.5 for two-dimensional models for the convenience of the reader.

2.2. Hydrology Background

Water is one of our most valuable natural resources. About 70 percent of the human body is water. The bodies of all plants and animals contain water. Therefore, it is difficult to imagine life without water. Nature limits the supply of water available for our use. Although there is plenty of water on earth, it is not always in the right place, at the right time and of the good quality. Chemical waste produced from factories make the water polluted. To understand the complex water systems of the Earth and address water related problems, Hydrology has evolved as a science. Hydrologists play a significant role in finding solutions to societal water related problems by applying scientific knowledge and mathematical principles.

There are two main sources of water: surface water and groundwater. Surface water is found in lakes, rivers, and reservoirs. Groundwater lies under the surface of the land, where it travels through and fills openings in the rocks/sediments. The sediments that store and transmit the groundwater are called aquifers. Typically, the groundwater moves very slowly (a few meters a year). As it moves, the groundwater is naturally purified – pollutants and harmful bacteria are removed. This phenomenon creates good quality water that is suitable for the preparation

2

of drinking water. Groundwater is often cheaper, more convenient and less vulnerable to pollution than the surface water. Therefore, it is mainly used for public water supplies. Groundwater is also often withdrawn for agricultural, municipal, and industrial use by constructing and operating extraction wells. In the Netherlands, the groundwater level varies from 0.5 to 1.0 meter below the surface in the western parts of the land; in the higher areas (eastern side) from 1.0 to 20.0 meter.

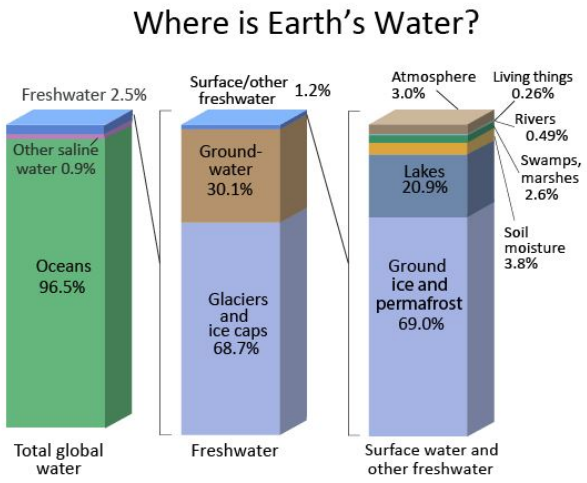


Figure 2.1: Distribution of Earth's Water taken from [5]

In nature, the analysis of virtually every physical process involves a potential gradient. For example, an electrical current flows through circuits from higher voltages to lower. The similar concept can be defined for water flow: there is a potential gradient that determines the direction of flow. Hydraulic head determines this potential gradient. The hydraulic head is a measurement of the total mechanical energy per weight of the groundwater flow system. It is a fundamental component of Darcy's law (defined in the section 2.3.1) which describes fluid flow through porous media. It is usually measured as a liquid surface elevation, expressed in units of length (in meters). The equation for hydraulic head (h) has three components: elevation head and pressure head and kinetic energy head defined below:

$$h = z + \frac{P}{\rho g} + \frac{v^2}{2g} \tag{2.1}$$

where:
z = elevation of the fluid above a reference elevation (L)
P = pressure ($ML^{-1}T^{-2}$)
ρ = density of fluid (ML^{-3})
g = acceleration of gravity (LT^{-2})

Since the groundwater flow is very slow, hence practically, the contribution of kinetic energy is negligible in pressure head calculation. Therefore, the equation 2.1 reduces to:

$$h = z + \frac{P}{\rho g} \tag{2.2}$$

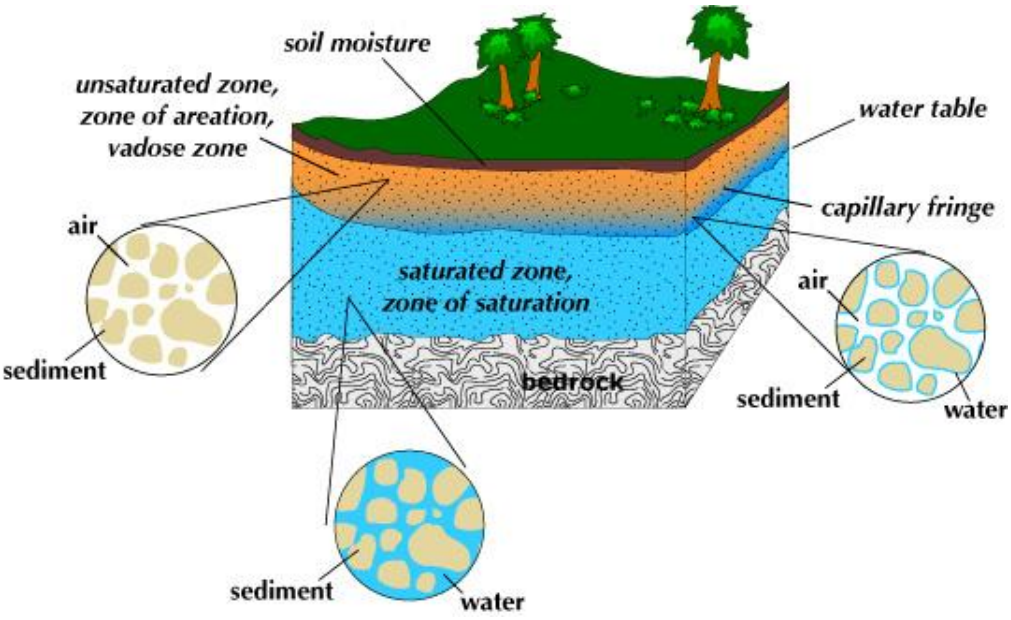


Figure 2.2: Illustration of different zones in hydrology beneath earth's surface

In the figure 2.2, we illustrate the geometries of various zones present beneath earth's surface. Groundwater (the blue area) is the water present in soil pore spaces surrounding the sediments in a saturated zone. The cross section showing the sediment and water is an `aquifer`. The depth at which soil pore spaces and voids in the sediment become completely saturated with water is called a `water table`. The `saturation zone` (the blue area), is the area in an aquifer, below the water table, in which relatively all pore spaces are saturated with water. The unsaturated zone (the yellow area) is the portion of the subsurface above the groundwater table. The soil and rock in this region contain air as well as water in its pores. Under the unsaturated and saturated zone is a solid rock deposit called bedrock. Cross-section of the figure 2.2, representing mainly the groundwater flow is given in figure 2.3.

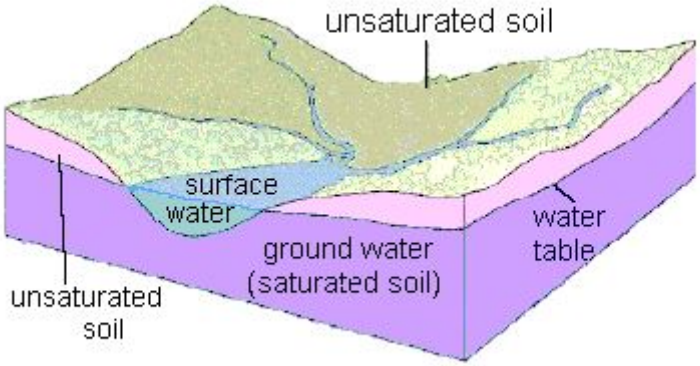


Figure 2.3: A cross section of figure 2.2 taken from USGS [5]

2.3. Mathematical Model

In this section, we define Darcy's law which is the basic principle governing the flow of the fluid. Then, we describe the continuity equation and the groundwater flow equation.

2.3.1. Darcy's Law

Darcy's law defines the movement of water in the subsurface. It is an equation that represents the ability of a fluid to flow through a porous medium such as rock. It relies on the fact that the amount of flow between two points is directly related to the difference in pressure between the points, the distance between the points, and the inter-connectivity of flow pathways in the rock between the points. The Darcy's law was formulated by a French Engineer Henry Darcy based on the results of experiments on the flow of water through beds of sand in $1855 - 56$.

Let a and b be center of cell A and B. Let h_a and h_b be hydraulic head at a and b respectively, and let l be the distance between a and b. Darcy's law is defined as

$$q = -KA\frac{(h_b - h_a)}{l} \quad \text{or} \qquad Q = -K(h_b - h_a) = -K\Delta h \tag{2.3}$$

where

q is the volumetric flow rate (L^3T^{-1}) from cell A to cell B;

Q is the the specific discharge (LT^{-1});

K is the hydraulic conductivity (LT^{-1});

A is the cross sectional area (L^2);

$(h_a - h_b)/l$ is the hydraulic gradient, the change in head over the length of interest.

2.3.2. Continuity Equation

Continuity equation states that the sum of all flows into and out of the cell must be equal to the rate of change in storage within the cell. We assume that the density of ground water is constant. Let there be N total flow contributions into a cell (i,j) . The continuity equation expressing the balance of flow for a cell is defined by

$$\sum_{n=1}^{N} q_{i,j,n} = S_s \Delta V \frac{\Delta h}{\Delta t} \tag{2.4}$$

where

$q_{i,j,n}$ is a flow rate into cell (i,j) from n^{th} source ;

S_s is the volume of water that can be injected per unit volume of aquifer material per unit change in head. It is also known as specific storage (L^{-1});

ΔV be the volume of the cell (i,j) (L^3);

Δh is the change in head over a time interval of length Δt.

2.3.3. Groundwater Flow Equation

The movement of ground water of constant density through porous earth material can be described by the partial differential equation (PDE):

$$-\nabla \cdot Q + W = S_s \frac{\partial h}{\partial t} \tag{2.5}$$

where

W is a volumetric flux per unit volume representing sources and/or sinks of water, with W<0.0 for flow out of the ground-water system, and W>0.0 for flow into the system (T^{-1});

Q is flux across the cross section area (LT^{-1});

h is the hydraulic head (L); and t is time (T).

S_s Refer to the subsection 2.3.2.

In the above equation, h and Q are unknown. We eliminate Q using equation 2.3 and rewrite equation 2.5 as:

$$\nabla \cdot (K \Delta h) + W = S_s \frac{\partial h}{\partial t} \tag{2.6}$$

Expanding the divergence operator in equation 2.6, it can be written as

$$\frac{\partial}{\partial x}\left(K_{xx}\frac{\partial h}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_{yy}\frac{\partial h}{\partial y}\right) + \frac{\partial}{\partial z}\left(K_{zz}\frac{\partial h}{\partial z}\right) + W = S_s \frac{\partial h}{\partial t} \tag{2.7}$$

where, K_{xx}, K_{yy} and K_{zz} are values of hydraulic conductivity along the x, y, and z coordinate axes (LT^{-1}).

The steady state groundwater flow (Storage term S_s is zero) is given as

$$\frac{\partial}{\partial x}\left(K_{xx}\frac{\partial h}{\partial x}\right)+\frac{\partial}{\partial y}\left(K_{yy}\frac{\partial h}{\partial y}\right)+\frac{\partial}{\partial z}\left(K_{zz}\frac{\partial h}{\partial z}\right)+W=0 \tag{2.8}$$

The proof of above equation can be found in section 2.3 of [29]. All of our numerical experiments in chapter 6 and 7 are done for the steady state groundwater flow equation.

2.4. Discretization

Applying finite volume integration and Gauss's theorem, equation 2.6 can be written as

$$\begin{aligned}\int_V \nabla\cdot(K\Delta h)\,dV+\int_V W\,dV&=\int_V S_s\frac{\partial h}{\partial t}\,dV\\ \int_S (K\Delta h\cdot\vec{n})\,dS+\int_V W\,dV&=\int_V S_s\frac{\partial h}{\partial t}\,dV\end{aligned} \tag{2.9}$$

The surface integrals represents the flux. MODFLOW computes the flux in for each cell. Substituting equation 2.3, equation 2.9 transforms to the following

$$\sum_{n=1}^{N} q_n + W\Delta V = S_s\Delta V\frac{\Delta h}{\Delta t}$$

Now, we illustrate how cells are formed where we can use the flux representing surface integrals. The three-dimensional domain of an aquifer system is discretized spatially using a grid of blocks as shown in the figure 2.4. In this report, we call each block as one cell. The total number of cells in x, y, and z directions are denoted as $NCOL, NROW, NLAY$ respectively. For example, in the figure 2.4, we have NCOL = 9, NROW = 5, and NLAY = 5. Each cell in this grid can be uniquely defined by providing cell index (i, j, k), where $i = 1, 2, \ldots, NROW$; $j = 1, 2, \ldots, NCOL$ and $k = 1, 2, \ldots, NLAY$. Regarding Cartesian coordinates, the index k changes along the vertical, z; because the convention followed in this model is to number layers from the top to down, an increment in the k index corresponds to a decrease in elevation. For example, index k for the top layer is always 1. Similarly, rows would be considered parallel to the x axis, so that increments in the row index, i, would correspond to decreases in y coordinate and columns would be considered parallel to the y axis, so that increments in the column index, j, would correspond to increases in x.

After dividing the domain into cells, one needs to compute the hydraulic head at one point in the cell, called node. We have used a cell centered formulation, in which, nodes are present at the center of each cell. So we compute the hydraulic head at the center of the cell. In the current implementation of MODFLOW, the cell length in x and y direction can be chosen to be different. For simplicity, we choose it to be the same in our numerical experiments. The typical range of cell size is 100 or 250 meter.

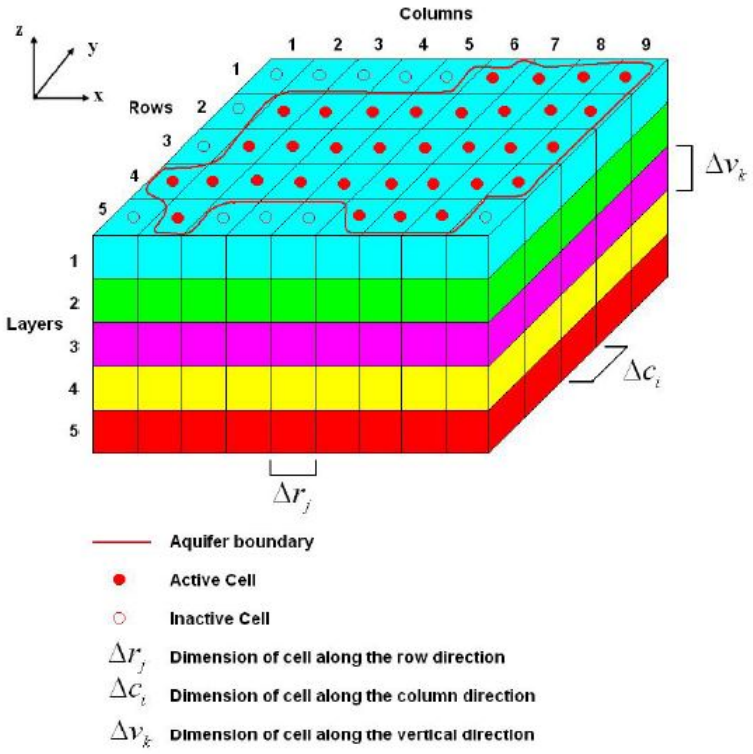


Figure 2.4: Three-dimensional control volume finite difference grid used in MODFLOW taken from [25]

Since mixed derivatives (such as $\partial x \partial y$) are not involved in the equation 2.7. Therefore, in the finite volume stencil, there is no flow contribution into cell (i, j, k) from neighboring diagonal cells. The flow is only from the left-right, top-bottom, and up-down direction. Including the current cell (i, j, k), the total contribution is from 7 cells. Therefore, we obtain a 7 point stencil as given in the figure 2.5.

Now we have all the ingredients ready to define the total groundwater movement into a cell using Darcy's law. However, for simplicity, we only stick to one layer and show the flow into a cell only from left-right and up-down neighboring cells.

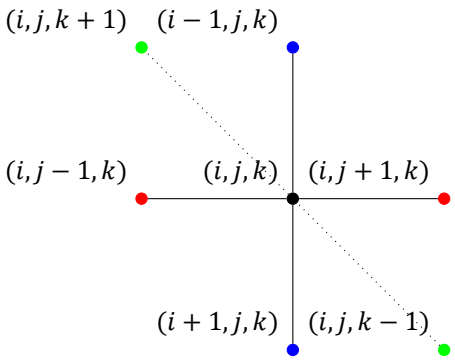


Figure 2.5: 6 adjacent cell nodes surrounding the cell (i,j,k) in MODFLOW

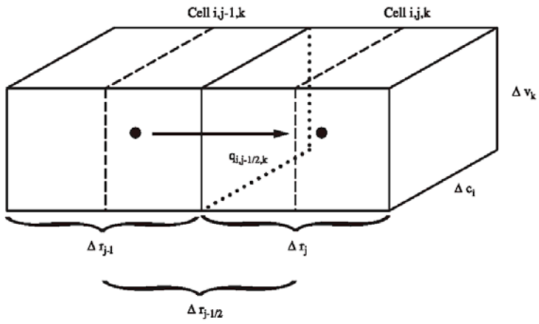


Figure 2.6: Flow from cell $i, j-1, k$ into cell i, j, k

Therefore, we consider the flow into cell (i,j) from its adjacent 4 cells. Flow from cell $(i,j-1)$ to (i,j) is illustrated in figure 2.6. Applying Darcy's law equation 2.3 for figure 2.6, we get

$$q_{(i,j-\frac{1}{2})} = KC_{(i,j-\frac{1}{2})} \Delta c_i \Delta v_k \frac{(h_{i,j-1} - h_{i,j})}{\Delta r_{j-\frac{1}{2}}} \tag{2.10}$$

where

$KC_{(i,j-\frac{1}{2})}$: The hydraulic conductivity along the column between the cell $(i,j-1)$ and cell $(i,j)(LT^{-1})$;

$\Delta c_i \Delta v_k$: The surface area of the plane normal to the direction of flow (L^2);

$\Delta r_{j-\frac{1}{2}}$: Distance between the node of cell (i,j) and $(i,j-1)$.

Note: The index of the row and column in the grid is denoted by i and j respectively. We indicate the conductance along the row by CR and column by CC

respectively.

Eventually, we are interested in the relation between the head and the flow rate, so we define conductance $CC_{(i,j-\frac{1}{2})}$ between nodes $(i,j-1)$ and (i,j) to make the equations similar

$$CC_{(i,j-\frac{1}{2})} = \frac{KC_{(i,j-\frac{1}{2})}\Delta c_i \Delta v_k}{\Delta r_{j-\frac{1}{2}}} \tag{2.11}$$

Analogy of the conductance is given to the resistance in the Ohm's law. Substituting equation 2.11 into equation 2.10, we obtain

$$q_{(i,j-\frac{1}{2})} = CC_{(i,j-\frac{1}{2})}(h_{i,j-1} - h_{i,j}) \tag{2.12}$$

Now we calculate the flow into cell (i,j) from other 3 neighbors using the notation from figure 2.5, and adapting equation 2.12.

$$q_{(i,j+\frac{1}{2})} = CC_{(i,j+\frac{1}{2})}(h_{i,j+1} - h_{i,j}) \tag{2.13}$$

$$q_{(i-\frac{1}{2},j)} = CR_{(i-\frac{1}{2},j)}(h_{i-1,j} - h_{i,j}) \tag{2.14}$$

$$q_{(i+\frac{1}{2},j)} = CR_{(i+\frac{1}{2},j)}(h_{i+1,j} - h_{i,j}) \tag{2.15}$$

Adding equation 2.12-2.15, we can describe the flow into cell (i,j) from 4 neighboring cells as

$$q_{(i,j)} = q_{(i,j-\frac{1}{2})} + q_{(i,j+\frac{1}{2})} + q_{(i-\frac{1}{2},j)} + q_{(i+\frac{1}{2},j)} \tag{2.16}$$

2.5. MODFLOW Packages

The flow is described into a cell (i,j) with the concept of hydrological packages in MODFLOW. There are two types of packages: the first type is the internal flow package, which simulates flow between adjacent cells. The second category is the stress package, which simulates a particular kind of stress (such as rivers, wells, and recharge). The stress packages add terms to the flow equation representing inflows or outflows. Mathematically, these are boundary conditions. So far, we have described flow into cell (i,j) from neighboring cells in equations 2.12-2.15. This task is carried out using an internal flow package in MODFLOW. In this section, we outline the effect of the stress packages. We restrict ourselves to the flow into or out of the cell (i,j) from wells, external sources, rivers and drain.

2.5.1. Effect of Wells

Well adds water to or withdraws water from the aquifer at a constant rate q. Negative values of q are used to indicate well discharge (pumping), whereas positive values of q indicate a recharging well.

If there are more than one well for an aquifer cell (i,j), and one assumes one well recharges to the cell with rate q_1 and other wells pump from the cell with rate q_2, the flow into the cell from the wells may be defined by

$$q_{(i,j)} = q_1 - q_2 \tag{2.17}$$

2.5.2. Effect of External Source

Assume that there is an external source with constant head H_{ext}.The flow into the cell (i,j) from this source depends upon difference between the head in the cell and the head assigned to the external source and may be given by

$$q_{(i,j)} = C_{ext}(H_{ext} - h_{(i,j)}) \tag{2.18}$$

where, C_{ext} is the boundary conductance.

2.5.3. Effect of Rivers

Rivers and streams contribute water to or drain water from the aquifer cell (i,j), depending on the head gradient between the river and the cell.

We assume that, the water level does not drop below the bottom of the riverbed layer. Under this assumption, the flow from the river (with head H_{RIV}) into the cell (i,j) is given by

$$q_{(i,j)} = C_{RIV}(H_{RIV} - h_{(i,j)}) \tag{2.19}$$

where, C_{RIV} is the hydraulic conductance of bed sediment.

If the water level in the aquifer falls below a certain point, seepage from the river no more depends on the head in the aquifer. We denote the bottom of a riverbed by R_{BOT}. The flow through the riverbed layer is given by $C_{RIV}(H_{RIV} - R_{BOT})$. The general equation of the movement of water from the river into the cell (i,j) is given by

$$q_{(i,j)} = \begin{cases} C_{RIV}(H_{RIV} - R_{BOT}), & \text{if } h_{(i,j)} \leq R_{BOT} \\ C_{RIV}(H_{RIV} - h_{(i,j)}), & \text{if } h_{(i,j)} > R_{BOT}. \end{cases} \tag{2.20}$$

We have restricted ourselves by defining the flow into the cell without discussing the physical phenomenon in detail. Interested readers may look in chapter 6 of MODFLOW 2005 manual [25].

2.5.4. Effect of Drain

In some cases such as agricultural drains, it is required to remove water from the aquifer to save the crops. The cutoff head is called drain elevation, denoted by H_{DRN}. The drainage flow into the cell (i,j) (which is negative) is given by

$$q_{(i,j)} = \begin{cases} 0, & \text{if } h_{(i,j)} \leq H_{DRN} \\ C_{DRN}(H_{DRN} - h_{(i,j)}), & \text{if } h_{(i,j)} > H_{DRN}. \end{cases} \tag{2.21}$$

where, C_{DRN} is the drain conductance.

We denote the flow from the source n into cell (i,j) by $q_{(i,j,n)}$. From above boundary conditions, we observe that the inflow rate $q_{(i,j,n)}$ varies linearly with the head. In general, it can be represented by the equation 2.22. If the flow into the cell (i,j) is from the river, we observe from equation 2.20, the coefficient $a_{(i,j,n)}$ depends upon the head $h_{(i,j)}$. It makes the system of equation nonlinear. In MODFLOW, a Picard iteration is used to modify the system of equations 2.30 in the linear form (refer section 2.8).

$$q_{(i,j,n)} = a_{(i,j,n)}h_{(i,j)} + b_{(i,j,n)} \tag{2.22}$$

In general, if there are N external sources with a flow into a single cell, the combined flow is expressed by

$$\sum_{n=1}^{N} q_{(i,j,n)} = \sum_{n=1}^{N} a_{(i,j,n)}h_{(i,j)} + \sum_{n=1}^{N} b_{(i,j,n)} \tag{2.23}$$

We substitute following $A_{(i,j)}$ and $B_{(i,j)}$ into equation 2.23 to yield 2.24

$$A_{(i,j)} = \sum_{n=1}^{N} a_{(i,j,n)} \quad ; \quad B_{(i,j)} = \sum_{n=1}^{N} b_{(i,j,n)}$$

$$\sum_{n=1}^{N} q_{(i,j,n)} = A_{(i,j)}h_{(i,j)} + B_{(i,j)} \tag{2.24}$$

The total flow into cell (i,j) comprises of flow from 4 neighboring cell as given in equation 2.16 and the flow from stress packages given in equation 2.24. Substituting both the equations into continuity equation 2.4, we obtain

$$\begin{aligned} q_{(i,j-\frac{1}{2})} + q_{(i,j+\frac{1}{2})} + q_{(i-\frac{1}{2},j)} + q_{(i+\frac{1}{2},j)} + \\ A_{(i,j)}h_{(i,j)} + B_{(i,j)} = S_s \Delta V \frac{\Delta h}{\Delta t} \end{aligned} \tag{2.25}$$

Substituting equations 2.12-2.15 into 2.25, and replacing ΔV from notations of the figure 2.4, we obtain

$$\begin{aligned} CC_{(i,j-\frac{1}{2})}(h_{i,j-1} - h_{i,j}) + CC_{(i,j+\frac{1}{2})}(h_{i,j+1} - h_{i,j}) + \\ CR_{(i-\frac{1}{2},j)}(h_{i-1,j} - h_{i,j}) + CR_{(i+\frac{1}{2},j)}(h_{i+1,j} - h_{i,j}) + \\ A_{(i,j)}h_{(i,j)} + B_{(i,j)} = S_s(\Delta r_j \Delta c_i \Delta v_k)\frac{\Delta h_{i,j}}{\Delta t} \end{aligned} \tag{2.26}$$

Thus we formulate the system of equations which are to be solved for head values.

2.6. Groundwater Simulation Type

The MODFLOW supports two types of groundwater simulation: a transient and a steady state simulation. In this section, we briefly discuss the formulation of system of equations in both simulation models. However, all the numerical experiments done in the chapter 6 and 7, are done for the steady state model.

2.6.1. Transient State Simulation

The objective of a transient simulation is to compute the head distributions at successive times, given the initial head distribution, the boundary conditions and the hydraulic parameters. We assume that, the head distribution is known to be $h_{(i,j)}^{(m-1)}$ at time $t^{(m-1)}$ and we aim to compute the head distribution at time $t^{(m)}$. We define the head gradient by

$$\frac{\Delta h_{i,j}}{\Delta t} \approxeq \frac{h_{(i,j)}^{(m)} - h_{(i,j)}^{(m-1)}}{t^{(m)} - t^{(m-1)}} \tag{2.27}$$

Substituting equation 2.27 into 2.26 and rearranging the terms, we obtain

$$\begin{aligned} CR_{(i-\frac{1}{2},j)} h_{(i-1,j)}^{(m)} + CC_{(i,j-\frac{1}{2})} h_{(i,j-1)}^{(m)} + (-CR_{(i-\frac{1}{2},j)} - CC_{(i,j-\frac{1}{2})} \\ -CC_{(i,j+\frac{1}{2})} - CR_{(i+\frac{1}{2},j)}) h_{(i,j)}^{(m)} + CC_{(i,j+\frac{1}{2})} h_{(i,j+1)}^{(m)} + CR_{(i+\frac{1}{2},j)} h_{(i+1,j)}^{(m)} + \\ A_{(i,j)} h_{(i,j)}^{(m)} + B_{(i,j)} = S_s(\Delta r_j \Delta c_i \Delta v_k) \frac{h_{(i,j)}^{(m)} - h_{(i,j)}^{(m-1)}}{t^{(m)} - t^{(m-1)}} \end{aligned} \tag{2.28}$$

The initial head distribution provides a value $h_{(i,j)}^{(0)}$ at each cell (i,j). Head distribution at $t^{(1)}$ can be obtained by solving equation 2.28 with $m = 1$. This results into N[1] linear system of equations. In the similar way, we use the head distribution at $t^{(m)}$, and reformulate the LSE to solve for $t^{(m+1)}$. Various iterative methods are used to solve these system of equations, treated in chapter 3 in more detail.

Equation 2.28 can be seen as a formulation of system of equations.

$$A\underline{u} = \underline{f} \tag{2.29}$$

where

A is a known matrix of the coefficients (conductances CR, CC, and CV) of the head at the node of all active cells (define in section 2.7) in the grid.

$\underline{u}$ is an unknown vector of head values at the end of time step m for nodes of active cells in the grid;

$\underline{f}$ is a known vector of the constant terms, RHS, for nodes of active cells in the grid.

[1] Total number of cells in 2D domain in layer k

2.6.2. Steady State Simulation

A steady state simulation is represented by a single stress period having a single time step with the storage term (S_s) set to zero. In a steady state simulation, the sum of all inflows (where the outflow is a negative inflow) from adjacent cells and external processes must be zero for each cell in the model. A steady-state problem requires a single solution of LSE, rather than multiple solutions at multiple time steps. The boundary conditions are also time independent in steady state simulations.

The complexity in equation 2.28 can be reduced by assuming that the time superscript is m unless otherwise shown. This results into

$$\begin{aligned} &CR_{(i-\frac{1}{2},j)}h_{(i-1,j)} + CC_{(i,j-\frac{1}{2})}h_{(i,j-1)} + H_c h_{(i,j)} + \\ &CC_{(i,j+\frac{1}{2})}h_{(i,j+1)} + CR_{(i+\frac{1}{2},j)}h_{(i+1,j)} = RHS_{(i,j)} \end{aligned} \tag{2.30}$$

where,

$$H_c = -CR_{(i-\frac{1}{2},j)} - CC_{(i,j-\frac{1}{2})} - CC_{(i,j+\frac{1}{2})}$$

$$-CR_{(i+\frac{1}{2},j)} + A_{(i,j)} - \frac{S_s(\Delta r_j \Delta c_i \Delta v_k)}{t - t^{(m-1)}}, \quad \text{and}$$

$$RHS_{(i,j)} = -B_{(i,j)} - \frac{S_s(\Delta r_j \Delta c_i \Delta v_k) h_{(i,j)}^{(m-1)}}{t - t^{(m-1)}}.$$

The entire system of equations 2.30 can be written in matrix form as

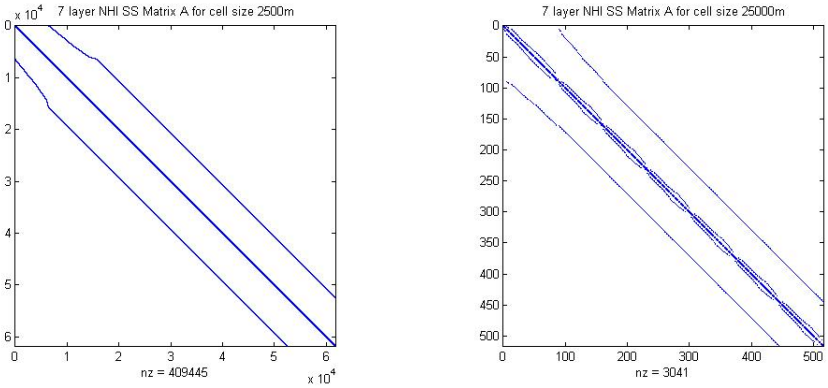
$$A\underline{u} = \underline{f} \tag{2.31}$$

Description of A, $\underline{u}$ and $\underline{f}$ is similar to the equation 2.31. Treatment of the solution of this system of equations shall be discussed in detail in chapter 3.

In the figure 2.7, we present the sparsity structure of the matrix A for NHI SS model. The description of NHI SS model has been given in the chapter 7. The number of cells in the first layer denoted by L_1, is less compared to other 6 layers below L_1. Therefore, we observe that, the band of the layer L_1 is smaller than the band of other layers (figure 2.7a). Also, we are not able to see the 7 bands clearly in 2.7a, so we present the sparsity of A from a coarser model. We can see 7 bands in figure 2.7b.

2.7. Type of Model Cells

In practice, formulating an equation of the form of equation 2.31 for every cell in a model grid is not required because the status of the cells is specified at the start of the simulation. The aquifer is of irregular shape, whereas the model grid is always rectangular. Therefore, we need to define some flag to distinguish the aquifer cells and to impose boundary conditions. We define the IBOUND variable for that



(a) cell size 2500 m (b) cell size 25000 m

Figure 2.7: Spy plot of coefficient matrix A from first Picard iteration in 7 layer NHI SS Model, grid cell: $120 \times 130 \times 7$ in 2.7a and $12 \times 13 \times 7$ in 2.7b.

purpose. Figure 2.8 taken from MODFLOW 2005 manual [25], shows a graphical representation of the aquifer, model domains and three type of cell mentioned below:

No-flow cells: No-flow cells are those for which no flow into Or out of the cell is permitted. These cells are used to simulate the boundary condition for example to signify the domain of the model. The value of the IBOUND variable is zero. No flow cell is also called as inactive cell. There is no contribution of the conductance from the no-flow cells to the nearby active cells, so the corresponding conductances (CC, CR, and CV) are set to zero for the no-flow cell. We do not solve for the head using LSE for no-flow cells. We also do not allocate the deflation vector entry in the no-flow cell.

Constant-head cells: Constant-head cells are those for which the head is specified for each time, and the head value does not change as a result of solving the flow equations 2.31. These cells are also used to simulate the boundary condition. These are used when there is a requirement from physical properties of an area, e.g. if we want to keep a constant head in a sea or river. The IBOUND variable has value less than zero in a cell with the constant head. Constant head serves as nonhomogeneous Dirichlet boundary condition (see the unit model problem in chapter 6). Hence, the matrix A and the RHS vector for the cells near the constant head boundary is updated using with the conductances of the constant-head cells. Since the head values are known at constant-head cells, We do not solve for the head using LSE. We also do not allocate the deflation vector entry in these cells. The deflation method has been described in the chapter 5.

Variable-head cells: The variable-head cells are characterized by heads that are

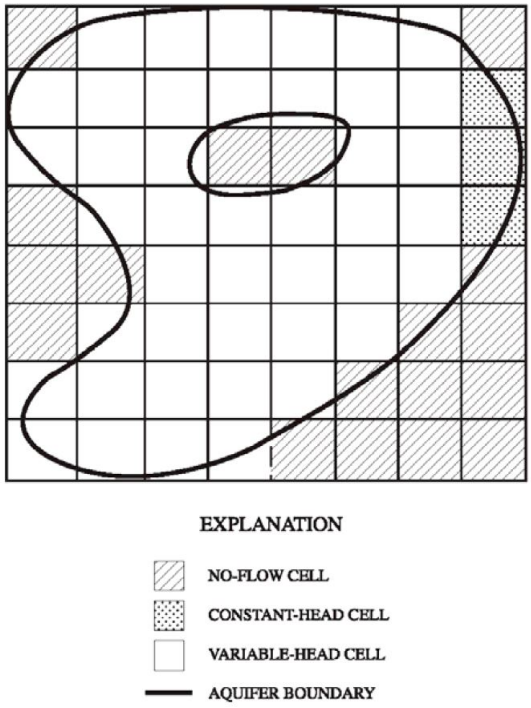


Figure 2.8: Discretized aquifer showing boundaries and cell designations

unspecified and free to vary with time. These are also called active cells. The LSE 2.31 is solved to compute the head for only variable-head cells. We allocate the entries in the deflation vector only for variable-head cells. The IBOUND variable has the value greater than zero in these cells.

2.8. Overview of MODFLOW Simulation

In figure 2.9, we present different steps used in the Ground-Water Flow Process. The total period of simulation is divided into a series of stress periods within which specified stress data values (such as pumping rate in the well) are constant. Each stress period, in turn, is divided into a series of time steps. The system of finite volume equations of the form of equation 2.28 is formulated and solved to yield the head at each node at the end of each time step. A preconditioned conjugate gradient (PCG) iterative solver is used to solve for the heads for each time step. Thus, the program includes three nested loops: a stress-period loop, within which there is a timestep loop, which in turn contains a solver iteration loop. The initial head values are chosen from the computed head at previous time step in case of transient state simulation.

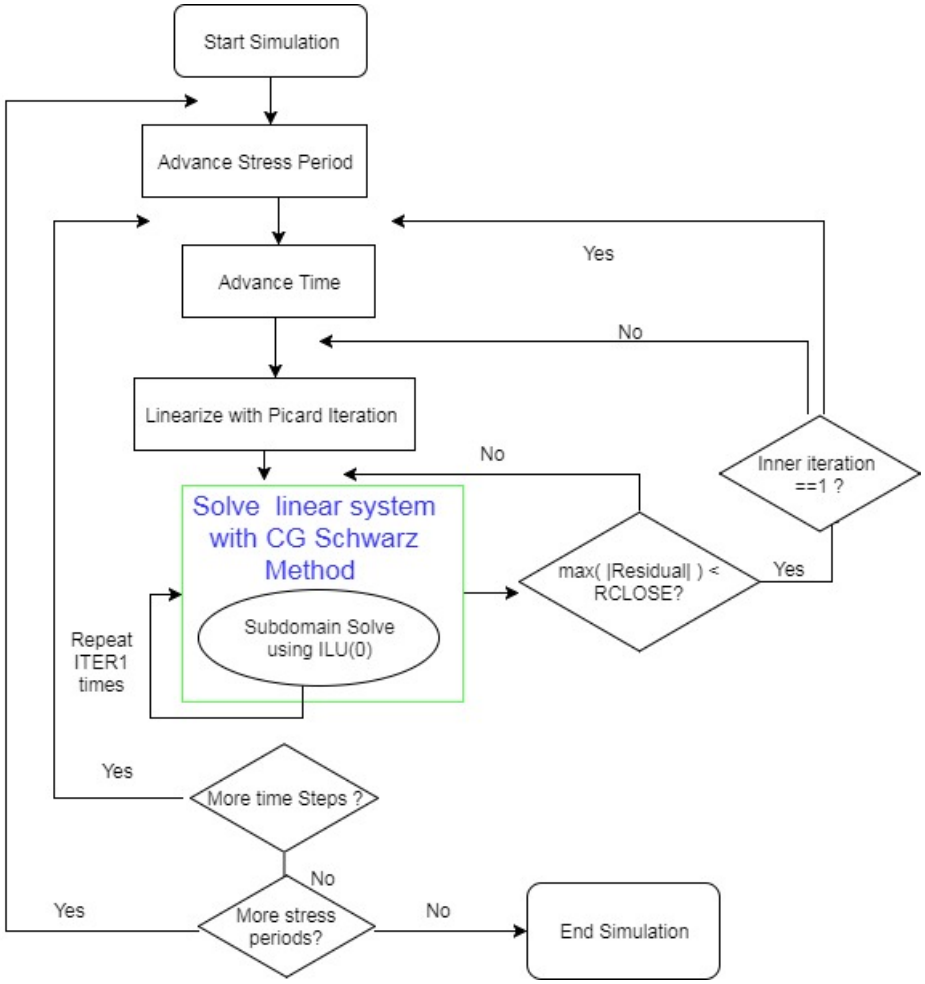


Figure 2.9: Flowchart of program to simulate Groundwater Flow in MODFLOW

Now, we discuss the solver loop in detail. The system of equations 2.28 may be nonlinear with respect to the head (refer section 2.5). Therefore, the matrix coefficients in A in the equation 2.30 may depend upon the current head values. The concept of Picard iteration is used to make the system of equations linear.

Solver iteration: Two type of loops (iterations) are performed in the solver: outer iteration and inner iteration. An outer iteration is a Picard iteration, and the inner iteration is conjugate gradient (CG) iteration. In each Picard iteration, the matrix A and the right-hand side vector f are set using the current head values. Thus LSE is constructed in each Picard iteration. The head values solved in previous Picard iteration are used as an initial head values to solve the LSE in the current Picard iteration. A maximum number of inner iterations, denoted by `ITER1` (by default 30) are set. In each Picard iteration, the solver is said to be converged when the absolute maximum of head values of the difference between the current and previous inner iteration becomes smaller than `HCLOSE`, and the maximum residual norm becomes smaller than `RCLOSE`. The `infinity` norm has been used to check the convergence for the residual norm. The inner iteration loop terminates either when convergence is met, or when a maximum number of inner iterations have occurred. Outer iteration loop terminates when `HCLOSE` and `RCLOSE` termination criteria is fulfilled in the first inner iteration.

In our research, we have considered that the stress period and time step are fixed since we are only interested in reducing the inner and outer iterations in the solver. Therefore, we focus only on the linear system (colored in green in figure 2.9. The linear system is being solved by a domain decomposition method, as described in chapter 4.

In this chapter, we have established the ingredients to define the linear system of equations. In the next chapter, we will focus on the iterative scheme to solve such systems. Since the system is symmetric positive definite (SPD), we will stick to the PCG method.

3

Iterative Methods

3.1. Introduction

The large nonlinear system of equations formulated in the preceding chapter are linearized with the Picard iteration, to form the linear system of equations (LSE). In this chapter, we present iterative methods to solve the LSE.

We start by giving the motivation to use the iterative methods to the reader in section 3.2. We define the basic and Krylov subspace iterative methods in section 3.3 and 3.4. At the end of the chapter, we define the preconditioning techniques to solve the LSE efficiently.

3.2. Why Iterative Methods?

In this section, we describe two approaches to solve the linear system of equations (LSE) given by 3.1.

$$A\underline{u} = \underline{f} \quad \text{where} \quad A \in \mathbb{R}^{n\times n},\ \underline{u}, \underline{f} \in \mathbb{R}^{n}. \tag{3.1}$$

In the above equation, A is assumed to be non-singular. We have underlined the vectors in equation 3.1, so that the reader can differentiate these vectors with the scalars. However, many such vectors appear in the upcoming part in this report, and sometimes the notations get complicated. Hence, we do not show a line under the vectors.

3.2.1. Direct Methods

Direct methods attempt to solve the LSE by a finite sequence of operations. In this approach, the coefficient matrix A is decomposed into 2 matrices, which are easier to solve. For example, in a so-called LU decompositions, A is decomposed into a lower triangular matrix L, and an upper triangular matrix U such that $A = LU$. Equation 3.1 becomes

$$LUu = f \Rightarrow \quad Ly = f \quad \text{where} \quad Uu = y \tag{3.2}$$

u can be obtained by first solving for y using forward substitution and then for u using backward substitution in the equation 3.2. To summarize, the LSE is solved in two stages and we mention the time and space complexity for the dense matrices below:

- Decomposition of A into LU - $O(n^3)$ computational complexity and storage space of $O(n^2)$ for L and U.
- Solution of equation 3.2 - $O(n^2)$ computational complexity and storage space of $O(n)$ for y.

From above we notice that the most expensive step is the decomposition of A into L and U. For dense matrices, LU decomposition requires $O(n^3)$ operations. For sparse matrices with band k, the LU decomposition complexity reduces to $O(nk^2)$ where $k = n^{\frac{1}{2}}$ when A is a 2D finite difference matrix and $k = n^{\frac{2}{3}}$ when A is a 3D finite difference matrix. Furthermore, for large matrices, it is unfeasible to store L and U in the computer memory. Due to large time complexity, the CPU time becomes quite large. Therefore, we need iterative methods to solve the LSE.

3.2.2. Iterative Methods

An iterative method is a mathematical procedure that generates a sequence of approximate solutions, in which the n-th approximation is derived from the previous ones. A termination criterion and initial approximation are specified for an iterative algorithm. An iterative method is called convergent if the corresponding sequence converges to the solution for given initial approximation.

We denote the converging sequence of iterates by

$$\{u^{(k)}\}_{k\geq 0}, \text{ where } \quad u^{(k)} \to u^* \text{ for } \quad k \to \infty \tag{3.3}$$

where $u^{(0)}$ and u^* denotes the initial guess and the exact solution respectively. At the k-th iteration, we define error vector $e^{(k)}$ and residual vector $r^{(k)}$ by

$$e^{(k)} = u^* - u^{(k)}, \quad r^{(k)} = f - Au^{(k)} \tag{3.4}$$

From equation 3.1 and 3.4, we form the residual equation

$$Ae^{(k)} = Au^* - Au^{(k)} = f - Au^{(k)} = r^{(k)} \tag{3.5}$$

Now we split A into a non-singular matrix M(we assume it exists) and N by $A = M - N$ and thus equation 3.1 becomes

$$Mu = Nu + f \tag{3.6}$$

Multiplying M^{-1} to the left of equation 3.6, we define the iterative scheme:

$$\begin{aligned} u^{(k+1)} &= M^{-1}Nu^{(k)} + M^{-1}f \\ &= M^{-1}(M-A)u^{(k)} + M^{-1}f \\ &= u^{(k)} + M^{-1}(f - Au^{(k)}) \\ &= u^{(k)} + M^{-1}r^{(k)} \end{aligned} \tag{3.7}$$

Various iterative schemes can be developed by choosing different values of the non singular matrix M in equation 3.7. We treat a couple of these schemes in next section.

3.3. Basic Iterative Methods

Matrix A can be decomposed into a lower triangular matrix L, and upper triangular matrix U and diagonal matrix D such that $A = L + D + U$. In this section, we define two basic iterative methods (BIM).

3.3.1. Jacobi Method

We substitute $M_{JAC} = D$ in equation 3.7 to get

$$\begin{aligned} u^{(k+1)} &= u^{(k)} + D^{-1}(f - Au^{(k)}) \\ &= u^{(k)} + D^{-1}(f - (D+L+U)u^{(k)}) \\ &= D^{-1}f + u^{(k)} - u^{(k)} - D^{-1}Lu^{(k)} - D^{-1}Uu^{(k)} \\ &= D^{-1}(f - Lu^{(k)} - Uu^{(k)}) \end{aligned} \tag{3.8}$$

We can define the update scheme from equation 3.8

$$u_i^{(k+1)} = [f_i - \sum_{j=1, j\neq i}^{n} a_{ij}u_j^{(k)}]/a_{ii} \quad \forall i = 1, \dots, n \tag{3.9}$$

3.3.2. Gauss-Seidel Method

We substitute $M_{GS} = D + L$ in equation 3.7 to get

$$\begin{aligned} u^{(k+1)} &= u^{(k)} + (L+D)^{-1}(f - Au^{(k)}) \\ (L+D)(u^{(k+1)} - u^{(k)}) &= (f - (D+L+U)u^{(k)}) \\ Du^{(k+1)} &= f - Lu^{(k+1)} - Uu^{(k)} \\ u^{(k+1)} &= D^{-1}(f - Lu^{(k+1)} - Uu^{(k)}) \end{aligned} \tag{3.10}$$

Now, we can define the update scheme from equation 3.10

$$u_i^{(k+1)} = [f_i - \sum_{j=1}^{i-1} a_{ij}u_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}u_j^{(k)}]/a_{ii} \quad \forall i = 1, \dots, n \tag{3.11}$$

From above, we observe that

- The Jacobi iteration allows to update all components of the iterates $u^{(k)}$ independently from each other and is therefore suitable for parallel computing. For example p parallel threads can independently work on subdomain to update the values from $u^{(k)}$ to $u^{(k+1)}$.
- The Gauss-Seidel iteration uses recent information as soon as it becomes available. So, Gauss-Seidel iteration converges faster the Jacobi since the spectral radius $\rho_{GS} < \rho_{JAC}$. However, the plain Gauss-Seidel is not suitable for parallelism.

Both algorithms converge slowly but provide a basic iterative scheme.

3.4. Krylov Subspace Methods

Modern iterative methods for solving large systems of linear equations avoid matrix-matrix operations, but rather multiply vectors by the matrix and work with the resulting vectors. Starting with a vector, f, one computes Af; then one multiplies that vector by A to find A^2f and so on. All algorithms that work this way are referred to as `Krylov subspace` methods; they are among the most successful methods currently available in numerical linear algebra. The name Krylov has been adapted after Russian applied mathematician and naval engineer Alexei Krylov, who published a paper about these methods in 1931.

We show the formation of the Krylov subspace for Richardson iteration. The Richardson update is given by

$$u^{(k+1)} = u^{(k)} + r^{(k)} \tag{3.12}$$

where, $u^{(k)}$ is the approximation of solution and $r^{(k)}$ is residual at k^{th} step. The iterates are given by

$$\begin{aligned} u^{(1)} &= u^{(0)} + r^{(0)} \\ u^{(2)} &= u^{(1)} + r^{(1)} \\ &= u^{(0)} + r^{(0)} + r^{(1)} \\ &= u^{(0)} + r^{(0)} + (f - A(x^{(0)} + r^{(0)})) \\ &= u^{(0)} + 2r^{(0)} - Ar^{(0)} \\ &= u^{(0)} + 2A^0r^{(0)} - A^1r^{(0)} \end{aligned} \tag{3.13}$$

From 3.13, we see that $(u^{(2)} - u^{(0)})$ can be represented as linear combination of A^0r and A^1r. In a similar way, $(u^{(k)} - u^{(0)})$ can be represented as linear combination of $A^0r, A^1r, \cdots, A^{k-1}r$. We define Krylov Subspace $\mathcal{K}_k(A; r^{(0)})$ by

$$\mathcal{K}_k(A; r^{(0)}) = \text{span}\{r^{(0)}, Ar^{(0)}, A^2r^{(0)}, \ldots, A^{k-1}r^{(0)}\}. \tag{3.14}$$

A BIM does not use the optimal approximation from $\mathcal{K}_k(A; r^{(0)})$ to get the solution and hence converges slowly. Krylov subspace methods are used for the faster

convergence because they are based on optimal properties. The best known Krylov subspace methods to solve LSE are Conjugate Gradient (CG), GMRES (generalized minimum residual), BiCGSTAB (bi-conjugate gradient stabilized), IDR(s) (Induced dimension reduction). The matrix in LSE that we get in MODFLOW is symmetric positive definite (SPD) (refer to Appendix). CG method is most efficient Krylov subspace method to solve SPD matrices. Therefore, we define it in the next section.

3.4.1. Conjugate Gradient (CG) Method

The CG methods requires the matrix A from equation 3.1 to be

Symmetric The matrix A should be such that, $A = A^T$. For symmetric matrix A, all eigenvalues of A are real.

Positive definite The matrix A should be such that, $y^T Ay > 0 \ \forall y \neq 0$. It implies that all eigenvalues of A are positive.

If A is symmetric and positive definite, it is called symmetric positive definite (SPD) matrix. The matrix in LSE to be solved by the CG method should be a SPD matrix.

The following properties of the CG method make it one of the best iterative method to solve SPD LSE:

- Small recurrences: $u^{(k)}, r^{(k)}, p^{(k)}$ is simple to implement.
- Optimization property: $\|u^{(k)} - u\|_A$ is minimal, where $\|x\|_A$ is called A-norm of a vector x, defined by $x^T Ax$.
- It is a Krylov Subspace method.

Convergence of CG method depends upon the condition number κ of the coefficient matrix A. The error at time step k is bounded by the equation 3.15. The proof can be found in [34].

$$\|u - u^{(k)}\|_A \leq 2\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k \|u - u^{(0)}\|_A \tag{3.15}$$

The condition number of A (κ) is huge if the eigenvalues are not well clustered. We observe from 3.15, the bound on the error is large. It slows the convergence of the CG method. In next section, we describe preconditioning technique to speed up the convergence. The Preconditioned Conjugate Gradient (PCG) algorithm is given in the algorithm 1.

3.5. Preconditioning

Convergence of the iterative methods depends on the condition number(κ) of the matrix. The condition number of a SPD matrix is defined as ratio of largest eigenvalue (λ_{max}) and smallest eigenvalue (λ_{min}).

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}}.$$

When eigenvalues of A are not well clustered, we say that the matrix is not well conditioned and we should incorporate some preconditioner to improve the spectrum of A. Usually, we apply the preconditioner to the left of A, and thus known as left preconditioner denoted by M. The transformed system is given by

$$M^{-1}Au = M^{-1}f \tag{3.16}$$

Algorithm 1 Preconditioned Conjugate Gradient (PCG) Algorithm

procedure PCG($A, f, u^{(0)}, tol, k_{max}, M$)
  $r^{(0)} = f - Au^{(0)}, k = 1$ ▷ Initialization
  while ($k < k_{max}$ **and** $\|r^{(k-1)}\| > tol$) **do**
    $z^{(k-1)} = M^{-1}r^{(k-1)}$ ▷ Preconditioning
    if $k = 1$ **then**
      $p^{(1)} = z^{(0)}$
    else
      $\beta_k = \frac{(r^{(k-1)})^T z^{(k-1)}}{(r^{(k-2)})^T z^{(k-2)}}$
    $p^{(k)} = z^{(k-1)} + \beta_k p^{(k-1)}$ ▷ Search direction
    end if
    $\alpha_k = \frac{(r^{(k-1)})^T z^{(k-1)}}{(p^{(k)})^T A p^{(k)}}$
    $u^{(k)} = u^{(k-1)} + \alpha_k p_k$ ▷ Iterate update
    $r^{(k)} = r^{(k-1)} - \alpha_k A p_k$ ▷ Residual update
    $k = k + 1$
  end while
  k=k-1
  return $u^{(k)}$ ▷ The converged solution
end procedure

After applying the preconditioner, instead of solving equation 3.1, the equation 3.16 is solved because the solution converges faster due to clustered spectrum of $M^{-1}A$. M should also be SPD to solve the system 3.16 with the CG method. The following requirements should be fulfilled for the preconditioner:

- The eigenvalues of $M^{-1}A$ should be clustered around 1.
- $M^{-1}z$ should be cheap to compute for vector z.

The two obvious choices for M are Identity matrix (I) and coefficient matrix (A), but both have the following pitfalls:

- Identity matrix (I): It is the same LSE. The spectrum has not improved.
- Coefficient matrix (A): $M^{-1}z$ is NOT cheap to compute.

The above are two extreme cases of M. The preconditioner M should lie somewhere in between A and I. We define Jacobi and ILU preconditioner in the next section.

3.5.1. Jacobi Preconditioner

The Jacobi Preconditioner is perhaps the simplest preconditioner also known as diagonal scaling. It is defined by

$$m_{i,j} = \begin{cases} a_{i,i} & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Bounds on the eigenvalues can be given by Gershgorin circle theorem[9] in which, the eigenvalues are contained in the circle with center $a_{i,i}$ and radius $\sum_{j=1, i \neq j}^{N} a_{i,j}$. Applying the preconditioner to the left, the diagonal entries in $M^{-1}A$ become 1, while the sub-diagonal entries in $M^{-1}A$ become smaller than sub-diagonal entries of A. For the transformed system $M^{-1}A$, the center of the Gershgorin circle becomes 1, and the radius becomes smaller than the radius of Gershgorin circle of A. In this way, we see that the eigenvalue spectrum becomes more clustered (around 1) in $M^{-1}A$ compared to A. Division operation is costlier than the multiplicative operator. Hence, we store $\frac{1}{a_{i,i}}$ in M^{-1} and $M^{-1}z$ is cheaper to compute. The i^{th} entry in $M^{-1}z$ simply becomes $\frac{z_i}{a_{i,i}}$.

We also present another version of Jacobi preconditioner, known as Block Jacobi preconditioner. It is also known as Additive Schwarz preconditioner that we present in the next chapter. Let the index set $S = \{1, 2, .., n\}$ be partitioned such that $S = \cup S_i$ with the sets S_i mutually disjoint. The block Jacobi preconditioner is defined by

$$m_{i,j} = \begin{cases} a_{i,j} & \text{if } i \text{ and } j \text{ are in the same index subset,} \\ 0 & \text{otherwise.} \end{cases}$$

The block Jacobi preconditioner is suitable for parallel programming. Each set S_i can be seen as i-th subdomain grid points. It is natural to let the partitioning coincide with the division of variables over the multiple processors. For the case of 2 processors, The coefficient matrix A and block Jacobi preconditioner M are given as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad M = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix}$$

3.5.2. ILU Preconditioner

The sparse linear system from 3.1 can be solved by computing the LU factorization with L unit lower triangular and U upper triangular. One then solves $Ly = b, Ux = y$ with forward and backward substitution. For a typical sparse matrix, the matrices L and U can be much less sparse than the original matrix (refer to figure 3.1b and 3.1c). Incomplete factorization of A is performed to maintain the sparsity structure similar to A in L and U.

An incomplete factorization seeks triangular matrices $\tilde{L} \approx L$ and $\tilde{U} \approx U$, such that $A \approx \tilde{L}\tilde{U}$ rather than A=LU. The sparsity pattern in $\tilde{L}$ and $\tilde{U}$ is often chosen to

be similar to the original matrix A. (refer to figure 3.1d and 3.1e). Solving for x from $\tilde{L}\tilde{U}x = b$ is cheaper than from $LUx = b$, but does not yield the exact solution to $Ax = b$. The matrix $M = \tilde{L}\tilde{U}$ as a preconditioner. This preconditioner is called ILU(0). In ILU(0), zero level of fill-in is used.

There are other variants of ILU with more levels of fill-in. They are known as ILU(p) with p >0 level of fill in. Increasing the value of p, it converges faster than ILU(0) in but the computational work per iteration becomes more than that of ILU(0), so there is a trade-off between the accuracy and the computational cost. The optimal wall-clock is expected from the intermediate level of fill-in, but we found the optimal wall-clock for ILU(0). To see the effect of fill-in on the number of CG iterations and wall clock time, we experimented in MATLAB (refer to Appendix). We found out that the number of iterations decreases with more levels of fill-in but the time taken to set up the preconditioner increases when we increase the fill in p since L and U become denser. We conclude that concerning wall clock time; the ILU(0) preconditioner should be preferred over ILU(p) with $p > 0$ for these problems.

Since forward and backward substitution can not be performed in parallel fashion, ILU preconditioners are not suitable for parallel computations. ILU(0) is used to solve the subdomain problem in-exactly in MODFLOW. To exploit parallelism in ILU(0), Saad[30, chap 12] has suggested Red Black ordering of grid points. In this approach, we define the red and black label for alternative indexes in 2D such that update to red points does not depend on black points and vice versa. The black points can be updated in parallel, and red points can also be updated in parallel.

3.6. Connection with the PKS Package

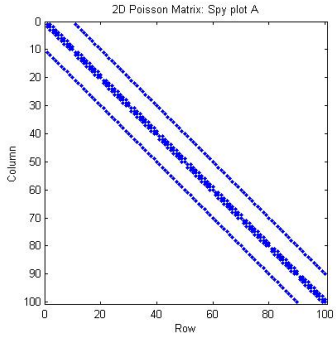
In the PKS package, serial and parallel implementation of PCG is given as follows:

3.6.1. Serial Implementation

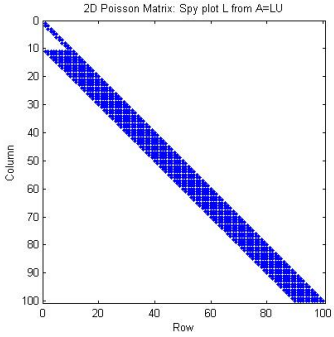
- PKS includes PCG and BiCGSTAB respectively to solve symmetric and asymmetric linear system.
- The matrix A is sparse. However, the full LU decomposition of A removes the sparsity in L and U. Hence, in the preconditioning step, the linear system is solved inaccurately using incomplete LU decomposition with zero fill in, i.e. ILU(0) preconditioner.
- PKS has been developed for unstructured grids.

3.6.2. Parallel Implementation

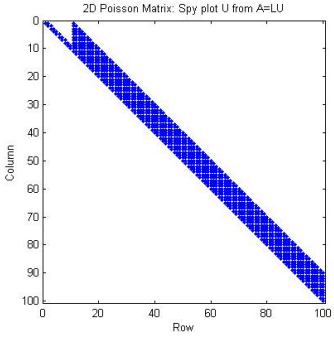
- Due to inherent parallelism, the block Jacobi preconditioner has been implemented in the PCG solver.



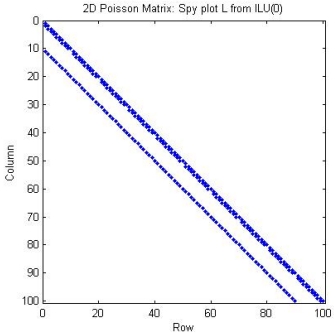
(a) A



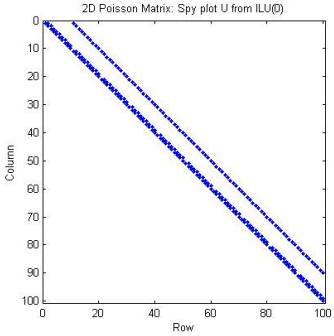
(b) L from A=LU



(c) U from A=LU



(d) L from ILU(0)



(e) U from ILU(0)

Figure 3.1: Spy plot of various matrices arising from 2D Poisson matrix, with no fill-in and full fill-in.

4

Domain Decomposition Methods

4.1. Introduction

In Numerical Analysis, Domain Decomposition(DD) methods attempt to solve a Boundary Value Problem by splitting it, and solving sub problems on various subdomains. In Parallel Programming context, DD denotes decomposing the physical domain of a particular computational task across different processors.

DD is an active research field [6] since long run time of simulations can be reduced with DD methods using state of the art parallel processors. The study of domain decomposition methods can be motivated by the following factors [17]:

- Each subproblem can be independently solved in parallel on different processors.
- Reduction of memory requirements per subproblem, since each subproblem is smaller than the given problem.
- Complicated geometry can be solved by solving simple geometry problem on different subdomains.

Surveys of different DD methods can be found in Saad [30, chap 14] and Dolean [6, chap 1,2]. We will restrict ourselves to Schwarz DD methods.

The structure of this chapter is as follows. We describe Additive and Multiplicative Schwarz methods in section 4.2 and 4.3. In section 4.4, we define two techniques to speed up the PCG solver in PKS. In section 4.5, we give connection of this chapter to the PKS package. We end this chapter by specifying the global and local grid numbering in the PKS in section 4.6.

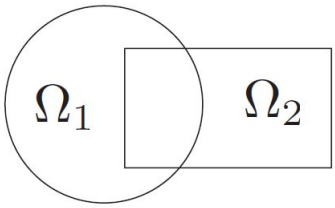


Figure 4.1: Complex geometry made from two simple domains showing the domain decomposition

Schwarz Alternating Methods: Schwarz Methods dates from 1870, developed by a German mathematician Hermann Schwarz, are overlapping domain decomposition methods. They solve a PDE by alternating to various sub-domain: solving the PDE on one domain at each iteration, and taking the updated value from the sub-domain as a boundary condition for the neighboring sub-domain (see Saad [30, 14.3]). For simplicity we restrict to a one dimension domain (1D) case with 2 subdomains. It can be generalized to one dimension domain (2D) and more subdomains with ease.

4.2. Multiplicative Schwarz Method

The Multiplicative Schwarz Method is most natural form of Schwarz methods, which attempts to solve the following differential equation:

$$\begin{aligned} \mathcal{L}u &= f \quad \text{in } \Omega = [\alpha, \beta] \\ u(\alpha) &= p, \quad u(\beta) = q. \end{aligned} \tag{4.1}$$

The domain Ω is decomposed into subdomains, $\Omega = \Omega_1 \cup \Omega_2$, such that $\Omega_1 \cap \Omega_2 \neq \emptyset$. Let the solution in Ω_1 and Ω_2 be given by u_1 and u_2 respectively. We assume that initial guess $u_2^{(0)}$ is known. Please refer to figure 4.2 for notations.

We solve subproblem $u_1(k)$ iteratively in subdomain Ω_1, for $k = 1, 2, \ldots$ until convergence.

$$\begin{aligned} \mathcal{L}u_1^{(k)} &= f \quad \text{on } \Omega_1, \\ u_1^{(k)}(\alpha) &= p, \\ u_1^{(k)} &= u_2^{(k-1)} \quad \text{on } \Gamma_1. \end{aligned} \tag{4.2}$$

followed by the subproblem $u_2(k)$ in subdomain Ω_2,

$$\begin{aligned} \mathcal{L}u_2^{(k)} &= f \quad \text{on } \Omega_2, \\ u_2^{(k)} &= u_1^{(k)} \quad \text{on } \Gamma_2, \\ u_2^{(k)}(\beta) &= q. \end{aligned} \tag{4.3}$$

For the overlapping subdomain, u_1 or u_2 can be chosen to be the solution u, so here we take u_1 as a solution and we define the $k-$th iteration by

$$u^{(k)}(x) = \begin{cases} u_1^{(k)}(x), & \text{if } x \in \Omega \setminus \Omega_2 \\ u_2^{(k)}(x), & \text{if } x \in \Omega_2. \end{cases} \tag{4.4}$$

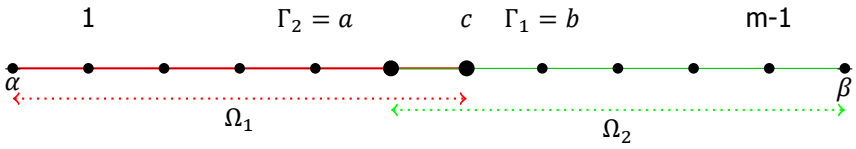


Figure 4.2: 1D domain and discretization of for Schwarz method

Discretization of equation 4.1 leads to the linear system

$$A\underline{u} = \underline{f} \tag{4.5}$$

Let n be total number of grid-points in Ω. Let n_1 and n_2 be the number of grid-points in the subdomain Ω_1 and Ω_2 respectively such that $n_1 + n_2 > n$ (overlapping subdomains). We denote the index sets of interior points in the subdomain Ω_1 and Ω_2 by I_1 and I_2. Let $A \in \mathbb{R}^{n\times n}$ be the coefficient matrix given by

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

where for $i = 1, 2$, $A_{ii} \in \mathbb{R}^{n_i \times n_i}$ denotes (local) coupling in the subdomain while $A_{i(3-i)} \in \mathbb{R}^{n_i \times n_{(3-i)}}$ denotes (global) coupling across the subdomains. Overlapping in the diagonal block matrices are shown using the box.

To define the local coupling sub matrices we need to define the restriction operators $R_i : \mathbb{R}^n \to \mathbb{R}^{n_i}$ such that for $i = 1, 2$

$$\{R_i v\}_j = \begin{cases} v_j, & \text{if } \quad j \in I_i \\ 0, & \text{otherwise.} \end{cases}$$

From the above definitions of R_i and A we can write,

$$A_{11} = R_1 A R_1^T, \qquad A_{22} = R_2 A R_2^T.$$

Algorithm 2 Multiplicative Schwarz Procedure

1: **procedure** MSM($u^{(0)}$)
2: **for** $k = 1, 2, \ldots$ until convergence **do**
3: $u^{(k-\frac{1}{2})} = u^{(k-1)} + R_1^T A_{11}^{-1} R_1 (f - A u^{(k-1)})$ ▷ Subdomain 1 correction
4: $u^{(k)} = u^{(k-\frac{1}{2})} + R_2^T A_{22}^{-1} R_2 (f - A u^{(k-\frac{1}{2})})$ ▷ Subdomain 2 correction
5: **end for**
6: **return** $u^{(k)}$ ▷ Converged solution
7: **end procedure**

4.3. Additive Schwarz Method

Additive Schwarz Method (ASM) solves the total problem approximately by splitting it into smaller domains, solving subproblems on these subdomains and adding the results. Since ASM is suitable in parallel computers, it has gained popularity recently.

Algorithm 3 Additive Schwarz Procedure

1: **procedure** ASM($u^{(0)}$)
2: **for** $k = 1, 2, \ldots$ until convergence **do**
3: $u_{c1} = R_1^T A_{11}^{-1} R_1 (f - A u^{(k-1)})$ ▷ Subdomain 1 correction
4: $u_{c2} = R_2^T A_{22}^{-1} R_2 (f - A u^{(k-1)})$ ▷ Subdomain 2 correction
5: $u^{(k)} = u^{(k-1)} + u_{c1} + u_{c2}$ ▷ Add the subdomain corrections
6: **end for**
7: **return** $u^{(k)}$ ▷ Converged solution
8: **end procedure**

The above algorithm can be re-written as

$$u^{(k)} = u^{(k-1)} + M^{-1}(f - A u^{(k-1)}) \tag{4.6}$$

For 2 subdomains, $M_{AS}^{-1} = R_1^T A_{11}^{-1} R_1 + R_2^T A_{22}^{-1} R_2$.

ASM preconditioner is same as block Jacobi preconditioner. If $\Omega 1 \cap \Omega 2 = \emptyset$, the block-Jacobi preconditioner can be defined as:

$$M = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix}$$

In algorithm 3, corrections in both subdomains are independent, so this method suits well for parallelism. For example processor P_1 updating I_1 indexes, and processor P_2 updating I_2 indexes can work in parallel to produce the subdomain correction terms. In the non-overlapping case, the addition of subdomain corrections to $u^{(k-1)}$ requires adding different subdomain components of the vector to obtain $u^{(k)}$. In the presence of an overlap, the corrections obtained from different subdomains (processors) are summed up to form the vector $u^{(k)}$.

Although the Additive Schwarz(AS) method is suitable for parallelism, it converges slowly. Various techniques are employed to make the convergence faster such as, introducing (global coupling) and introducing Krylov acceleration (CG-Schwarz method in the PKS package).

In the Krylov acceleration approach, the Krylov subspace methods are used to solve the LSE and the AS method is used as a preconditioner. To generalize, we increase the number of subdomains from 2 to s. The local subdomain A_{ii} matrices and restriction matrices are defined similar to 2 subdomain case. In the PCG method, $z^{(k-1)} = M^{-1}r^{(k-1)}$ (line 4 in the PCG algorithm 1 in chapter 3) can be calculated in parallel using the algorithm 4 by all s subdomains. All the s subdomains have local vector $z^{(k-1)}$ and the remaining steps (lines) of the PCG algorithm are executed.

Algorithm 4 Additive Schwarz as a preconditioner

1: **procedure** ASP(r)
2: **for** $i = 1, 2, ..., s$ **do** ▷ Subdomain i correction
3: $c_i = (R_i^T A_{ii}^{-1} R_i)r$
4: **end for**
5: $c = c_1 + c_2 + ... + c_s$
6: **return** $M^{-1}r$ ▷ Preconditioning step in PCG
7: **end procedure**

Rewriting equation 4.6 we get

$$u^{(k)} = u^{(k-1)} + (R_1^T A_{11}^{-1} R_1 + R_2^T A_{22}^{-1} R_2)(f - Au^{(k-1)}) \tag{4.7}$$

4.3.1. Overlapping AS Method

To increase the convergence of ASM, we introduce global coupling. The global coupling can be increased by introducing the overlap between neighboring subdomains. This introduce extra non-zero entries in subdomain matrices A_{11} and A_{12}. Overlapping introduces extra overhead because of duplicate computations, hence minimal overlapping is recommended for Schwarz DD methods. In the context of parallel programming, overlapping increases communication between the processors. Equation 4.7 can be simplified using the notation from figure 4.2 to obtain 4.8.

$$u^{(k)} = \begin{bmatrix} u_1^{(k-1)} \\ \vdots \\ u_a^{(k-1)} \\ u_{a+1}^{(k-1)} \\ \vdots \\ u_{b-1}^{(k-1)} \\ u_b^{(k-1)} \\ \vdots \\ u_{m-1}^{(k-1)} \end{bmatrix} + \begin{bmatrix} A_{11}^{-1} \begin{bmatrix} f_1 \\ \vdots \\ f_a \\ f_{a+1} \\ \vdots \\ f_{b-1} - \frac{u_b^{(k-1)}}{h^2} \end{bmatrix} - \begin{bmatrix} u_1^{(k-1)} \\ \vdots \\ u_a^{(k-1)} \\ u_{a+1}^{(k-1)} \\ \vdots \\ u_{b-1}^{(k-1)} \end{bmatrix} \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ A_{22}^{-1} \begin{bmatrix} f_{a+1} - \frac{u_a^{(k-1)}}{h^2} \\ \vdots \\ f_{b-1} \\ f_b \\ \vdots \\ f_{m-1} \end{bmatrix} - \begin{bmatrix} u_{a+1}^{(k-1)} \\ \vdots \\ u_{b-1}^{(k-1)} \\ u_b^{(k-1)} \\ \vdots \\ u_{m-1}^{(k-1)} \end{bmatrix} \end{bmatrix} \tag{4.8}$$

Limitations of overlapping AS Method

- The above iterative method can be seen as a fixed point iteration, and it turns out, that spectral radius(ρ) [1] of iteration matrix is 1. This eigenvalue corresponds to the overlap region (a, b), so AS is not convergent in the overlap (see Efstathiou [22]).
- In parallel programming context, when we assume that the update of each subdomain is assigned to a process, only partial sums are computed by each process in the overlap, so each process has to communicate its partial sums to produce the global sum. This adds substantial overhead.

We have investigated the ways to overcome the above limitations and found out that restrictive overlapping can be used (see Cai [19]).

4.3.2. Restrictive Overlapping in AS Method

Let n be total number of interior grid-points in Ω. Let $\tilde{n_1}$ and $\tilde{n_2}$ be the restricted number of grid-points in the restricted subdomain $\tilde{\Omega_1} : \{1, 2, ..., c-1\}$ and $\tilde{\Omega_2} : \{c, ..., m-1\}$ respectively such that $\tilde{n_1} + \tilde{n_2} = n$ and $\tilde{\Omega_1} \cap \tilde{\Omega_2} = \emptyset$. We denote the index sets of interior points in the restricted subdomains $\tilde{\Omega_1}$ and $\tilde{\Omega_2}$ by $\tilde{I_1}$ and $\tilde{I_2}$. Let $A \in \mathbb{R}^{n \times n}$ be the coefficient matrix given by

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

To define the local coupling sub matrices we need to define the restriction operators $\tilde{R_i} : \mathbb{R}^n \to \mathbb{R}^{\tilde{n_i}}$ such that for $i = 1, 2$

$$\{\tilde{R_i} v\}_j = \begin{cases} v_j, & \text{if} \quad j \in \tilde{I_i} \\ 0, & \text{otherwise.} \end{cases}$$

[1] Supremum among the absolute values of the elements in its spectrum.

Restrictive Additive Schwarz (RAS) method as a preconditioner is defined in the equation 4.9.

$$M_{RAS}^{-1} = \tilde{R}_1^T A_{11}^{-1} R_1 + \tilde{R}_2^T A_{22}^{-1} R_2. \tag{4.9}$$

From equation 4.6 and 4.9 we obtain

$$u^{(k)} = u^{(k-1)} + (\tilde{R}_1^T A_{11}^{-1} R_1 + \tilde{R}_2^T A_{22}^{-1} R_2)(f - Au^{(k-1)}) \tag{4.10}$$

We define Prolongation matrices $P_1 \in \mathbb{R}^{(m-1)\times(c-1)}$ and $P_2 \in \mathbb{R}^{(m-1)\times(m-c)}$ where $I_1 \in \mathbb{R}^{(m-1)\times(m-1)}$ and $I_2 \in \mathbb{R}^{(m-c)\times(m-c)}$

$$P_1 = \begin{bmatrix} I_1 \\ 0 \end{bmatrix}, \qquad P_2 = \begin{bmatrix} 0 \\ I_2 \end{bmatrix} \tag{4.11}$$

Now it can be simplified using the notation from figure 4.2 as,

$$u^{(k)} = \begin{bmatrix} u_1^{(k-1)} \\ \vdots \\ \vdots \\ u_{c-1}^{(k-1)} \\ u_c^{(k-1)} \\ \vdots \\ \vdots \\ u_{b-1}^{(k-1)} \\ \vdots \\ \vdots \\ \vdots \\ u_{m-2}^{(k-1)} \\ u_{m-1}^{(k-1)} \end{bmatrix} + P_1 \tilde{R}_1^T \left[A_{11}^{-1} \left[\begin{matrix} f_1 \\ \vdots \\ f_c \\ \vdots \\ f_{b-2} \\ f_{b-1} - \frac{u_b^{(k-1)}}{h^2} \end{matrix} \right] - \left[\begin{matrix} u_1^{(k-1)} \\ \vdots \\ u_c^{(k-1)} \\ \vdots \\ u_{b-2}^{(k-1)} \\ u_{b-1}^{(k-1)} \end{matrix} \right] \right] + P_2 \tilde{R}_2^T \left[A_{22}^{-1} \left[\begin{matrix} f_{a+1} - \frac{u_a^{(k-1)}}{h^2} \\ f_{a+2} \\ \vdots \\ f_c \\ \vdots \\ f_{m-1} \end{matrix} \right] - \left[\begin{matrix} u_{a+1}^{(k-1)} \\ u_{a+2}^{(k-1)} \\ \vdots \\ u_c^{(k-1)} \\ \vdots \\ u_{m-1}^{(k-1)} \end{matrix} \right] \right] \tag{4.12}$$

Remarks about RAS Method

- In Parallel Programming context, assume one process updates grid points of one subdomain. RAS does not involve partial sum calculations. Since process 1 updates the grid points in restricted subdomain 1 and process 2 updates the grid points in restricted subdomain 2. So process 1 does not have to communicate with process 2. Thus we can save communication among the processes.
- The spectral radius in the overlap region is less than 1 (see Efstathiou [22]). So RAS converges in the overlap also.
- It has one downside that the matrix M is not symmetric, so if we apply RAS as a preconditioner the matrix $M^{-1}A$ is no more symmetric. So we cannot apply RAS preconditioner for symmetric Krylov solvers.

4.4. Techniques to speed up the Solver

We suggest different ways to speedup the Krylov Solver. When looking at that problem from Parallel Computing point of view, we note that when we increase the number of subdomains(processors), the solver iteration does not remain fixed [26] . It hampers the scalability of the solver. To remedy this various methods are presented and compared (see Nabben [27]). We present the ideas behind the Deflation and coarse grid correction here. However, deflation method has been found superior (in iterations) than coarse grid correction in the paper by Nabben and Vuik. Therefore, we implement only Deflation in the PKS package.

4.4.1. Deflation

We have implemented Deflation method in PKS. Deflation is described in detail in the chapter 5.

4.4.2. Coarse Grid Correction

As we increases the number of subdomains, it takes more steps to propagate the information across the subdomain interfaces, so it takes more iterations to converge. Also, the small eigenvalues of A represent some global information which has to handled efficiently [6]. A classical remedy is to introduce coarse grid correction (CGC) that couples all the subdomains at each iteration of the iterative method. In this approach, the correction obtained from coarse grid is added to the preconditioner (M^{-1}). CGC preconditioner is also known as two level Additive Schwarz preconditioner. For 2 subdomains it is defined in equation 4.13. Symbols and operators have been defined in chapter 5 and this chapter are consistent for e.g. $E = Z^T AZ$.

$$M_{CGC}^{-1} = ZE^{-1}Z^T + R_1^T A_{11}^{-1} R_1 + R_2^T A_{22}^{-1} R_2. \tag{4.13}$$

Remarks about CGC Preconditioner

- The structure of two level AS preconditioner is same as that of one level AS preconditioner. So it's implementation is straight forward, given the implementation of one level AS preconditioner.
- In the two level Schwarz method only local subproblems are solved in parallel and $ZE^{-1}Z^T$ is global in nature.
- Extra overhead of inverting $E \in \mathbb{R}^{d \times d}$ in $ZE^{-1}Z^T$ is not that much compared to gain in iterations [21].

In the equation 4.13, $ZE^{-1}Z^T$ is the correction term obtained from coarse grid. It contributes to both subdomain terms. Z can be constructed using Nicolaides coarse space [21].

$$Z_i = R_i^T D_i R_i v, \quad i = 1, 2 \tag{4.14}$$

where $D_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ is a diagonal matrix and $v \in \mathbb{R}^n$ is a vector with all 1's. n_i is number of grid points in $i-$ th subdomain.

4.5. Connection with the PKS package

In the PKS package, an Schwarz domain decomposition method is implemented with the following properties [11]:

- Parallel Additive Schwarz (AS) method (also called block Jacobi preconditioner) is used in the PKS package. Therefore the PCG solver can also be called as CG-Schwarz method.
- In the preconditioning step, the sub-domain solutions are obtained by solving the linear system inaccurately using ILU(0).
- PKS only supports Dirichlet interface transmission conditions.
- PKS has been parallelized using MPI (Message Passing Interface) parallel programming paradigm. It also supports OpenMP. However, in all the numerical experiments done in chapter 6 and 7, we use one thread in OpenMP.
- Load balance: PKS support two types of domain decomposition (partitioning) methods: Uniform Partitioning and Recursive Coordinate Bisection (RCB) partitioning [18]. Since there are so many inactive cells in the model (refer figure 4.3), the uniform partitioning can create some idle processes. It will suffer from the load imbalance. Therefore RCB is used for the domain decomposition to provide optimal load distribution in all the processes .
- One subdomain is assigned to one MPI process in all the numerical experiments done in chapter 6 and 7. In this way PKS is scalable regarding problem size and hardware.
- Exchanging data between the subdomains is done by MPI subroutines, and typically involves communication for each inner CG iteration, ensuring a tight coupling: local (point-to-point) communication for the vector updates and global (all-reduce) communications for computing interior products and evaluating stopping criteria. Since this is done for each inner iteration the expected speed-up in computational time with PKS largely depends on MPI (latency and bandwidth).

4.6. Grid Numbering in PKS

For a domain in figure 4.4, we illustrate the active cell with white background and inactive cells with gray background. In this domain , two types of numbering in the computational grid are defined: Global structured numbering and Global unstructured numbering.

Structured numbering is defined for the whole domain that includes active and inactive cells (refer 4.4a). Therefore, it does not differentiate the presence of inactive cells. We call it structured numbering since it is defined for the cells with regular connectivity. Unstructured numbering is defined only for the active cells in an irregular subdomain (refer 4.4b). The inactive cells are marked with x. The

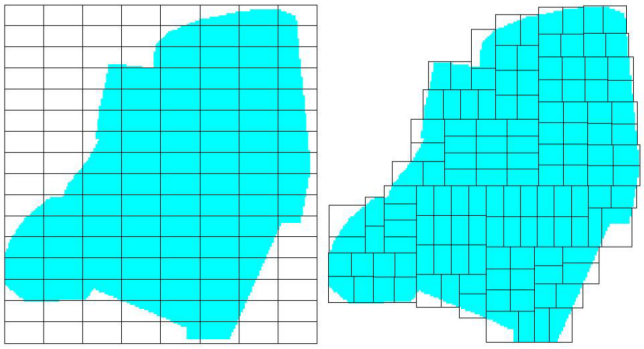


Figure 4.3: Two domain decomposition methods for the steady state NHI model for 128 processes. Left: uniform partitioning; right: Recursive Coordinate Bisection partitioning [33].

purpose of defining unstructured numbering is because we solve for head values only at the active cells. Therefore, the indexes in PKS data structures, such as matrix A and other vectors, are defined using unstructured numbering. All the entries in linear and constant deflation vectors (refer chapter 5) are also defined only for active cells using unstructured numbering .

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

(a) Structured

X	1	2	3	4	X
X	5	6	7	8	X
X	9	10	11	12	X
X	13	14	15	16	X

(b) Unstructured

Figure 4.4: Global grid numbering for $6 \times 6 \times 1$ grid cell

Since all the MPI processes solves on their subdomains locally, the local numbering needs to be defined in all the subdomains. Furthermore, near the boundary, each subdomain requires information from cells in neighboring subdomain (due to 7 point stencil), for example in order to carry out the matrix-vector multiplication. Therefore, they are also included in the local grid numbering. We call these extra cells as GHOST (filled in orange in figure 4.5 and 4.6) cells. Like figure 4.4, two types of grid numbering are defined locally on each subdomain: structured local numbering and unstructured local numbering.

Structured local numbering: Like figure 4.4a, structured numbering is defined for the whole domain that includes active and inactive cells. Assume that for a subdomain with local index of the cell (i, j, k), number of columns and rows in the matrix representing local grid domain are $NCOL_{loc}$ and $NROW_{loc}$

respectively. We define the structured index numbering `StIndex` as follows:

$$\text{StIndex} = NCOL_{loc} * NROW_{loc} * (k-1) + NCOL_{loc} * (i-1) + j$$

For the domain decomposition of the figure 4.4a with two subdomains Ω_1 and Ω_2, structured local numbering has been illustrated in the figure 4.5.

Unstructured local numbering: Unstructured local numbering is used to define the indexes for local data structures in the active cells. All the entries in linear and constant deflation vectors are defined only for active cells using unstructured numbering locally. For the domain decomposition of the figure 4.4b with two subdomains Ω_1 and Ω_2, unstructured local numbering has been illustrated in the figure 4.6. The inactive cells are marked with `X`.

(a) Sub-domain Ω_1 (b) Sub-domain Ω_2

Figure 4.5: Structured local grid numbering

(a) Sub-domain Ω_1 (b) Sub-domain Ω_2

Figure 4.6: Unstructured local grid numbering

Index conversions between structured and unstructured local grid numbering is required for internal computations in the PKS package. Therefore two data structures are defied to convert the indexes: `IXMAP` and `NODEC`.

IXMAP is a one dimensional array used to convert a unstructured index to structured index. GHOST cells needs to be differentiated with the other interior cells in order to successfully compute the interior products of vectors. This is done by returning negative values corresponding to the GHOST cells in the IXMAP array. This mapping is only needed for the active cells so it is not defined for the inactive cells. For e.g., IXMAP(4)=6 for subdomain Ω_1 in figure 4.6a.

NODEC is a three dimensional array used to convert a structured index to unstructured index. It returns zero for the inactive cells. It does not differentiate the ghost cells. For e.g., NODEC(2,2,1)=4 for subdomain Ω_1 in figure 4.5a.

5

Deflation Preconditioning

5.1. Introduction

It is known that the rate of convergence of an iterative method directly depends upon the condition number of matrix A given by λ_n/λ_1, where λ_n is the largest eigenvalue and λ_1 is the smallest eigenvalue. To improve convergence, we need to decrease the condition number of A. It can be achieved either by increasing λ_1 or by reducing λ_n. In NHI model matrix A (refer to matrix A in the appendix), there are extremely small eigenvalues. In DD context, these small eigenvalues of A represent some global information which has to handled efficiently [6].

With an aim to remove the harmful eigenvectors, we define a deflation subspace constructed by the span of eigenvectors corresponding to small eigenvalues. We deflate the deflation subspace from the solution subspace by projecting the harmful eigendirections out of the residual. Thus we remove these extreme small eigenvalues from the linear system. In practice, the smallest eigenvalues almost become zero. Thus the condition number decreases since the new smallest eigenvalue (zero eigenvalues are not counted) is larger than the original smallest eigenvalue. We solve this new, well-conditioned linear system. It is expected to converge in fewer iterations than the original system. We consider the Deflation preconditioner in the form of a projector P used by Vuik [35].

The structure of this chapter is as follows. We start by explaining the basics of deflation in 5.2. In section 5.3, we present different methods to choose the deflation vectors. The section 5.4 outlines the Deflation algorithm. In the section 5.5, we give the implementation aspects of the Deflation algorithm. At the end of this chapter: section 5.6, we present the how Deflation has been implemented in the PKS package.

5.2. Deflation Basics

Let us consider the linear system again given in equation 5.1

$$Au = f \tag{5.1}$$

We assume $A \in \mathbb{R}^{n \times n}$ to be symmetric positive definite (SPD) matrix since a SPD matrix arises in the Control Volume Finite Difference Discretization in MODFLOW. We define a deflation matrix $Z \in \mathbb{R}^{n \times m}$, where $m << n$ in practice. The columns in Z are called deflation vectors and their span is the deflation subspace. The deflation vectors are linearly independent to each other, hence the rank of Z is m. We also define a matrix $E \in \mathbb{R}^{m \times m}$ such that $E = Z^T AZ$.

We define an operator $P = AZE^{-1}Z^T$. We verify that P is a projector since,

$$P^2 = AZE^{-1}Z^T * AZE^{-1}Z^T = AZE^{-1}EE^{-1}Z^T = AZE^{-1}Z^T = P$$

The projector P projects an input vector $v \in \mathbb{R}^n$ to the deflation subspace. Thus the range of P is the deflation subspace. However, we are not interested in the deflation subspace. We are interested in the subspace that is orthogonal to the deflation subspace, i.e., the null space of P. Fortunately, it is not that hard to accomplish. The complimentary projector to P, P_1 projects exactly on the null space of P [31]. It is defined as

$$P_1 = I - P = I - AZE^{-1}Z^T \tag{5.2}$$

Let $\mathcal{Z}$ be the subspace formed by spanning the columns of Z. From equation 5.3 we see that, the null space of $P_1 A$ is the subspace $\mathcal{Z}$. We also know that all the harmful eigendirections belong to the subspace $\mathcal{Z}$. Thus multiplying these eigendirections with the operator $P_1 A$ yields a zero vector. Therefore, the null space $\mathcal{Z}$ never enters into the iteration, and the corresponding zero eigenvalues do not hamper the CG convergence. Practically, m small eigenvalues of $P_1 A$ becomes zero [23]. Therefore, the condition number of $P_1 A$ becomes $\lambda_n / \lambda_{m+1}$ which is smaller than the condition number of A i.e. λ_n / λ_1.

$$\begin{aligned} P_1 AZ &= (I - AZE^{-1}Z^T)AZ, \\ &= AZ - AZE^{-1}E, \\ &= \underline{0}. \end{aligned} \tag{5.3}$$

Now we define an another projector P_2

$$P_2 = P_1^T = I - ZE^{-1}Z^T A, \quad \text{it follows} \quad P_1 A = AP_2. \tag{5.4}$$

The solution vector u can be seen as summation of projection of u on projector P_2 and its complementary projector $I - P_2$ given by

$$u = (I - P_2)u + P_2 u. \tag{5.5}$$

The first term in the equation 5.5 can be computed immediately since

$$(I - P_2)u = ZE^{-1}Z^T Au = ZE^{-1}Z^T f$$

To compute the second term in the equation 5.5, we solve the deflated system given in equation 5.6. $\tilde{u}$ stands for the solution of the deflated system. It is easier to see that $P_2 u = P_2 \tilde{u}$ [20]. So we compute $P_2\tilde{u}$ and substitute it as second term in equation 5.5. Adding the first and second term we obtain u.

$$P_1 A\tilde{u} = P_1 f \tag{5.6}$$

We have general remarks about the Deflation method:

- This method requires additional computation of $ZE^{-1}Z^T b$ but since usually $m << n$ and $E \in \mathbb{R}^{m\times m}$, the overhead is minimal.
- $P_1 A$ is singular in the deflated system 5.6 since m eigenvalues are zero. However, it is consistent since the same projection P_1 is applied to both sides of the nonsingular system, $Au = f$. Hence $\tilde{u}$ can be obtained by solving the equation 5.6 [27].
- The matrix $P_1 A$ is singular so $\tilde{u}$ in equation 5.6 can contain arbitrary components in the null space $\mathcal{Z}$ and is not unique. However, since the projected solution $P_2\tilde{u}$ is unique, the arbitrary components of null space $\mathcal{Z}$ in $\tilde{u}$ does not create any problem [23].

5.3. Choosing Deflation Vectors

The choice of deflation vector is crucial for the success of the deflation method. Essentially, we are interested in removing eigenvalues corresponding to m small eigenvalues from the solution subspace. Naturally, the thought arises to choose the eigenvectors as columns in the Z matrix. In this case, the proof of the fact that m eigenvalues become zero can be found in section 5.2 in [29]. In particular, the matrix $Z \in \mathbb{R}^{n\times m}$ should satisfy the following properties:

- The construction of Z should be inexpensive and problem independent.
- The span of columns in Z should approximate the eigenspace corresponding to the m small eigenvalues as closely as possible.
- The matrix Z should be chosen such that $E = Z^T AZ$ is non-singular since the matrix E needs to be decomposed into lower(L) and upper(U) triangular matrices.
- The more sparse the Z is, the lesser time it will take to perform the computation of subroutines involving Z matrix. It helps to lower the increase in the wall-clock time per PCG iteration.

- Z should be chosen such that the Deflation method does not cause too much overhead in wall clock time per PCG iteration.
- Communication of deflation vectors in Z to the neighboring subdomains should not be costly.
- Storing Z should be memory efficient.

The deflation vectors Z can be constructed in different ways such as eigenvalue deflation [29], physics-based deflation, algebraic deflation vectors. Computation of eigenvectors is costly for large matrices, so other techniques are used to obtain deflation vectors. In this section, we approximate the eigenvectors using constant and linear deflation vectors. Constructing Z with constant deflation vectors is straightforward to understand and implement. It is also known as subdomain deflation, and references can be found in Frank [24] and Nicolaides [28].

5.3.1. Constant Deflation Vectors

As mentioned before, the eigenvectors are expensive to compute. Hence, we need to think of other possible ways to construct Z satisfying the requirements from section 5.3, whose columns approximates the harmful eigenspace. One possible approach is to approximate eigenvectors with constant deflation vectors in each subdomain. Each subdomain Ω_j has a constant deflation vector z_j defined as:

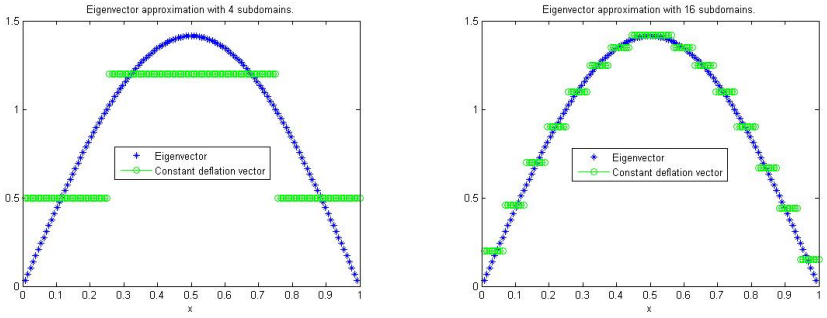
$$z_{ij} = \begin{cases} 1, & x_i \in \Omega_j \\ 0, & x_i \notin \Omega_j \end{cases} \quad j = 1, 2, ...m \tag{5.7}$$

The matrix $Z \in \mathbb{R}^{n \times m}$ is constructed using the above deflation vectors. The structure of Z is shown in equation 5.8 for a general one dimensional (1D) domain where each column represents one subdomain. We can see that the entries in each column are one at grid index of its corresponding subdomain and zero at grid indexes of other subdomains.

$$Z = \begin{bmatrix} & & & \\ z_1 & z_2 & \cdots & z_m \\ & & & \end{bmatrix} = \begin{bmatrix} \underline{1} & \underline{0} & \cdots & \underline{0} \\ \underline{0} & \underline{1} & \cdots & \underline{0} \\ \vdots & \vdots & \ddots & \vdots \\ \underline{0} & \underline{0} & \cdots & \underline{1} \end{bmatrix} \tag{5.8}$$

Now we want to see how well a harmful eigenvector is approximated using constant deflation vectors in different subdomains. For illustration, we take a 1D Poisson problem with homogeneous Dirichlet boundary conditions on $[0, 1]$ and using 128 grid intervals. We plot the eigenvector corresponding to smallest eigenvalue in the figure 5.1. We approximate this eigenvector using 4 and 16 constant deflation vectors. We observe that we can approximate it in a better way using 16 deflation vectors instead of using 4 deflation vectors. This illustration gives us an indication that we can expect the Deflation method to work better for a higher number of subdomains. Since in all of our experiments in chapter 6 and 7, we use one MPI process for one subdomain, we expect that the deflation method works better with

the higher number of processes. Throughout this report, we use the word subdomain and process interchangeably.



(a) $m = 4$ (b) $m = 16$

Figure 5.1: Approximation of eigenvector with smallest eigenvalue 9.86 using constant deflation vectors in 1D Poisson Problem

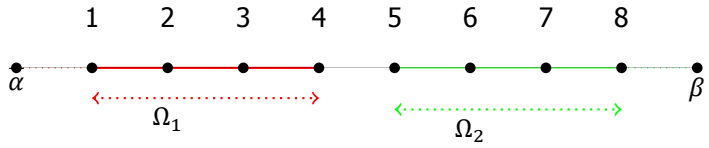


Figure 5.2: 1D grid to illustrate deflation vectors

Now we give an example illustrating the deflation vectors. In the figure 5.2, we have two subdomains, each containing 4 grid points. The local constant deflation vectors z_j, where $j = 1, 2$ and global Deflation matrix Z are defined as

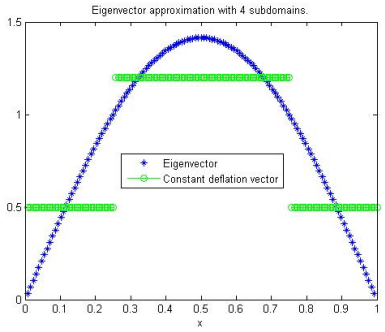
$$z_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \quad z_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

The colors used above are consistent with the figure 5.2.

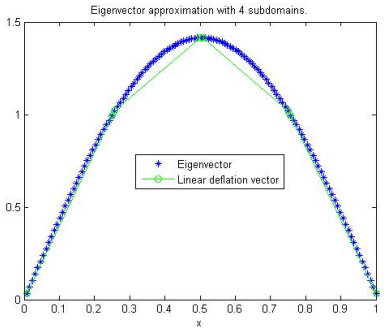
5.3.2. Linear Deflation Vectors

Constant deflation vectors provide a rough approximation to eigenvector corresponding to the small eigenvalue, so these eigendirections are not completely canceled from the solution subspace. Hence, we approximate these eigenvectors with

linear deflation vectors [32]. Linear deflation vectors provide a better approximation to these eigenvectors and are expected to cancel the harmful eigendirections more accurately. We illustrate this behavior in the figure 5.3 for 4 subdomain (deflation vectors) case.

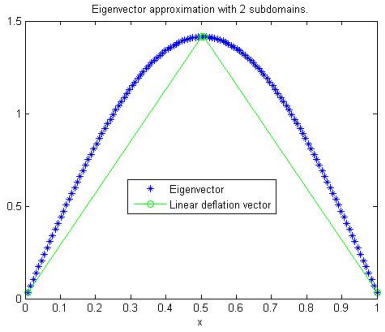


(a) Constant deflation vectors

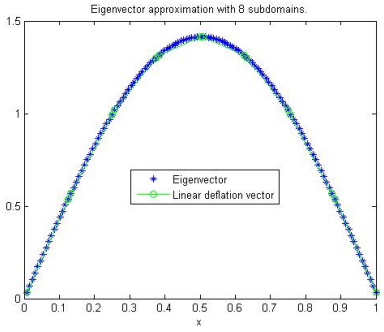


(b) Linear deflation vectors

Figure 5.3: Approximation of eigenvector with smallest eigenvalue 9.86 using deflation vectors in 1D Poisson Problem



(a) 2 linear deflation vectors



(b) 8 Linear deflation vectors

Figure 5.4: Approximation of eigenvector with smallest eigenvalue 9.86 using linear deflation vectors in 1D Poisson Problem

As in the case of constant deflation vectors, more linear deflation vectors gives a better approximation to the harmful eigenvector. It has been illustrated in the figure 5.4.

The harmful eigenvector is approximated in using notation from figure 5.2, the local constant deflation vectors $z_j, j = 1, 2$ and global matrix Z are defined as

$$z_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 3 \\ 1 & 4 \\ 0 & 0 \end{bmatrix}, \quad z_2 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 1 & 4 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 1 & 4 \end{bmatrix}$$

We observe that each subdomain $i = 1, 2$ has two deflation vectors per subdomain. The first vector is constant and the second vector is varying linearly in the x-direction. Since the grid is one-dimensional, two vectors can approximate the eigenvector correctly. In higher dimensions, we need more linear deflation vectors to approximate the eigenvector. In 2D, we need an extra deflation vector varying linearly in the y-direction, so in total, we need three deflation vectors per subdomain. In 3D, we need two additional deflation vectors varying linearly in the y and z-direction, so in total, we need four deflation vectors per subdomain. In general for d-dimensional domain, we need $d+1$ deflation vectors per subdomain. In these vectors, one vector is constant and the each of remaining d deflation vector varies linearly in one direction and constant in other $d-1$ directions.

Compared to the constant deflation vectors, Linear deflation vectors approximate the harmful eigenvectors more accurately, but it comes at an extra cost. Linear deflation vectors create more deflation overhead than constant deflation vectors since the E matrix has grown from NRPROC to d ×NRPROC, where NRPROC is the number of processes/subdomains.

5.4. Deflation algorithm

In this section, we define the Deflation algorithm. The flow of explanation of the algorithm is similar to the Deflation algorithm in [29].

Using initial solution vector $u^{(0)}$, the original LSE $Au = f$ can be written as

$$Ax = r^{(0)}, \text{ where } x = u - \tilde{u}^{(0)}, \; r^{(0)} = f - Au^{(0)} \tag{5.9}$$

We use $\sim$ symbol to denote the vectors obtained in the process of solving the deflated system given in equation 5.6. Using initial solution vector $\tilde{u}^{(0)} = u^{(0)}$, i.e. $\tilde{x}^{(0)} = \underline{0}$, the deflated system is written as

$$P_1 A\tilde{x} = P_1 r^{(0)}, \text{ where } \tilde{x} = \tilde{u} - u^{(0)}, \; r^{(0)} = f - Au^{(0)} \tag{5.10}$$

We aim to solve for x in the original system given in equation 5.9. We do not answer it directly but solve for $\tilde{x}$ in the deflated system given in equation 5.10 as the condition number of $P_1 A$ has reduced after removing the extreme small eigenvalues in the matrix A.

Using equation 5.5 for x, we project the x on P_2 and its complementary projector $I - P_2$ as

$$\begin{aligned} x &= (I - P_2)x + P_2 x \\ &= (I - P_2)x + P_2 \tilde{x} \\ &= (I - P_2)(u - u^{(0)}) + P_2 \tilde{x} \\ &= ZE^{-1}Z^T Au - ZE^{-1}Z^T Au^{(0)} + (I - ZE^{-1}Z^T A)\tilde{x} \\ &= ZE^{-1}Z^T(f - Au^{(0)}) + (I - ZE^{-1}Z^T A)\tilde{x} \\ &= ZE^{-1}Z^T r^{(0)} + (I - ZE^{-1}Z^T A)\tilde{x} \end{aligned} \tag{5.11}$$

Since initial residual vector $r^{(0)}$ is known at the start of PCG algorithm, the first term in 5.11 can be easily computed. To compute, the second term we need to solve the LSE given in 5.10.

Now we define q_1 and q_2 as

$$q_1 := E^{-1}Z^T r^{(0)}, \qquad q_2 := E^{-1}Z^T A\tilde{x}$$

We substitute q_1 and q_2 in 5.11 to obtain 5.12.

$$u = Z(q_1 - q_2) + \tilde{x} + u^{(0)} \tag{5.12}$$

In order to apply the deflation technique in the PCG algorithm 1, we need to compute $q_1, \tilde{x}$ and q_2. We call deflation enabled PCG algorithm as DPCG algorithm. The DPCG algorithm requires following additional work compared to the PCG algorithm:

- Computation of q_1 at the beginning of PCG algorithm.
- Computation of $\tilde{x}$ by solving the deflated system 5.10.
- Computation of q_2 by using $\tilde{x}$ at the end of PCG algorithm.

The additional work in DPCG is carried out in following three phases:

- Deflation Pre-Processing Phase or Deflation Init,
- Deflation Run-Time Phase,
- Deflation Post-Processing Phase or Deflation End.

5.4.1. Deflation Pre-Processing Phase

The first part of additional work in called as deflation pre-processing phase. We will also call this phase as Deflation Init in this report. It is carried out before the first CG iteration. In this phase, the required initial conditions to solve the deflated linear system given in equation 5.10 are set. We assume that initial solution of deflated system is zero, i.e $\tilde{x}^{(0)} = \underline{0}$. The initial residual of the deflated system is required in the first DPCG iteration. Using equation 5.10, it is calculated as

$$\begin{aligned}\tilde{r}^{(0)} &:= P_1 r^{(0)} - P_1 A\tilde{x}^{(0)} \\ &= P_1 r^{(0)} \quad (\text{since } \tilde{x}^{(0)} = 0) \\ &= (I - AZE^{-1}Z^T)r^{(0)} \\ &= r^{(0)} - AZE^{-1}Z^T r^{(0)} \\ &= r^{(0)} - AZq_1\end{aligned} \tag{5.13}$$

The AZ matrix in 5.13 is required in the deflation run time phase, so we save it in the deflation pre-processing phase to reduce the redundant computation. The same AZ is used later the deflation run time phase.

To compute q_1 in 5.13, the following linear system needs to be solved.

$$Eq_1 = Z^T r^{(0)} \tag{5.14}$$

When we use constant deflation vectors, structure of E is given by

$$E(i,j) = \text{ is } \begin{cases} \neq 0, & \text{if } i = j, \\ \neq 0, & \text{if } i \text{ and } j \text{ are neighboring subdomains,} \\ = 0, & \text{otherwise.} \end{cases} \tag{5.15}$$

The dimension of E depends upon the number of deflation vectors per subdomain. For d deflation vectors per subdomain and m number of subdomains, $E \in \mathbb{R}^{m*d \times m*d}$. The maximum value of d can be four when the domain is three dimensional. Our maximum number of subdomains is in the order of 100, so the maximum size of E is in the order of 400. This size is not large, so we can use a direct method to solve the LSE 5.14. For higher number of subdomains, we observe from 5.15 that the matrix E is sparse. Therefore, we use sparse LU factorization to solve the LSE 5.14.

For a simple case in CDPCG method, we investigate the structure of E. We notice that a non-directed graph arising from the domain decomposition gives the construction of E. Hence, the matrix E is symmetric. For symmetric E, the sparse Cholesky Factorization [2] is cheaper than sparse LU Factorization since it exploits the symmetry of E. However, since the matrix E is small, the solution of LSE 5.14 is not time-consuming, so we stick to the LU sparse solve.

The right-hand side in 5.14 is constructed by taking the dot products of the deflation vectors with the initial residual of the original system. Each subdomain (or process) stores the local dot products. These local dot products are gathered to form the term $Z^T r^{(0)}$. Each process (subdomain) has access to complete E matrix and $Z^T r^{(0)}$, so the same system 5.14 is solved by each process in parallel. We could also solve the linear system by one process and broadcast q_1 to other processes, but we need further communication in this case. Since the linear system is not that large, we assume that the communication is more expensive than the computation

in solving the linear system.

Once the lower triangular matrix L, the upper triangular matrix U and the vector $Z^T r^{(0)}$ are constructed, we solve the 5.14 with forward and backward substitution in parallel by each process.

5.4.2. Deflation Run-Time Phase

The second part of additional work in called as deflation runtime phase. This phase is introduced since we observe that introduction of Deflation has changed the linear system as

$$Ax = r^{(0)} \xrightarrow{Deflation} P_1 A\tilde{x} = P_1 r^{(0)} \tag{5.16}$$

We notice that the matrix has changed from A to $P_1 A$. Therefore, the matrix vector multiplication has changed from $v^{(k)}$ to $P_1 v^{(k)}$ in the DPCG algorithm as given in 5.17. We define $q_3^{(k)} := E^{-1} Z^T v^{(k)}$ to simplify 5.17.

$$\begin{aligned} P_1 v^{(k)} &= v^{(k)} - AZE^{-1}Z^T v^{(k)} \\ &= v^{(k)} - AZq_3^{(k)} \end{aligned} \tag{5.17}$$

Now we need to compute $P_1 v^{(k)}$ instead of $v^{(k)}$ in k^{th} DPCG iteration. In the Deflation Run time phase, we compute $q_3^{(k)}$ by solving the following linear system:

$$Eq_3^{(k)} = Z^T v^{(k)} \tag{5.18}$$

We construct the vector $Z^T v^{(k)}$ similarly as in deflation pre-processing phase. We use the same lower triangular matrix L and upper triangular matrix U to solve 5.18. The AZ matrix constructed in the deflation pre-processing phase is used in 5.17.

5.4.3. Deflation Post-Processing Phase

The last part of additional work in called as deflation post-prepossessing phase. We will also call this phase as Deflation End in this report. In this phase, we calculate the solution for the original problem 5.2 after solving the deflated system.

$$Eq_2 = Z^T A\tilde{x} \tag{5.19}$$

We obtain $\tilde{x}$ after solving the deflated system 5.10. We construct the vector $Z^T A\tilde{x}$ similarly as in deflation pre-processing phase. We use the same lower triangular matrix L and upper triangular matrix U to solve 5.19 for q_2.

The obtained vectors $\tilde{x}$, q_1 and q_2 after solving 5.2, 5.14 and 5.19 respectively, are substituted in 5.12 to get the solution vector u in the original linear system 5.2.

Algorithm 5 Deflated PCG Algorithm

procedure DPCG($A, f, u^{(0)}, tol, k_{max}, M, Z$)
  $r^{(0)} = f - Au^{(0)}$, k=1 ▷ Initialization
  if (deflation) **then** ▷ Deflation pre-processing phase
    $\tilde{u}^{(0)} = u^{(0)}$
    $u^{(0)} = 0$
    Decompose $Z^T AZ$ ($d \times$ LC, GC) = $\tilde{L}\tilde{U}$ ▷ d=3 for NHI model in LDPCG
    solve $\tilde{L}\tilde{q}_1 = Z^T r^{(0)}$ (GC); $\tilde{U}q_1 = \tilde{q}_1$ ▷ GC: Global communication
    $r^{(0)} = r^{(0)} - AZq_1$ ▷ LC: Local communication
  end if
  while ($k < k_{max}$ **and** $\|r^{(k-1)}\| > tol$) **do**
    $z^{(k-1)} = M^{-1}r^{(k-1)}$ ▷ Preconditioning with Additive Schwarz
    if $k = 1$ **then**
      $p^{(1)} = z^{(0)}$
    else
      $\beta_k = \frac{(r^{(k-1)})^T z^{(k-1)}}{(r^{(k-2)})^T z^{(k-2)}}$
      $p^{(k)} = z^{(k-1)} + \beta_k p^{(k-1)}$ ▷ Search direction
    end if
    $v^{(k)} = Ap^{(k)}$
    if (deflation) **then** ▷ Deflation run time phase
      solve $\tilde{L}\tilde{q}_3^{(k)} = Z^T v^{(k)}$ (GC); $\tilde{U}q_3^{(k)} = \tilde{q}_3^{(k)}$
      $v^{(k)} = v^{(k)} - AZq_3^{(k)}$
    end if
    $\alpha_k = \frac{(r^{(k-1)})^T z^{(k-1)}}{(p^{(k)})^T v^{(k)}}$
    $u^{(k)} = u^{(k-1)} + \alpha_k p_k$ ▷ Iterate update
    $r^{(k)} = r^{(k-1)} - \alpha_k v^{(k)}$ ▷ Residual update
    $k = k + 1$
  end while
  k = k -1
  if (deflation) **then** ▷ Deflation post-processing phase
    solve $\tilde{L}\tilde{q}_2 = Z^T Au^{(k)}$ (LC, GC); $\tilde{U}q_2 = \tilde{q}_2$
    $u^{(k)} = u^{(k)} + \tilde{u}^{(0)} + Z(q_1 - q_2)$
  end if
  return $u^{(k)}$ ▷ The converged solution
end procedure

5.5. Implementation

In this section, we discuss the implementation details of deflation subroutines. As defined in the algorithm 5, the deflation can be carried out in three phases using following subroutines/functions. In red color we show how each subroutine is computed in parallel and in red we mark communication detail across MPI processes.

- Deflation Init:
  - Construction of AZ: local exchange of linear deflation vector across neighboring subdomains,
  - Construction of E matrix using AZ: global communication of local E using `MPI_Allgather`,
  - Sparse LU decomposition of E matrix: same computation in parallel by all MPI processes,
  - Dot product to set the RHS of linear system solve: global communication of local dot products using `MPI_Allgather`,
  - Forward and backward substitution: same computation in parallel by all MPI processes,
  - Multiplication of AZ with q_1: individual computation by each MPI processes,
  - Residual and solution vector update: individual computation by each MPI processes.
- Deflation Runtime:
  - Dot product to set the RHS of linear system solve: global communication of local dot products using `MPI_Allgather`,
  - Forward and backward substitution: same computation in parallel by all MPI processes,
  - Multiplication of AZ with $q_3^{(k)}$:individual computation by each MPI processes,
  - Vector $v^{(k)}$ update: individual computation by each MPI processes.
- Deflation End:
  - Matrix vector multiplication individual computation by each MPI processes involving local communication of the vector,
  - Dot product to set the RHS of linear system to solve: global communication of local dot products using `MPI_Allgather`,
  - Forward and backward substitution: same computation in parallel by all MPI processes,
  - Multiplication of Z with q_2: individual computation by each MPI processes,

– Two vector updates of solution vector $u^{(k)}$: individual computation by each MPI processes .

We give implementation details of above subroutines in the serial and parallel case for a 1D Poisson problem. Out of subroutines mentioned above, construction of E is the most complicated, so we show this with a simple example. For other subroutines, we state the procedure but do not give an accurate example.

For simplicity we assume two subdomains, each containing three grid points as shown in figure 5.5. Further we assume homogeneous Dirichlet boundary conditions, i.e. $u(\alpha) = 0$ and $u(\beta) = 0$. We approximate the harmful eigenvector using one constant and one algebraic deflation vector per subdomain.

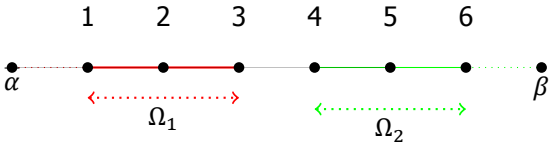


Figure 5.5: Global grid numbering

5.5.1. Serial Implementation of E

The coefficient matrix A (assuming $h = 1$) and the Deflation matrix Z is given as

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}, \; Z = \begin{bmatrix} z_1 & z_2 & z_3 & z_4 \end{bmatrix} \tag{5.20}$$

We define the columns of the matrix Z as

$$z_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \; z_2 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ for } \Omega_1; \qquad z_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \; z_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} \text{ for } \Omega_2; \tag{5.21}$$

From equation 5.20 and 5.21, we obtain

$$AZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 4 & -1 & -1 \\ -1 & -3 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 \end{pmatrix}, \quad E = Z^T AZ = \begin{pmatrix} 2 & 4 & -1 & -1 \\ 4 & 12 & -3 & -3 \\ -1 & -3 & 2 & 4 \\ -1 & -3 & 4 & 12 \end{pmatrix}. \tag{5.22}$$

5.5.2. Parallel Implementation of E

The subdomain Ω_i is processed by the process P_i, where $i = 1, 2$. From figure 5.6, we notice that update of grid point 3 requires the information from grid point 4, therefore ghost grid point 4 is allocated for process P_1. Similarly, ghost grid point 3 is allocated for process P_2.

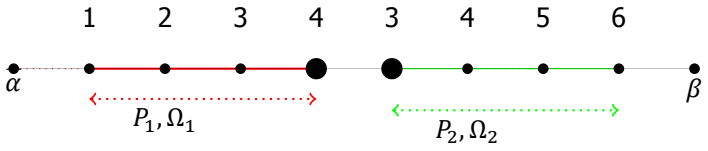


Figure 5.6: Global grid numbering

Locally each process maintains local grid numbering as shown in the figure 5.7. In this numbering, the ghost points (4 for P_1 and 1 for P_2) are also counted. Usually, we differentiate the ghost points from non-ghost points with a flag. (called IXMAP in PKS: refer section 4.6)

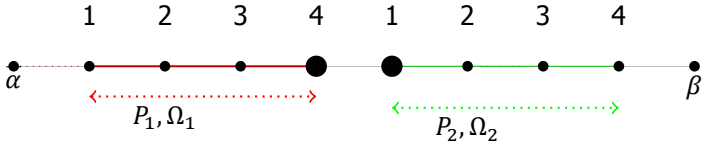


Figure 5.7: Local grid numbering

For the remaining part of this subsection, we will use the grid numbering from figure 5.7. Locally for $i = 1$ and 2, process P_i has a matrix A_i and a constant deflation vector z_{ic} and a linear deflation vector z_{il}. in the deflation vectors, c denotes constant and l denotes linear in x-direction. All the deflation vectors have zero value at ghost grid points.

The local matrices are given by

$$A_i = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \text{ for } i = 1, 2. \tag{5.23}$$

The local deflation vectors are given by

$$z_{1c} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix},\ z_{1l} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 0 \end{pmatrix} \text{ for } \Omega_1; \qquad z_{2c} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix},\ z_{2l} = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} \text{ for } \Omega_2; \tag{5.24}$$

From 5.23 and 5.24, we have

$$A_1 z_{1c} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ -1 \end{pmatrix},\ A_1 z_{1l} = \begin{pmatrix} 0 \\ 0 \\ 4 \\ 3 \end{pmatrix} \text{for } \Omega_1; \qquad A_2 z_{2c} = \begin{pmatrix} -1 \\ 1 \\ 0 \\ 1 \end{pmatrix},\ A_2 z_{2l} = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 4 \end{pmatrix} \text{for } \Omega_2; \tag{5.25}$$

Now we need to construct the matrix $E \in \mathbb{R}^{4\times4}$. The first two rows of E are constructed by P_1 and remaining two rows by P_2. The general structure of E is given as

$$E = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix}, \quad E_{ij} \in \mathbb{R}^{2\times2}, \text{ for } i,j = 1,2 \tag{5.26}$$

The matrix E_{11} and E_{22} denotes the local coupling in P_1 and P_2 respectively. Using 5.24 and 5.25, they are given as

$$E_{11} = \begin{bmatrix} z_{1c}^T A_1 z_{1c} & z_{1c}^T A_1 z_{1l} \\ z_{1l}^T A_1 z_{1c} & z_{1l}^T A_1 z_{1l} \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 4 & 12 \end{bmatrix} \tag{5.27}$$

$$E_{22} = \begin{bmatrix} z_{2c}^T A_2 z_{2c} & z_{2c}^T A_2 z_{2l} \\ z_{2l}^T A_2 z_{2c} & z_{2l}^T A_2 z_{2l} \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 4 & 12 \end{bmatrix} \tag{5.28}$$

The matrix E_{12} and E_{21} denotes the global coupling in P_1 and P_2 respectively. For example, in order to compute the entries in E_{12}, we need to extend the deflation vectors z_{2c} and z_{2l} over the subdomain Ω_1. Since z_{2c} and z_{2l} have a non-zero at the global grid index 4, it needs to be communicated to process P_1. We define local vectors denoted by $z_{2c\to1}$ (read this as z_{2c} extended for Ω_1) and $z_{2l\to1}$ for subdomain Ω_1. These vectors have zero value at all non-ghost points. Similarly to compute E_{21}, we extend z_{1c} and z_{1l} for Ω_2 denoted by $z_{1c\to2}$ and $z_{1l\to2}$ respectively. They are defined as:

$$z_{2c\to1} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix},\ z_{2l\to1} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \text{ for } \Omega_1; \qquad z_{1c\to2} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix},\ z_{1l\to2} = \begin{pmatrix} 3 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ for } \Omega_2; \tag{5.29}$$

Multiplying the extended deflation vectors in 5.29 with the local matrices A_1 and A_2 results in

$$A_1 z_{2c\to1} = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 2 \end{pmatrix},\ A_1 z_{2l\to1} = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 2 \end{pmatrix} \text{for } \Omega_1;\ A_2 z_{1l\to2} = \begin{pmatrix} 2 \\ -1 \\ 0 \\ 0 \end{pmatrix},\ A_2 z_{1l\to2} = \begin{pmatrix} 6 \\ -3 \\ 0 \\ 0 \end{pmatrix} \text{for } \Omega_2; \tag{5.30}$$

Using 5.29 and 5.30, the global coupling matrices are constructed as

$$E_{12} = \begin{bmatrix} z_{1c}^T A_1 z_{2c\to 1} & z_{1c}^T A_1 z_{2l\to 1} \\ z_{1l}^T A_1 z_{2c\to 1} & z_{1l}^T A_1 z_{2l\to 1} \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ -3 & -3 \end{bmatrix} \tag{5.31}$$

$$E_{21} = \begin{bmatrix} z_{2c}^T A_2 z_{1c\to 2} & z_{2c}^T A_2 z_{1l\to 2} \\ z_{2l}^T A_2 z_{1c\to 2} & z_{2l}^T A_2 z_{1l\to 2} \end{bmatrix} = \begin{bmatrix} -1 & -3 \\ -1 & -3 \end{bmatrix} \tag{5.32}$$

Substituting the local coupling matrices E_{11} and E_{22} from 5.27 and 5.28, and the global coupling matrices E_{12} and E_{21} from 5.31 and 5.32 into 5.26, we get the E matrix. This E is same as the matrix E given in the serial implementation in 5.22.

In the context of parallel programming, part of E is computed by different processors. In this example, the process P_1 and P_2 computes $E_1 \in \mathbb{R}^{2\times 4}$ and $E_2 \in \mathbb{R}^{2\times 4}$

$$E_1 = \begin{bmatrix} E_{11} & E_{12} \end{bmatrix}, \qquad E_2 = \begin{bmatrix} E_{21} & E_{22} \end{bmatrix} \tag{5.33}$$

E_1 and E_2 are gathered using `MPI_Allgather` subroutine. It ensures that same copy of E is available to all the processors.

We give general form of E for P number of processes when we approximate the harmful eigenvector using only constant deflation vector. Let A_i be the local matrix, and z_{ic} be the constant deflation vector available for i-th processor locally. Let the $z_{jc\to i}$ be j-th deflation vector extended for subdomain Ω_i. The matrix $E \in \mathbb{R}^{P\times P}$ is constructed as

$$E_{ij} = \begin{cases} z_i^T A_i z_i, & \text{if } i = j \\ z_i^T A_i z_{jc\to i}, & \text{if } i \neq j,\ |i-j| = 1 \\ 0. & \text{if } i \neq j,\ |i-j| \neq 1 \end{cases} \tag{5.34}$$

When we approximate the harmful vector using constant and linear deflation vector in 1D domain, the structure of E is block-wise tridiagonal given as

$$E = \left[\begin{array}{c|c|c|c|c} E_{11} & E_{12} & 0 & \dots & 0 \\ \hline E_{21} & E_{22} & E_{23} & 0 & 0 \\ \hline 0 & E_{32} & E_{33} & E_{34} & 0 \\ \hline \vdots & \ddots & \ddots & \ddots & \ddots \\ \hline 0 & 0 & 0 & \ddots & E_{pp} \end{array}\right] \tag{5.35}$$

The diagonal block matrices E_{ii} and non-diagonal block matrices E_{ij} are global coupling terms defined as

$$E_{ii} = \begin{pmatrix} z_{ic}^T A_i z_{ic} & z_{ic}^T A_i z_{il} \\ z_{il}^T A_i z_{ic} & z_{il}^T A_i z_{il} \end{pmatrix}, \; E_{ij} = \begin{pmatrix} z_{ic}^T A_i z_{jc\to i} & z_{ic}^T A_i z_{jl\to i} \\ z_{il}^T A_i z_{jc\to i} & z_{il}^T A_i z_{jl\to i} \end{pmatrix}, \; i,j = 1 \ldots p \quad (5.36)$$

The diagonal block matrices in E_{ii} are computed locally by i-th processor. Extending the deflation vector of a non-neighboring subdomain on Ω_i, we get $z_{j\to i}$, and since it is a non-neighboring subdomain, the ghost grid point value is also zero in $z_{j\to i}$ along with non-ghost values. It makes the vector $A_i z_{j\to i}$ zero. Therefore, the block matrices E_{ij} corresponding to non-neighboring subdomain becomes zero. In the neighboring subdomains, the ghost value in $z_{j\to i}$ is not zero, hence $A_i z_{j\to i}$ is also not a zero vector, and it makes E_{ij} non-zero.

Each process constructs a local AZ matrix while assembling the E matrix. The first d columns in the AZ comes from the local coupling in each process. The remaining columns originate from the global coupling. For the subdomain Ω_i, we only store the $Az_{j\to i}$ vector only when j is the neighboring subdomain to i. When j is non-neighboring subdomain the vector $Az_{j\to i}$ vanishes. Hence, we do not explicitly store these redundant columns in the AZ matrix. For the neighboring subdomains, we also store the rank of the process j along with $Az_{j\to i}$. In the figure 5.7, the AZ matrix is given as

$$AZ_i = \begin{bmatrix} A_i z_{ic} & A_i z_{il} & A_i z_{jc\to i} & A_i z_{jl\to i} \end{bmatrix}, \quad i,j = 1,2, \; i+j = 2 \quad (5.37)$$

We observe in 5.35 that the structure of E is quite sparse, so we use a sparse LU decomposition to generate the lower triangular matrix L and upper triangular matrix U.

5.5.3. Computation of Other Subroutines

In order to from the vector $Z^T r^{(0)}$ in $Eq_1 = Z^T r^{(0)}$ in Deflation Init, each process constructs a small vector of d elements consisting of local dot products of deflation vector z_i and $r^{(0)}$. All these small vectors are gathered with `MPI_Allgather` and thus the vector $Z^T r^{(0)}$ is available to all the processes.

The equation 5.14 is solved using forward and backward substitution in parallel by all the processes to yield q_1.

$$\text{Forward substitution: solve for } y \text{ in } \; Ly = Z^T r^{(0)}$$
$$\text{Backward substitution: solve for } q_1 \text{ in } \; Lq_1 = y$$

This AZ computed in the Deflation Init subroutine is used to compute AZq_1 and $AZq_3^{(k)}$ in Deflation Init and Deflation Runtime respectively. In the Deflation end subroutine, we require a matrix vector product $Au^{(k)}$ to set the right-hand side $Z^T A\tilde{x}$ in equation 5.19.

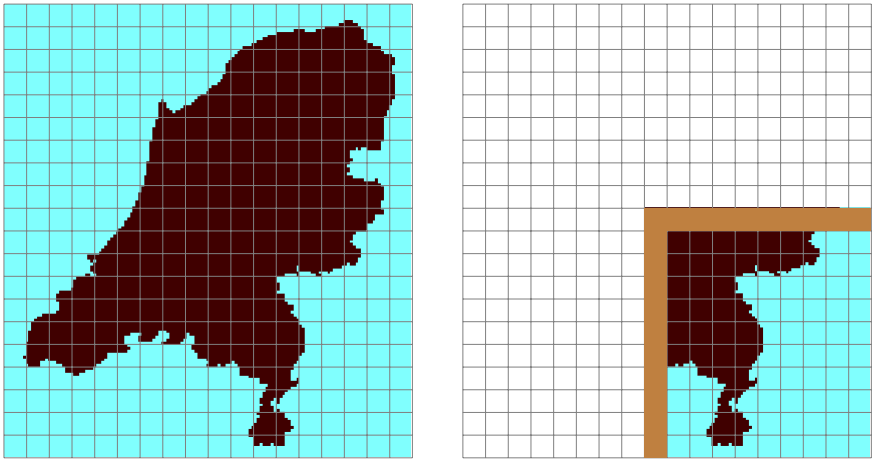
5.6. Connection with the PKS package

In this section, we present the connection of this chapter with the PKS package. We also give implantation detail.

5.6.1. Choosing Deflation Vectors for NHI SS model

We have implemented CDPCG and LDPCG in the PKS solver. In this section, we describe how we choose the constant and linear deflation vectors on the domain of the Netherlands used in NHI SS model in chapter 7.

We allocate the deflation vectors in each subdomain. However, to illustrate the deflation vectors in one subdomain, we extract the east-south part of the Netherlands from the top layer L_1 (refer the figure 5.8). This layer needs one layer (ghost layer) of grid points from left and adjacent top subdomain. We have colored these ghost layers in brown as shown in figure 5.8b.



(a) Layer L_1 in the domain of Netherlands (b) One subdomain from layer L_1 in a)

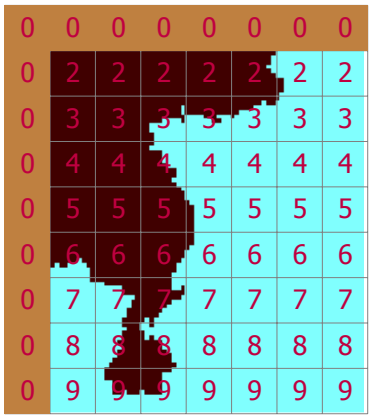
Figure 5.8: Extraction of one subdomain from the domain of the Netherlands

We show the constant and linear deflation entries in the figure 5.9. All the deflation vectors have zero entries in the ghost cells (shown in brown stripes). Constant deflation vectors have value 1 at all the interior points. The linear deflation vectors are defined in 3 directions: x, y and z. The linear-x (refer figure 5.9b) varies linearly only in the x direction, being constant in y and z direction. Similarly, the linear-y and linear-z deflation vectors vary linearly only in y and z direction respectively and being constant in remaining two directions. Since the z direction values are constant for layer L_1, we also observe that the entries in linear-z deflation vector (refer the figure 5.9d) is constant. For instance, th linear-z deflation vector entries will be 2 at all the interior points in L_2. The deflation vector entries in other layers of the NHI SS model (refer to chapter 7) are defined similar way as the layer

L_1.

(a) constant deflation vector

(b) linear-x deflation vector



(c) linear-y deflation vector

(d) linear-z deflation vector

Figure 5.9: Deflation vector entries in the subdomain taken from the figure 5.8b

5.6.2. Development of Deflation in the PKS Package

FORTRAN 90 implementation of `CDPCG` and `LDPCG` methods have been successfully completed in the PKS package in the iMOD [10] software. PKS solver related variables are defined in a FORTRAN module called `PKS7MODULE`. A flag called `DEFLATION` has been added to the PKS7MODULE which can be set to true or false. If it is set to false, the PCG solver (no deflation) is called. If it is set to true, either CDPCG or LDPCG is called. To differentiate among deflated solvers, another integer variable called `NDECVEC` (number of deflation vectors), has been added in the PKSMODULE. NDECVEC can be set to 1, 2, 3 and 4 in 3D models. The description

of the NDEFVEC variable is given below:

- When NDEFVEC is 1, the CDPCG solver is called for both 2D and 3D models.
- When NDEFVEC is set to 2, 3 and 4, LDPCG solver is called for 3D models. For 2D models NDEFVEC should only be set to 2, 3. The code terminates when NDEFVEC is set to 4 for 2D (one layer) models since for one layer models, there is no z direction.
- For 3D models, setting NDEFVEC= 2, LDPCG uses constant and linear deflation vectors in x directions. We also denote it by linear-x deflation in the report.
- For 3D models, setting NDEFVEC= 3, LDPCG uses constant and linear deflation vectors in x and y directions. We also denote it by linear-xy deflation in the report.
- For 3D models, setting NDEFVEC= 4, LDPCG uses constant and linear deflation vectors in x, y and z directions. We also denote it by linear-xyz deflation in the report.

Different groundwater models can be simulated using an executable created after compiling the FORTRAN code. Hydrologists use the executable with another input file, called `RUNFILE`. The RUNFILE contains a description of various model parameters such as coordinates specifying the domain, cell-size, boundary condition, solver settings, etc. Deflation and NDEFVEC flag will be added in the RUNFILE in the near future.

5.6.3. Efficient Storage of Deflation Vectors

For the larger number of subdomains, the structure of Deflation matrix Z gets very sparse. Initially, we stored the matrix slices in each subdomain representing submatrices of Z. It stores redundant zeros in the columns of subdomain matrix Z that comes from an extension of deflation vectors from other subdomains. This approach becomes memory inefficient for smaller cell sizes (finer model).

In our current implementation, we do not store these zeros. We allocate deflation vector (called `ZDEF` in PKS) instead of storing slices of Deflation matrix Z in each subdomain. For example in CDPCG solver, we store only 1's in each subdomain at non-ghost points.

6

Numerical Experiments for the Model Problem

6.1. Introduction

In this chapter, we apply the deflation method described in the preceding chapter, to 2D and 3D Poisson and two layer unit model problems. The aim to do this is to gain insights into the effect of deflation on the number of CG iterations. The Deflation methods have been coded in the iMOD software in FORTRAN. Experiments are carried out on the Dutch National Supercomputer Cartesius [7]. The numerical experiments are done one a 24 core CPU Intel® Xeon® Processor E5-2690 v3 [12]. We have considered the steady-state simulation in this chapter.

The structure of this chapter is as follows. We give the information about batch file options in section 6.2. We do experiments for Poisson problem and 2 layer iMOD unit case in section 6.3 and 6.4 respectively.

6.2. Running Jobs on Cartesius

Jobs used to carry out all the numerical experiments in current and succeeding chapter are run on the Dutch National Supercomputer Cartesius [7] using job scheduling system `Slurm` [15]. The Slurm Workload Manager (formerly known as Simple Linux Utility for Resource Management or SLURM), is a free and open-source job scheduler for Linux and Unix-like kernels. Slurm is the workload manager on about 60 % of the TOP500 [1] supercomputers.

The batch files are submitted to the slurm using sbatch command, as the name sbatch stands for. The batch script may be given to sbatch through a file name on the command line, or if no file name is specified, sbatch will read in a script from standard input. The batch script may contain options preceded with "#SBATCH"

before any executable commands in the script. Our batch file include the following options:

Runtime Command used is `#SBATCH -t <HH:MM:SS>` . It sets the maximum job duration to `HH` hours, `MM` minutes and `SS` seconds. The jobs will be killed once the job duration is over, even if it has not finished.

Job output Command used is `#SBATCH –output=<scriptname>_%j.out`. where %j stands for job id: all the jobs on cartesius are identified with a uniques id called job id. After completion of the job the output file will be created with name <scriptname>_%j.out. We use the command `#SBATCH –output=<scriptname>_%j.out` to see if the job has encountered any error during run. After completion of the job the error file will be created with name =<scriptname>_%j.err. The error file is usually empty if the job run was successful.

Node type Cartesius consists of a large number of batch nodes and a small number of special purpose nodes. For the batch nodes, cartesius differentiates between so-called thin nodes and fat nodes. Fat nodes are used for the jobs requiring more memory whereas thin nodes are used to run normal or small jobs. Each fat node has 256 GB memory and 32 physical cores. Each thin node has 64 GB memory and 24 physical cores. The command used in job script is given below:

```
#SBATCH -p <node type>
```

where <node type> is thin for all experiments for NHI SS model and <node type> is fat for 50m california miamore model since the domain decomposition algorithm (RCB partitioning) requires more memory.

Node constraint Processesors in thin nodes consists of two types: Haswell and Ivy Bridge. The Haswell is Intel's 4^{th} generation micro-architecture whereas Ivy Bridge is 3^{rd} generation micro-architecture. For consistency, our all jobs are run only on the Haswell node. It can be done in the script by adding the following :

```
#SBATCH –constraint=haswell
```

If we do not specify any constraint, the job can run on any combination of nodes within the partition.

Numer of processes We can specify the number of nodes by writing the following in the job script:

```
#SBATCH -N <number of nodes>
```

We can specify the number of total processes by writing the following in the job script:

```
#SBATCH -n <number of processes>
```

Distribution We use block for at the placeholders <type 1> and <type 2> in our job script as follows:

```
#SBATCH –distribution= <type 1> : <type 2>
```

The block distribution method defined for the placeholder <type 1>, distributes tasks in a job to a node such that consecutive tasks share a node. For example, consider an allocation of three nodes each with two CPUs. A four-task block distribution request will distribute those tasks to the nodes with tasks one and two on the first node, task three on the second node, and task four on the third node. Block distribution is the default behavior if the number of tasks exceeds the number of allocated nodes. block is used in the following.

The block distribution method defined for the placeholder <type 2>, distributes tasks to sockets such that consecutive tasks share a socket.

We have added a sample job script file in the appendix section at the end of this report.

6.3. Poisson Problem

As our first experiment, we consider the Poisson Problem given in equation 6.2 since we want to know the behavior of deflation on the number of iterations in the first Picard iteration. In the deflation method, we use constant and linear deflation vectors in our experiments.

6.3.1. Model Description

We start with a one layer model. We denote this layer by L_1. The description of the model parameters are given below:

- The domain of the model represents an area of size 2.5 km ×2.5 km with the coordinates x_{min} = 200000m, x_{max}=202500m, y_{min} = 400000m and y_{max} is 402500m. Here x_{min} denotes starting coordinate in x direction and x_{max} denotes ending coordinate in x direction, similar notation for y-direction as well. We can vary size of the cell. For instance in the figure 6.1, if we keep the cell size to 250 meter, we get 10 cells both in x and y direction.
- There are no no-flow cells in the layer. In the left and right side stripe (column 1 and 10 in figure 6.1), there are constant head cells with head value 1 meter with IBOUND -1. All other cells are variable-head cells with IBOUND 1.
- The transmissivity is 100 m^2 per day, and vertical resistance is 50 days.
- The active modules are well, river, drainage, recharge, overflow, and constant-head.

- There are two discharge (pumping) wells: first at coordinate (201225m,401275m) and second at (200600m,400650m). Both the wells pump water at 2500 cubic meter per stress period.
- The starting head value at all variable-head cells is 1 meter.

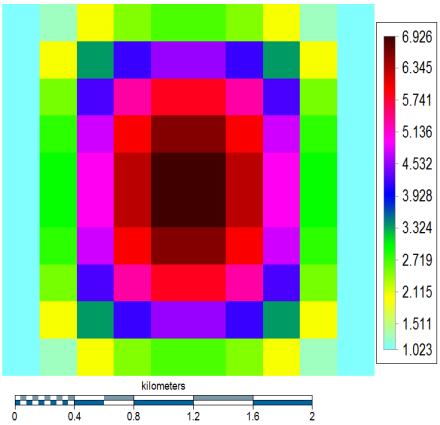


Figure 6.1: Hydraulic head in 2D square domain with 10 ×10 cells and cell-size 250 meter

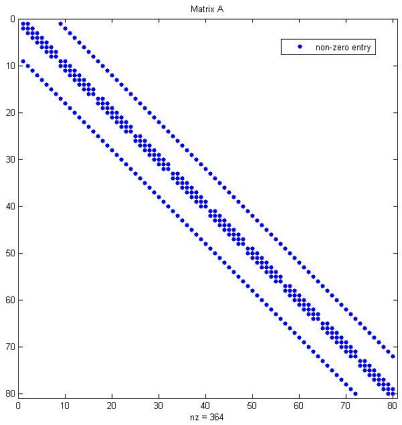


Figure 6.2: Non zero pattern of matrix A in the grid from figure 6.1

6.3.2. 2D Poisson Equation Setup

Let us recall the PDE from chapter 2 that governs the movement of steady state groundwater:

$$\nabla \cdot (K \Delta u) + W = 0 \tag{6.1}$$

For simplicity, we consider the the two dimensional (2D) Poisson problem on a square domain represented in the figure 6.1:

$$-\Delta u(x,y) = \frac{1}{h^2}, \quad (x,y) \in \Omega \equiv (0,2500) \times (0,2500), \qquad u(0,y) = u(0,y) = 1 \tag{6.2}$$

where h is the distance between the node of two neighboring cell centered cells. h turns out to be the size of the cell. In equation 6.2, we have divided the right-hand side constant vector 1 by h^2 to nullify the h^2 term which arises in the denominator of the coefficients in the Poisson matrix A after approximating the Δu with second order finite difference scheme. It ensures that the 5 point stencil for the discretized Poisson matrix A will be independent of the cell size h. The stencil is defined as

$$\begin{array}{ccc} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{array} \tag{6.3}$$

Keeping in mind the complexity of the FORTRAN code in iMOD [10] due of parallel subroutines, we have created 2D Poisson test case also in the iMOD software. However, we observe from section 2.5 that the entries in the MODFLOW matrix A comes from the conductance values CR, CC, and CV of the neighboring cells. To make the discretized matrix coefficients similar to the 2D Poisson matrix, we need to update the matrix coefficients. Approximating the second order partial derivatives in equation 6.2 with central finite difference [8] yields the five point stencil given in equation 6.3. Therefore, we overwrite all the diagonal entries by 4 and non-diagonal entries by -1 in the FORTRAN Code. We also notice from chapter 2 that, the right-hand side in the LSE depends upon various parameters such as specific storage, cell size, etc. For simplicity, we assume the right hand side vector is 1 in equation 6.2. In iMOD, it means to set the initial right-hand side vector for interior and cell adjacent to boundary cells to 1. The right-hand side vector for cells adjacent boundary cells is computed with non-homogeneous boundary condition as given in equation 6.2.

For the remaining experiments in the section 6.3, we want to see the variation of total CG iterations with increasing number of subdomains. By total CG iterations we mean, inner CG Schwarz iterations for all Picard iterations. We need a substantial computational intensive problem, so we need to make the cell size small. We fix our cell size 2.50 meter, so we have 1000 cells both in x and y direction. Since the left and right side strip in L_1 contains constant head cells, they are not counted in the LSE unknown variables. we have 998 $\times$1000 cells variables to solve for in L_1. We choose, head change termination criteria (HCLOSE) to be 10^{-6}, and residual change closing criteria (RCLOSE) to be 10^{-4}. To control the termination criteria of the PCG solver, PKS has a variable called `ICNVGOPT`. Depending on the value of

ICNVGOPT, the different termination criteria can be used such as infinity norm, L2 norm. In all our observation in this chapter, we have set ICNVGOPT to 0 that means infinity norm termination criterion for all the solvers.

6.3.3. Variation of Solver Iterations with Number of Subdomains

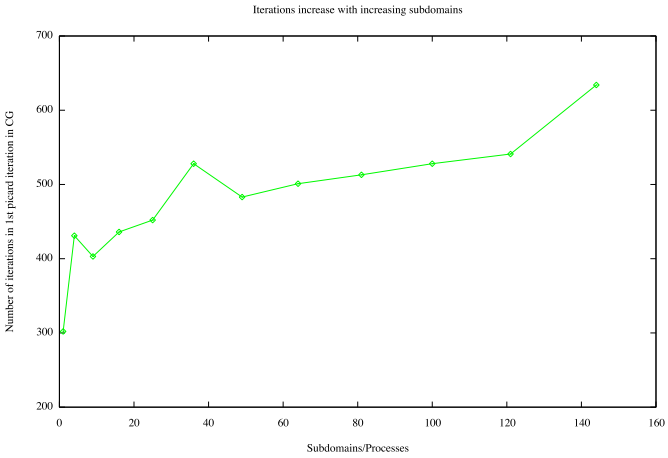


Figure 6.3: Increase of iteration in 1000 ×1000 grid cell with increasing number of subdomains in 1st picard iteration

To solve the computational intensive problem, we divide a big problem into small subproblems, which are computed in parallel and we expect that this procedure scales well, in our case, it means that when we increase the number of subdomains/processes, the number of iterations should not increase. However, we inspect the growth in the number of iterations with growing subdomains as shown in the figure 6.3. As we increase the number of subdomains, we increase the subdomain interfaces. Therefore, it takes more iterations to transfer/propagate the information.

We approximate the harmful eigenvector using constant deflation vectors as defined in section 5.2. We observe that the CG iterations do not increase with increasing subdomains as shown in figure 6.4. We then approximate the harmful eigenvector using linear deflation vectors, first only in the x-direction, and then in x and y-directions. We observe that the linear deflation scheme performs better than the constant deflation scheme.

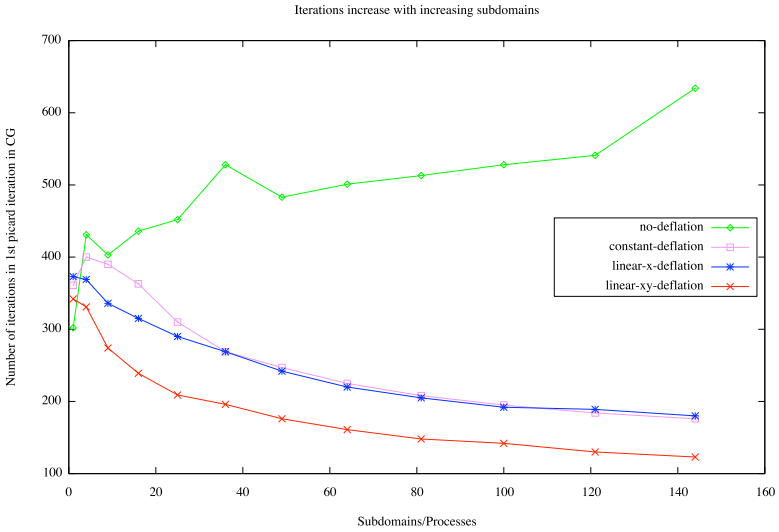
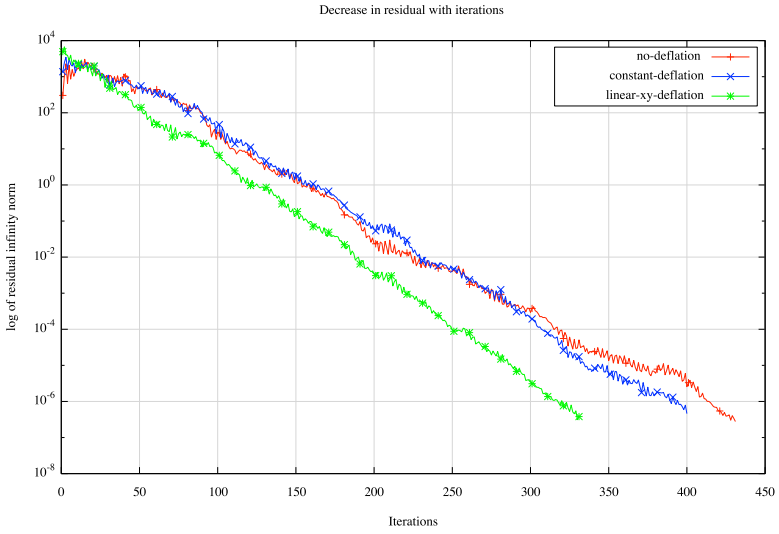


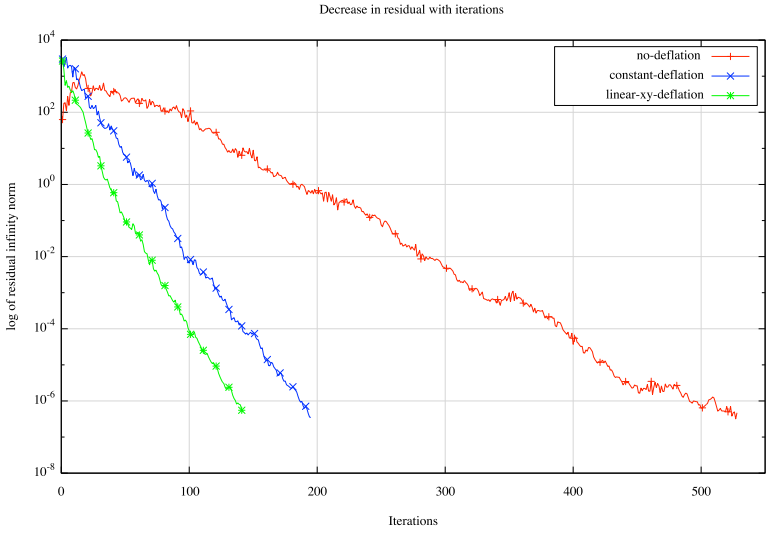
Figure 6.4: Increase of iteration in 1000 ×1000 grid cell with increasing number of subdomains in 1st picard iteration

6.3.4. Variation of Residual Norm with Global Iteration

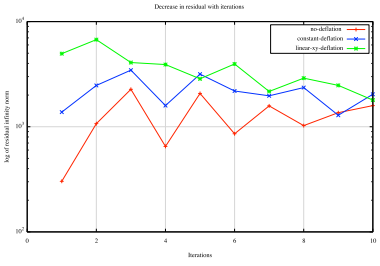
To check the convergence of any iterative method, one should look into the residual. From the figure 6.5, we see that the residual norm decreases faster in DPCG with constant deflation vectors than the original PCG method. Also within DPCG, the residual norm decreases faster for linear deflation vectors than the constant deflation vectors. We also see that the residual norm decrease is more rapid in DPCG than PCG method for the higher number of subdomains. In the initial iterations, we observe from the zoomed figures that the infinity norm of the deflated residual vectors is higher than the corresponding not deflated residual vectors. This increment is due to computation of initial residual of the deflated system in the deflation pre processing phase.



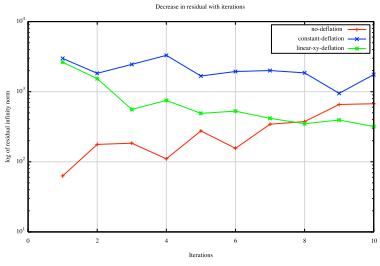
(a) 4 subdomains



(b) 100 subdomains



(c) zoom of a)



(d) zoom of b)

Figure 6.5: Decease of residual infinity norm for 2D poisson 1000 ×1000 grid cell

6.4. 2 Layer iMOD Unit Case

Now since we know the how deflation works for the synthetic Poisson PDE, we want to get insight into real life case. Since the actual models are three dimensional, we want to gain insight into the effect of deflation on 3D models.

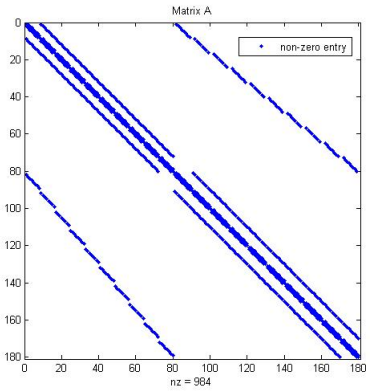


Figure 6.6: Non zero pattern of matrix A in the grid 10 ×10 ×2 representing 2 layer iMOD unit case

6.4.1. Model Description and Setup

To construct a 3D unit case in iMOD, we add one layer below the layer L_1 in the existent model described in the section 6.3.1. We denote this layer by L_2. model description about the layer L_1 is same. The extra information about this 3D unit case is given below:

- The domains of layer L_2 is same as that of layer L_1. The number of cells is also same in L_1 and L_2.
- No-flow cells are absent in the layer L_2 like layer L_1. Unlike L_1 layer, the left and right side stripe are also variable-head (active) cells. So all the cells in L_2 layer are active cells with IBOUND value 1.
- The transmissivity is 100 in layer L_2 like L_1 .
- The starting head value at cells is 1 meter.
- Boundary conditions are not specified in the layer L_1. They are only specified in the top (here L_1) layer.

We are interested in computing the hydraulic head h in the following steady state PDE 6.4 (refer to chapter 2).

$$\frac{\partial}{\partial x}\left(K_{xx}\frac{\partial h}{\partial x}\right)+\frac{\partial}{\partial y}\left(K_{yy}\frac{\partial h}{\partial y}\right)+\frac{\partial}{\partial z}\left(K_{zz}\frac{\partial h}{\partial z}\right)+W=0 \tag{6.4}$$

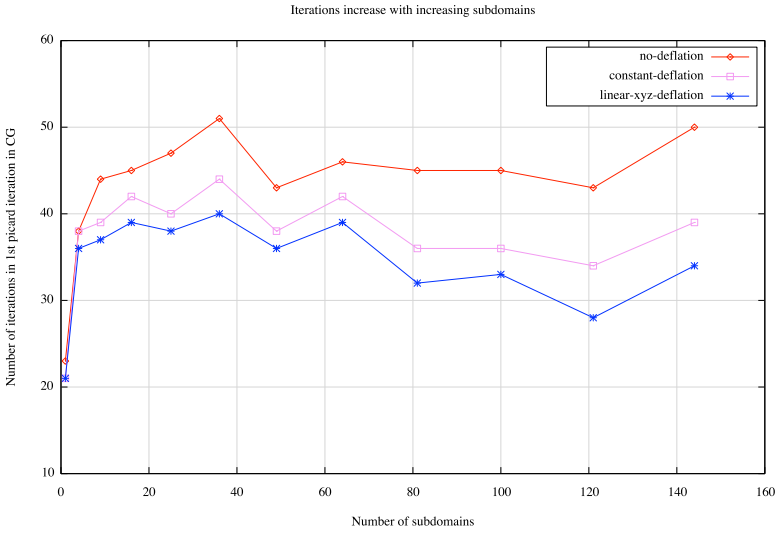
We fix the cell size to 25 meters, so we have 100 cells both in x and y direction. Since the left and right side strip in L_1 contains constant head cells, they are not counted in the LSE unknown variables. we have 98 ×100 cells variables to solve for in L_1. All the cells are active in L_2, so we have to solve for 100 ×100 cells variables. In total, we solve the LSE for 19800 unknown variables. Like Poisson 2D problem, We choose the head change termination criteria (HCLOSE) to be 10^{-6}, and residual change closing criteria (RCLOSE) to be 10^{-4}.

6.4.2. Variation of Solver Iterations with Number of Subdomains

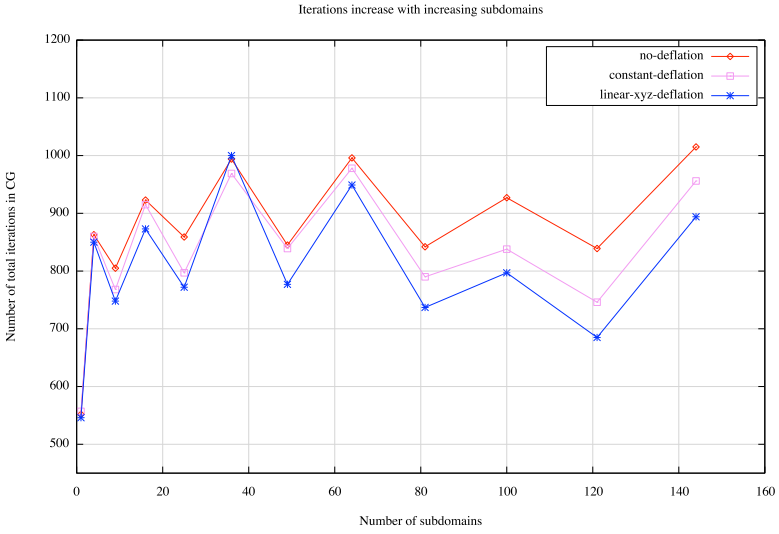
In the figure 6.7, we observe that introduction of deflation reduces the number of CG iteration in the first Picard iteration and also the total iteration (using all the Picard iterations).

In our experiments, we have used 30 inner iterations by default. However, the faster convergence is achieved in DPCG than PCG for each outer iterations. We would like to see the effect of deflation in each outer iteration. In figure 6.8 we plot the number of inner iteration in each Picard iteration. We observe that for the small number of subdomains (4 subdomains in the figure 6.8a), there is not much difference in the number of inner iterations in PCG, CDPCG and LDPCG solver. However, for the larger number of subdomains (100 subdomains in the figure 6.8c), we see that the CDPCG take less inner iteration than PCG in each Picard iteration. LDPCG even takes lesser inner iterations than the CDPCG method. In the figure 6.8b, we see the it is somewhere between the figure 6.8a and figure 6.8c as far as inner iteration decrease is concerned in the Deflated solver. From this experiment we draw two conclusions:

- Difference in inner iterations in Deflated and PCG method is higher for higher number of subdomains.
- For higher number of subdomains, the LDPCG method takes less inner iterations than the CDPCG method for initial Picard iterations.

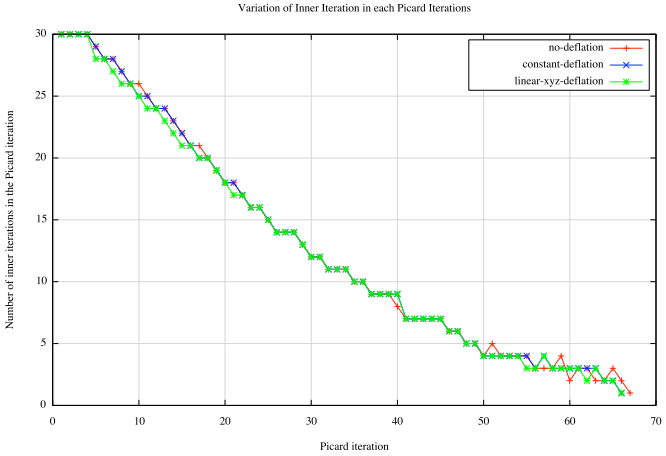


(a) first Picard iteration

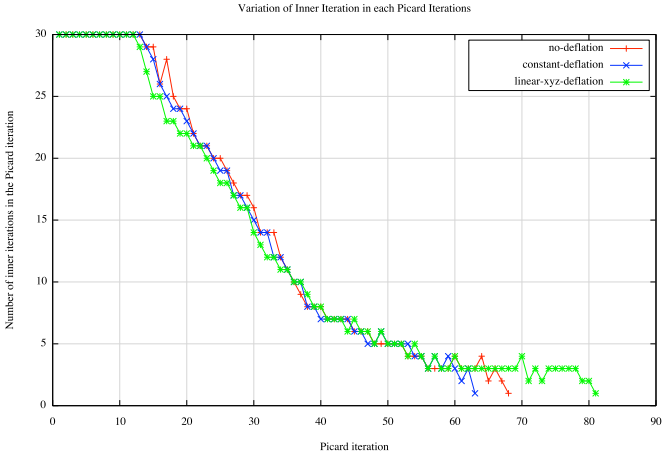


(b) Full simulation

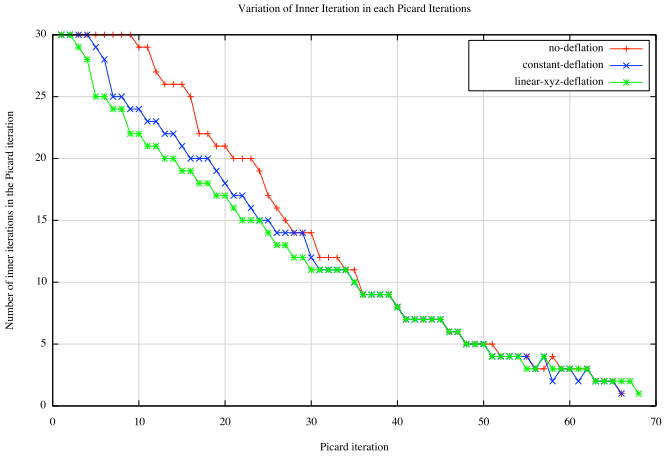
Figure 6.7: Increase of iterations in 100 ×100 ×2 model grid-cell with increasing number of subdomains



(a) 4 subdomains



(b) 36 subdomains



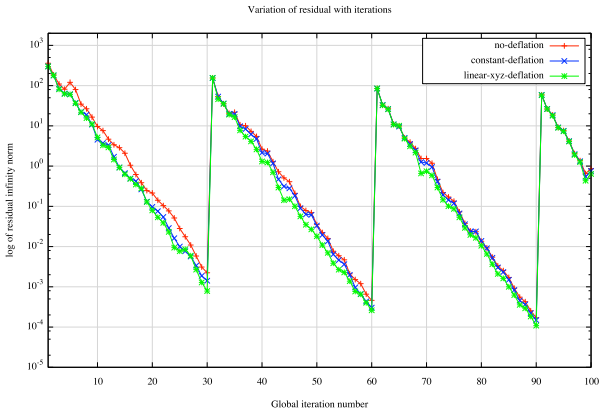
(c) 100 subdomains

Figure 6.8: Number of inner iteration required in each Picard iteration in 100 ×100 ×2 model grid-cell

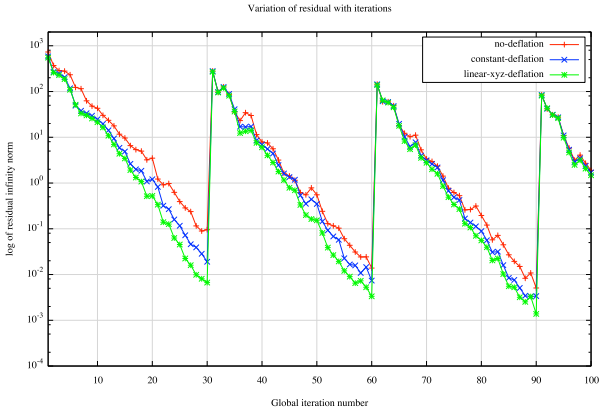
6.4.3. Variation of Residual Norm with Global Iterations

Now to check the rate of convergence of the PCG, CDPCG and LDPCG method, we plot the residual in first 100 global iterations in figure 6.9. We call the iteration number including Picard and inner CG iteration as global iteration. We observe the following:

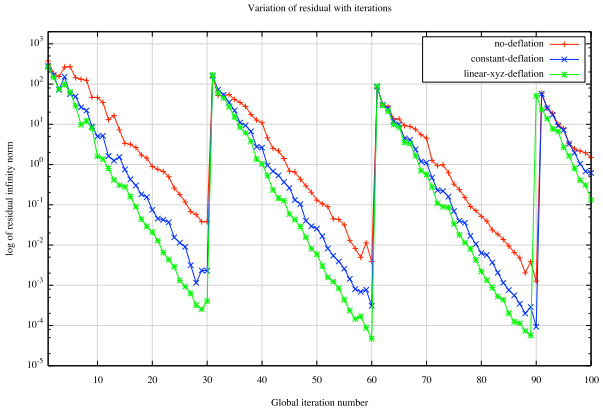
- At the end of each Picard iteration (global iteration number 30, 60 and 90), the residual norm increases in all the solvers since new residual is computed at construction of new linear system.
- The residual decrease is faster in Deflated solvers than PCG solver for higher number of subdomains.
- For higher number of subdomains, the LDPCG method converges faster than the CDPCG method in all Picard iterations.



(a) 4 subdomains



(b) 36 subdomains



(c) 100 subdomains

Figure 6.9: Variation of residual norm with global iteration in 100 ×100 ×2 model grid-cell

7

Numerical Experiments for the Real Life Models

7.1. Introduction

In this chapter, we apply constant and linear deflation on Nederlands Hydrologisch Instrumentarium (NHI) steady state model. Experiments are carried out on the Dutch National Supercomputer Cartesius [7]. The numerical experiments are done 24 core CPU Intel® Xeon® Processor E5-2690 v3 [12].

The structure of this chapter is as follows. We give NHI model description and briefly mention scalasca profiler [14] in section 7.2. We present numerical results for NHI model with various cell sizes in section 7.3, 7.4 and 7.5. We give numerical results for the California model in section 7.6. During the development of Deflation in the PKS package, we started with an inefficient implementation. However, we improved our code after taking the scalasca profiler readings. We present two such examples in section 7.8.

7.2. Setup Description

In this section, we familiarize the reader with NHI model.

7.2.1. Introduction to NHI SS Model

National and regional water authorities develop long-term plans for sustainable water use and safety under changing climate conditions in the Netherlands. Based on available data and state-of-the-art technology, the decisions about investments are supported by the Netherlands Hydrological Instrument (NHI).

NHI is the collection of software and data for the development of groundwater and surface water models for the Netherlands on a national and regional scale. The

NHI is intended to bundle knowledge of specialists at water managers, institutes, and consultants to achieve quality tools for sustainable development. The NHI models have been developed with the collaboration of various research institutes such as Deltares, Rijkswaterstaat, STOWA, PBL, and Alterra. The national applications of NHI include Landelijk Hydrologisch Model (LHM) [13]. Different software component simulates various hydrological regions such as MODFLOW for saturated ground water, METSWAP for the unsaturated zone, etc., as shown in the figure 7.1. We have considered only groundwater simulation in the MODFLOW code. Also, our research focuses only on the solver (Picard and CG-Schwarz) since we have used one stress loop and one time-step loop (SS Model).

For the saturated zone, the NHI model contains 7 layers representing hydrological information about different layers such as sand and clay layer. For example, representation of the various layers beneath the subsurface between Amsterdam and Utrecht can be found in the figure 7.2. The data has been taken from the publishing portal of TNO called DINOloket Geological Service Netherlands [4]. The hydrological data is publicly available at DINOloket for free.

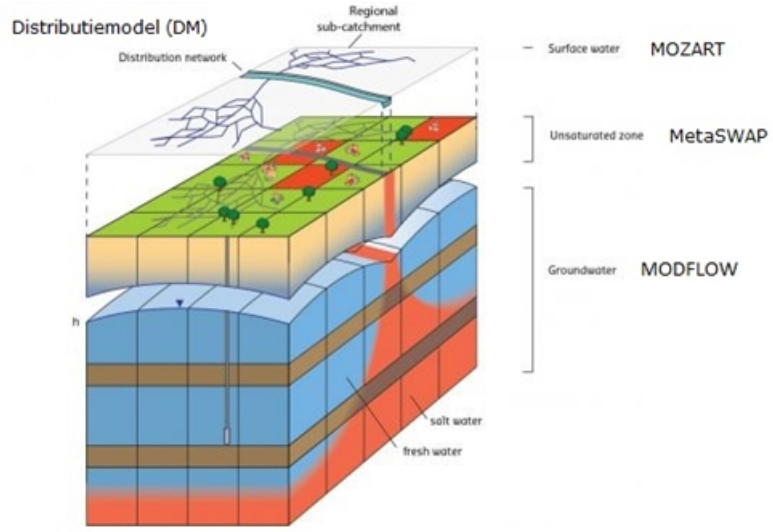


Figure 7.1: Different models present in the LHM

7.2.2. Model Description

The description of the model parameters is given below [33]:

- The domain of the model (the rectangle in figure 7.3) represents an area of size 300 km ×325 km with the coordinates x_{min} = 0 m, x_{max}=300000 m, y_{min} = 300000 m and y_{max} is 625000 m. Here x_{min} denotes starting coordinate in x

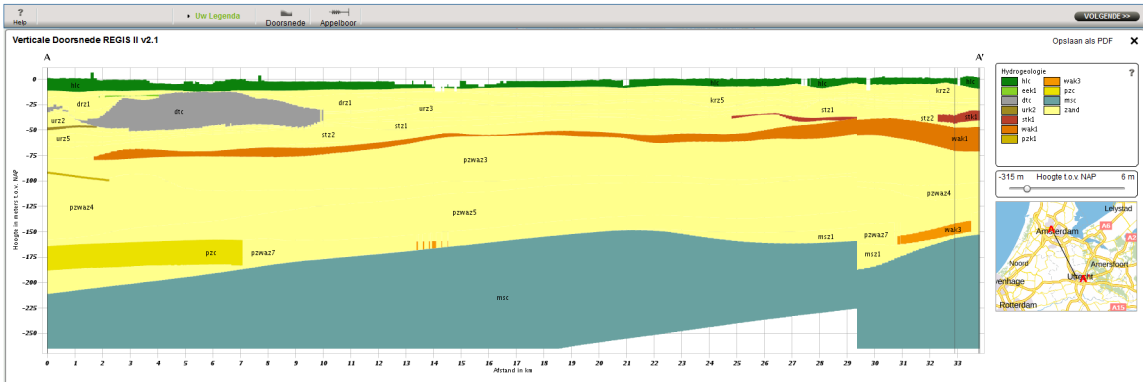


Figure 7.2: Distribution of 7 layers underneath the ground between Amsterdam and Utrecht

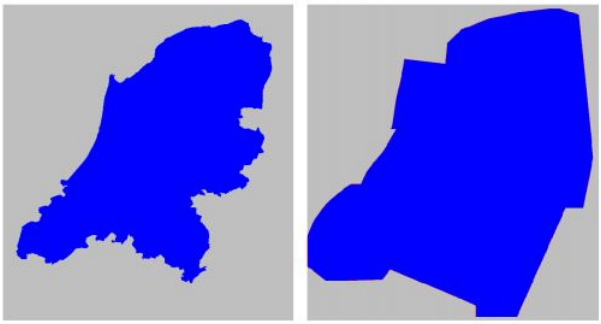


Figure 7.3: Domain of top layer L_1 (left) and bottom layers L_2 to L_7 (right)

direction and x_{max} denotes ending coordinate in x-direction, similar notation for y-direction as well. We can vary size of the cell. The cell size in the current NHI model is 250 meter. It leads to 1200 cells in x-direction and 1300 cells in y-direction. Thus the total number of cells (active and inactive) are 10.92 million ($1200 \times 1300 \times 7$).

- The model contains 7 layer denoted as L_1, $L_2 \ldots L_7$ in order. The topmost layer is present on earth's surface, denoted by L_1, and the bottommost layer is denoted by L_7. The cells shown in the gray background in the figure 7.3 are inactive cells (no flow cells) with IBOUND 0. The cells shown in the blue are active cells (variable-head cells) with IBOUND 1. The number of active cells in layer L_1 is lesser than the number of active cells in layer L_2. Number of active cells in all the layers beneath L_1 is same and equal to the number of active cells in L_2. The total number of active cells are $6,292,108$. Therefore, 57.62 % of the cells active in this 7 layer NHI model.
- number of drains: 960,586, Number of rivers: 1,340,376, Number of wells: 62,752 .

- General head boundaries: 176,243, Anisotropy cells (Deltares package): 87,291, Horizontal flow barriers: 3623.
- Old starting heads are used as initial head values in the first Picard iteration. In the consecutive Picard iteration, the solution obtained from the previous Picard iteration is used.
- We choose, head change termination criteria (HCLOSE) to be 10^{-4}, and residual change closing criteria (RCLOSE) to be 10^{-1}.
- In all our observation in this chapter, we have set the flag ICNVGOPT to 0 that means infinity norm termination criterion has been selected for all the solvers.

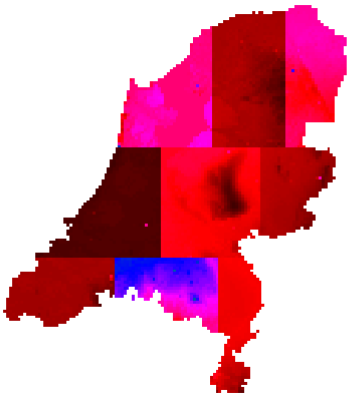


Figure 7.4: Domain decomposition consisting of 3×3 subdomains using Recursive Coordinate Bisection partitioning

7.2.3. Scalasca Profiling

Scalasca is a software tool that supports the performance optimization of parallel programs by measuring and analyzing their runtime behavior. The analysis identifies potential performance bottlenecks and offers guidance in exploring their causes. Thus, we can improve the performance of the parallel codes by addressing to the bottlenecks reported by scalasca profiler.

To profile the code using scalasca, we append `scalasca -analyze` before the string in the line, which contains codes to run the executable. It generates a folder starting with name `scorep`. This folder contains a file with extension `cubex`. The cubex file contains information of various subroutines such as time taken, MPI communication etc. We use this information to optimize our parallel code. To launch the Graphical User Interface (GUI) of scalasca, one needs to use scalasca keyword

`square` before the folder starting with scorep. Scalasca has been used to decrease the wall clock time, during the development of CDPCG and LDPCG method.

7.3. 250m NHI SS model

To support the decision makers in solving hydrological problems, detailed high-resolution models are often needed. These models typically consist of a large number of computational cells. Such large models have very long computational hours when solved on a serial computer. For instance, at the start of the development of NHI models in 2006, the serial run took about 20 days to run all the models (figure 7.1) in LHM for a thirty-year duration. The Parallel Krylov Solver (PKS) package was developed to reduce the run time of the simulation. Preliminary results with PKS show that increasing the number of subdomains results in increasing the number of CG iterations. Maximum 18% increase in the number of CG iterations is observed until 120 subdomains [26] in the current implementation with cell-size 250 meters.

Currently, other groundwater models of Deltares, such as IBRAHYM for the province of Limburg, have a cell-size of 25 meter for modeling at a regional scale. To obtain consistency in the data generated by regional and national (NHI) groundwater models, Deltares has the ambition to increase the horizontal resolution to 25 m. The current approaches include upscaling in which, the information known at regional model can be used to make predictions about national scale (NHI) models. However, it is not so straightforward to obtain. It would be best for Deltares if the performance of the PKS package is improved so that computational time can be reduced to achieve 25 meter resolution for NHI model. For coupled NHI models (MODFLOW + METASWAP), load imbalance across subdomains is one of the important issues.

Subdomain Index	$\|u\_PCG - u\_CDPCG\|_\infty$	$\|u\_PCG - u\_LDPCG\|_\infty$
0	7.16E-007	1.30E-006
1	1.89E-006	1.35E-005
2	3.04E-005	5.35E-006
3	2.89E-006	4.57E-006
4	9.52E-006	2.06E-005
5	8.29E-006	2.23E-005
6	5.96E-006	1.50E-005
7	3.95E-004	9.53E-004
8	1.66E-005	2.23E-005

Table 7.1: head difference in 3×3 subdomain given in figure 7.4

The increase of total PCG solver iterations with increasing number of subdomains is one of the bottlenecks in the current implementation in the PKS package. The growth in a number of subdomains reduces the global coupling in the entire domain. Deflation can achieve the global coupling. In our first experiments with the

NHI 250 SS model, we observe that the number of iterations does not increase with the increasing number of subdomains after applying coarse space correction deflation (refer figure 7.5). As mentioned in the model description, we use HCLOSE value 10^{-4}, RLOSE value 10^{-1} and the 30 inner iterations (ITER1) per outer iteration. We compared the head values in the original and deflated solution using infinity(max) norm in each subdomain decomposition. For 3×3 decomposition corresponding to the figure 7.4, the difference in head values (in meter) in each subdomain is given in table 7.1. We denote the computed solution from PCG solver by u_PCG. We denote the solution obtained with constant deflation vectors by u_CDPCG whereas the solution obtained with constant and x, y, z linear deflation vectors by u_LDPCG.

Correctness in CDPCG and LDPCG: To check the correctness of the Deflated algorithms, we present the absolute maximum of the head difference between PCG and Deflated PCG across all the subdomains in table 7.2. We notice that an accuracy of order E-003, i.e., 10^{-3} is obtained.

#Subdomains	$\|u\_PCG - u\_CDPCG\|_\infty$	$\|u\_PCG - u\_LDPCG\|_\infty$
1	1.46E-003	2.53E-003
4	3.07E-004	6.19E-004
9	3.95E-004	9.53E-004
16	8.04E-004	1.91E-003
25	3.97E-004	6.13E-004
36	2.93E-004	1.91E-003
49	1.52E-004	7.42E-004
64	1.06E-003	2.72E-003
81	1.75E-003	2.48E-003
100	2.81E-003	1.95E-003
121	4.48E-003	1.88E-003
144	5.03E-004	3.09E-003

Table 7.2: Difference between the head values in PCG and Deflated PCG solvers

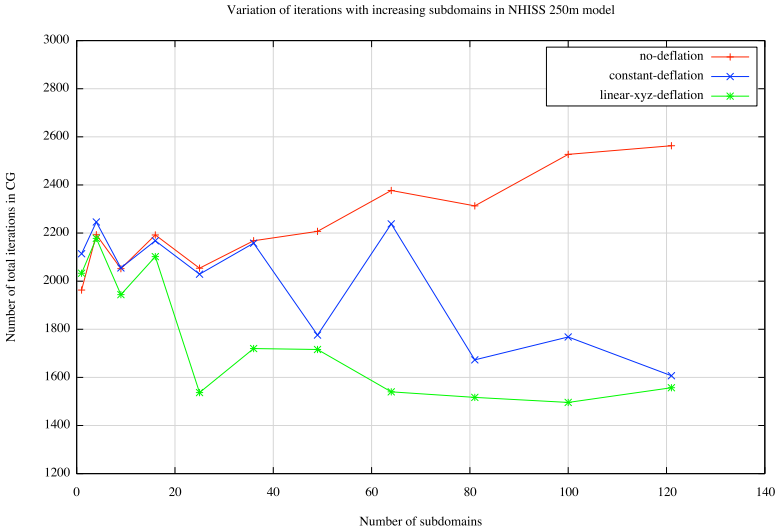
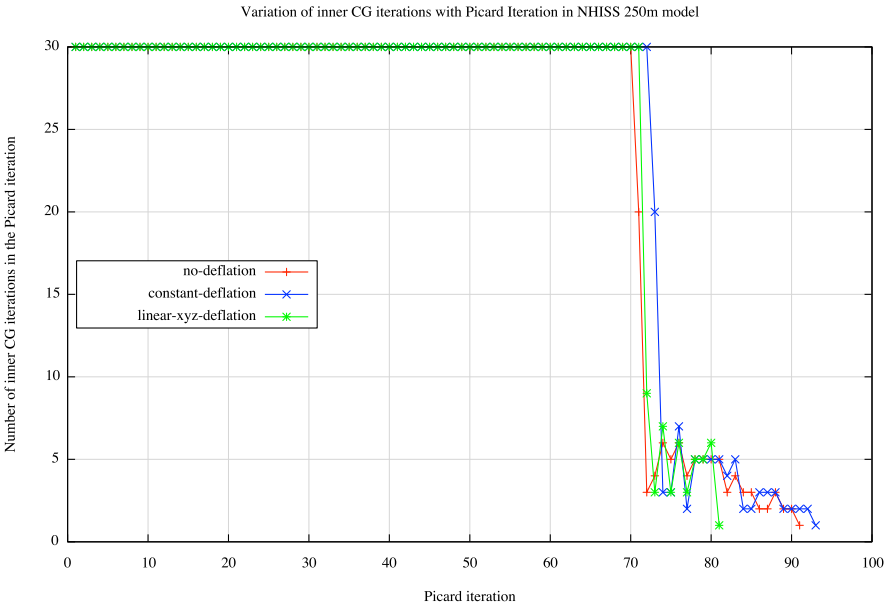


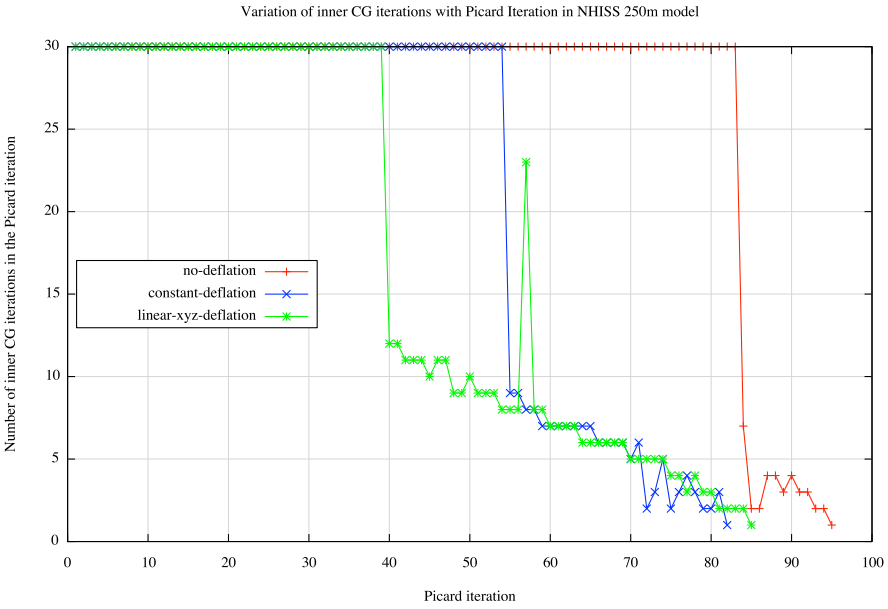
Figure 7.5: Variation of total iterations with increasing number of subdomains in 1200 ×1300 ×7 grid cell in 250m NHI SS model

The number of total iterations includes outer Picard iterations and inner CG iterations. One linear system of equations is created in each Picard iteration. Also, the Deflation method is used in each Picard iteration. We want to gain insight into the number of inner iteration used in each outer iteration in the PCG and DPCG method. From figure 7.5, we also observe that the iteration difference between PCG and DPCG increases with increasing number of subdomains. We investigate the effect on one lower (4) and one higher (100) number of subdomains.

In figure 7.6a, we notice that until about 70 Picard iterations, both PCG and DPCG method takes 30 number of inner iterations. In the remaining Picard iterations, also both PCG and DPCG takes about the same number of inner iterations. Since the number of iteration is small, the global coupling (global information) is not completely lost in 4 subdomains. Therefore, both the PCG and DPCG algorithm takes about the same number of total iterations. However, we observe in 7.6b that after 38^{th} Picard iteration, the DPCG method with linear deflation vectors takes fewer inner iterations than the PCG method. After 54^{th} Picard iteration, the DPCG method with constant deflation vector also takes fewer inner iterations than the PCG method. To understand the number of iterations, we need to look into the residual in PCG and DPCG method.

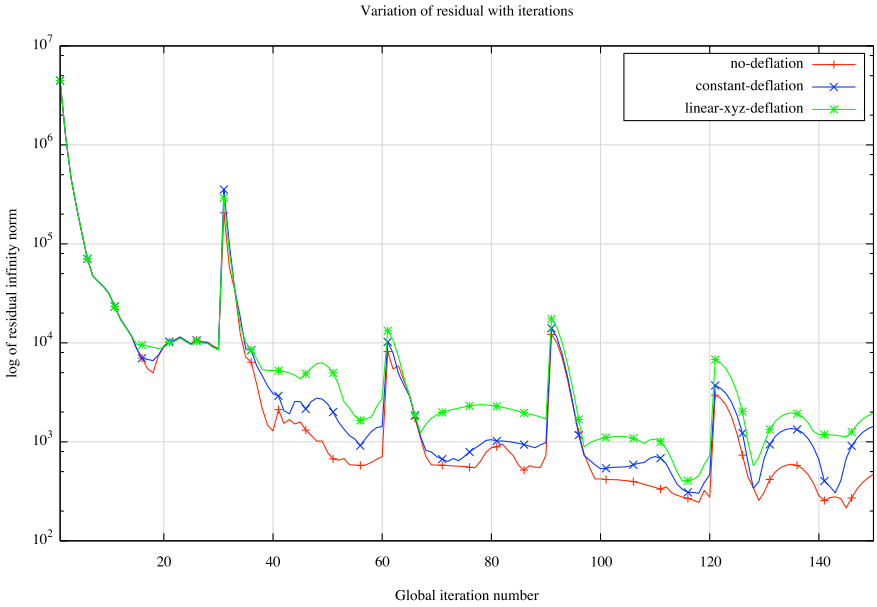


(a) 4 subdomains

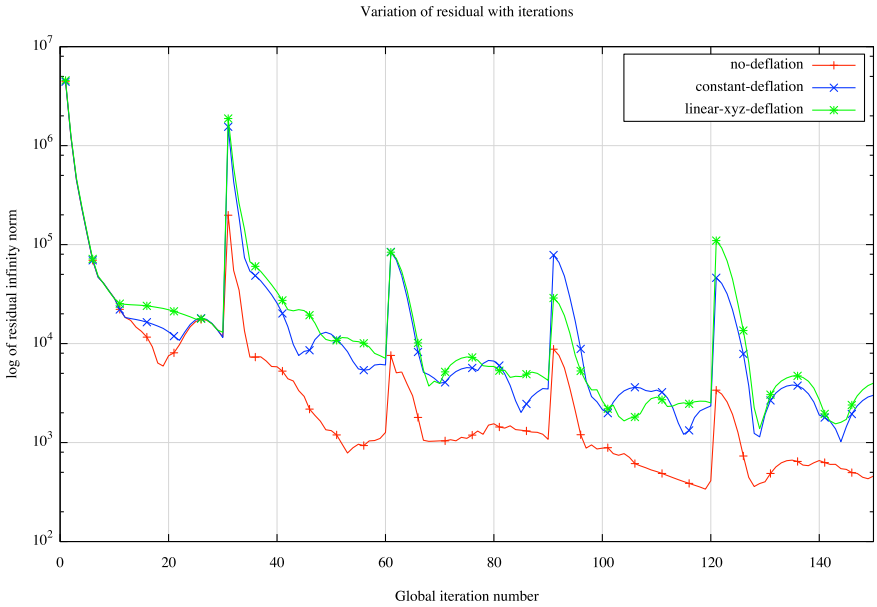


(b) 100 subdomains

Figure 7.6: Number of inner iteration required in each outer (Picard) iteration 1200 in ×1300 ×7 grid cell in 250m NHI SS model

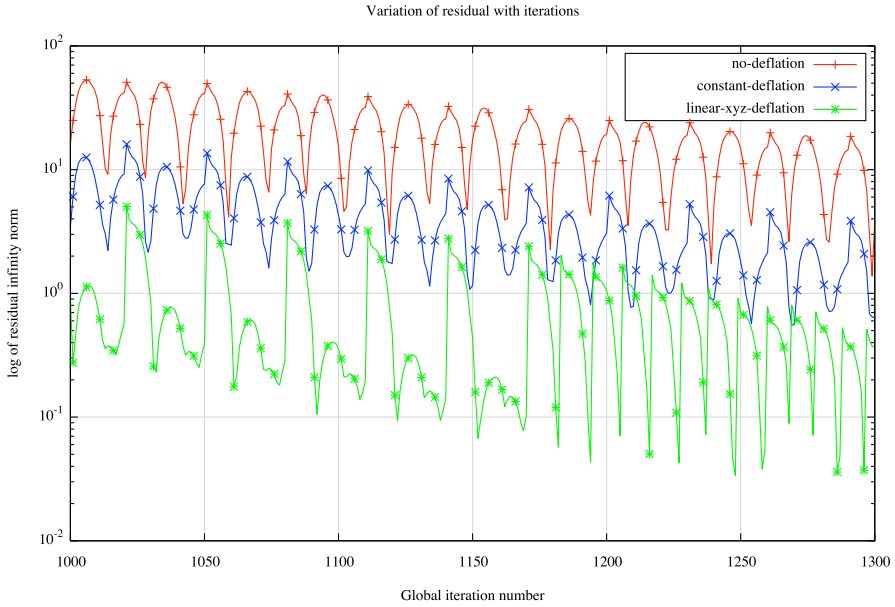


(a) 4 subdomains

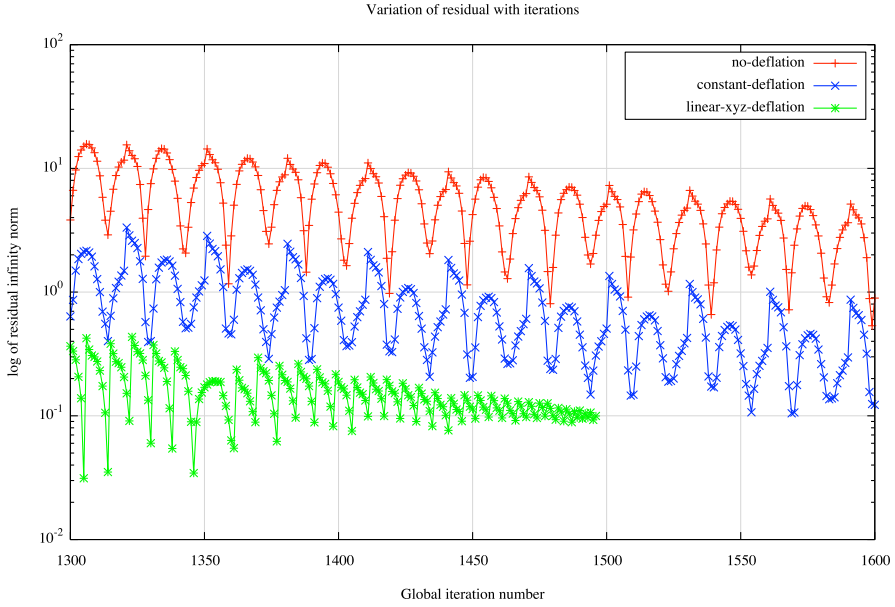


(b) 100 subdomains

Figure 7.7: Variation of Infinity residual norm with initial 5 Picard iterations in 1200 ×1300 ×7 grid cell in 250m NHI SS model



(a) global iteration number 1000- 1300



(b) global iteration number 1300- 1600

Figure 7.8: Variation of residual norm with global iteration number using 100 subdomains in 1200 ×1300 ×7 grid cell in 250m NHI SS model

Each MPI process (subdomain) computes its local residual using infinity norm. Then the global infinity norm is calculated by communicating local residual norm across all the process and selecting the maximum of them. To check the residual norm for initial iterations, we plot the residual norm for initial 5 Picard iterations in 4 and 100 subdomains in the figure 7.7a and 7.7b respectively. We observe that the residual decreases for first 30 iterations (first Picard iteration) in PCG and DPCG method. However, since the RCLOSE termination criteria are not met, so the linear system of equations are again constructed and solved in the second Picard iteration. The initial residual at 1st inner iteration in 2nd Picard iteration (global iteration number 31) is computed using the solution computed at the end of 1st Picard iteration and matrix formed at the beginning of the second Picard iteration. Therefore, we observe an increase in the residual norm at 31st global iteration. A similar increase in residual is seen at the beginning of 3rd, 4th and 5th Picard iteration (global iteration: 61, 91 and 121 respectively. We also observe a behavior that the residual norm in the DPCG method (both constant and linear deflation) is higher than the PCG method. This effect goes away after few initial picard iterations. We do not completely understand it yet but we expect that it is due to the deflation set up.

To get an overview of the convergence of the NHI SS simulation, we plot the residual norm for higher global iterations in figure 7.8. From figure 7.8a, we observe that the residual norm decreases faster in the DPCG method than the PCG method. Furthermore, we see in the illustration 7.8b that the LDPCG method has converged at the global iteration 1496.

7.4. 100m NHI SS Model

In this section we present plots of iteration and residual similar to 250 m case. Behavior of the results are similar to 250 m case so we do not explain them in details. Improvement in the iterations can be found in appendix. The wall clock improvement is tabulated below.

subdomains	PCG	CDPCG	LDPCG
1	5 HH, 9 MM, 3 SS	5 HH, 38 MM, 10 SS	8 HH, 26 MM, 37 SS
4	2 HH, 20 MM, 13 SS	2 HH, 48 MM, 17 SS	2 HH, 3 MM, 19 SS
16	53 MM, 2.015 SS	1 HH, 2 MM, 35 SS	2 HH, 16 MM, 52 SS
36	24 MM, 8.431 SS	24 MM, 11.912 SS	32 MM, 26.582 SS
64	13 MM, 57.191 SS	21 MM, 32.596 SS	10 MM, 23.553 SS
100	21 MM, 15.975 SS	7 MM, 26.627 SS	5 MM, 15.601 SS
144	6 MM, 39.927 SS	4 MM, 42.409 SS	3 MM, 59.860 SS
196	5 MM, 19.766 SS	3 MM, 56.918 SS	3 MM, 34.621 SS

Table 7.3: Wall clock time in 3000 ×3250 ×7 grid cell in 100m NHI SS model

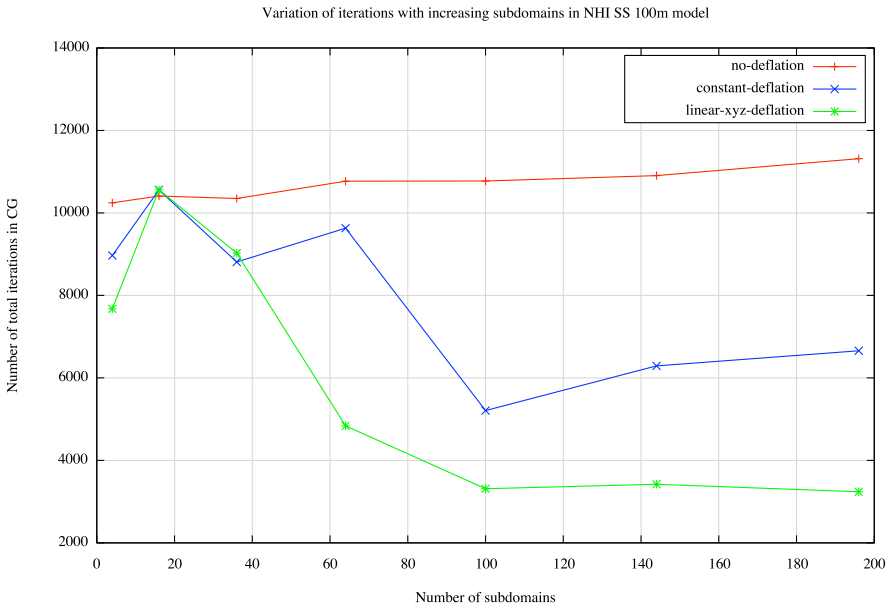
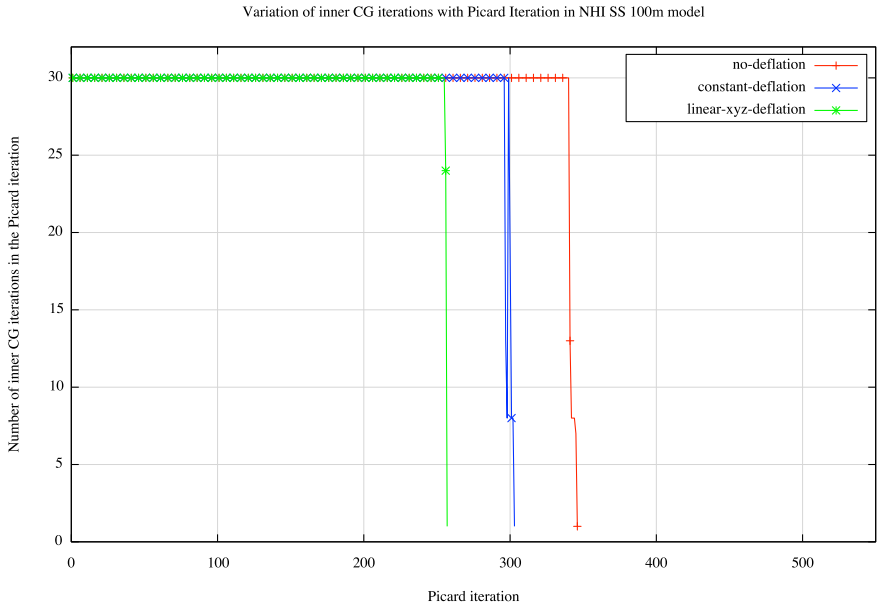
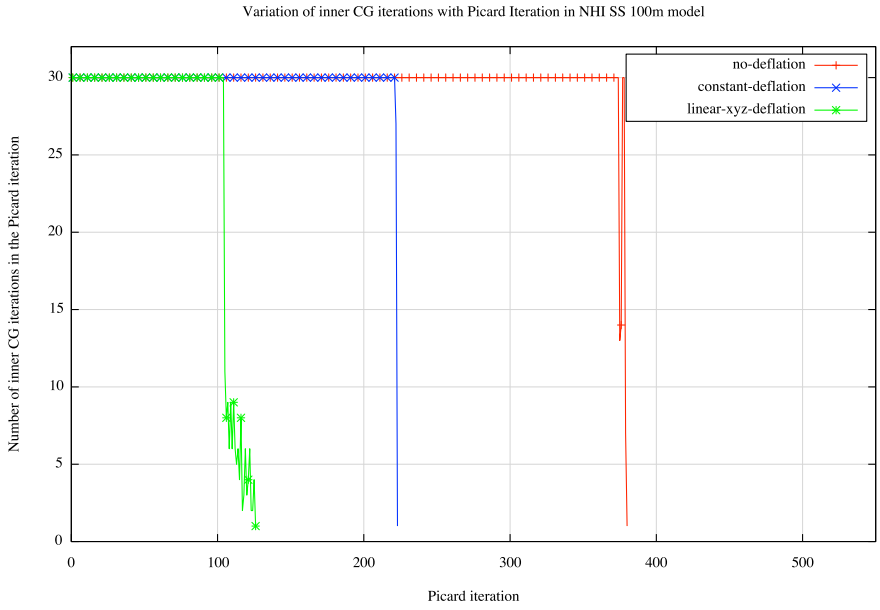


Figure 7.9: Variation of total iterations with increasing number of subdomains in 3000 ×3250 ×7 grid cell in 100m NHI SS model

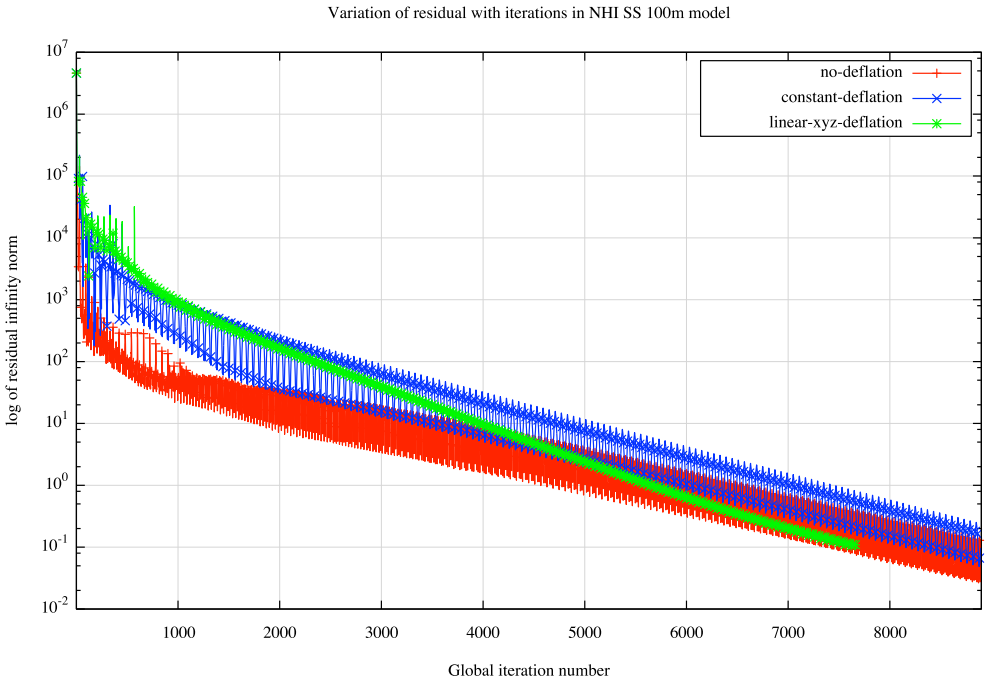


(a) 4 subdomains

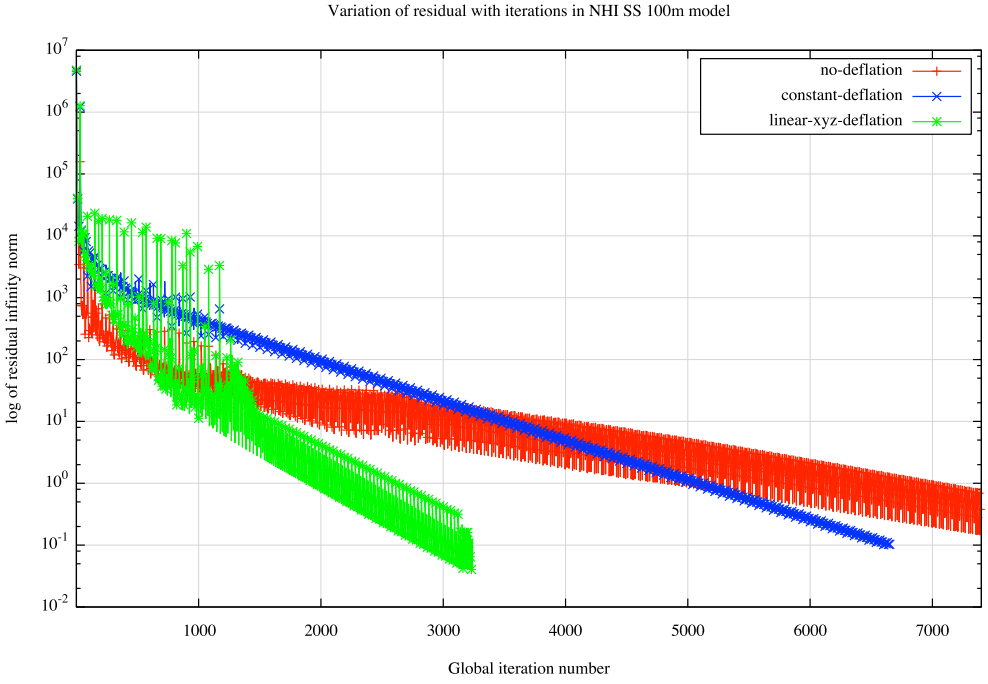


(b) 196 subdomains

Figure 7.10: Number of inner iteration required in each outer (Picard) iteration in 3000 ×3250 ×7 grid-cell in 100m NHI SS model



(a) 4 subdomains



(b) 196 subdomains

Figure 7.11: Variation of residual norm with global iteration number in 3000 ×3250 ×7 grid-cell in 100m NHI SS model

7.5. 50m NHI SS model

In this model, the cell size is 50 meter, so we solve the system of equations for $6000 \times 6500 \times 7$ grid. Since the model is bigger than previously discussed models, we take 50 inner iteration in this model. We tabulate the total number of iterations in the appendix. Wall clock time has been tabulated below.

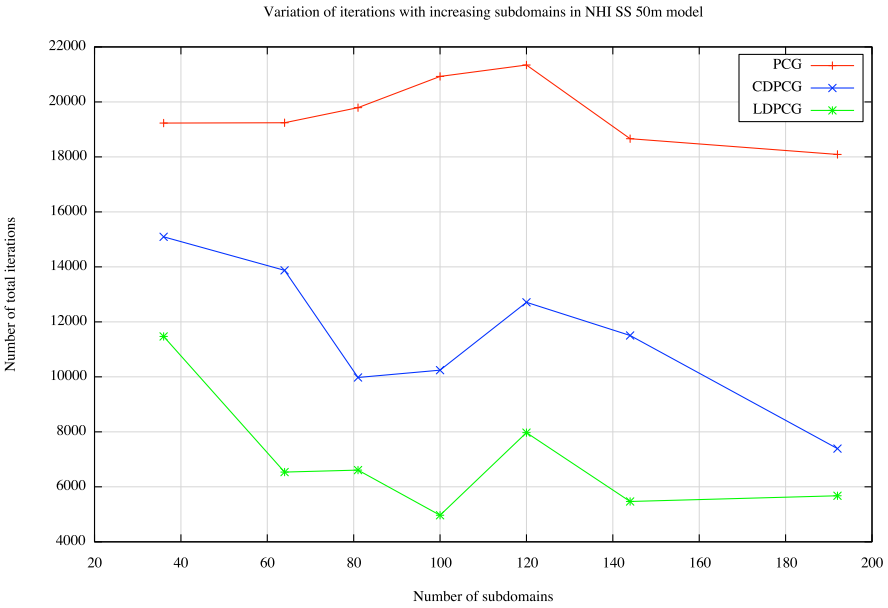
#subdomains	PCG	CDPCG	LDPCG
36	4 HH, 25 MM	3 HH, 8 MM, 11 SS	5 HH, 29 MM, 35 SS
64	2 HH, 6 MM, 27 SS	1 HH, 19 MM, 51 SS	1 HH, 26 MM, 12 SS
81	2 HH, 32 MM, 52 SS	1 HH, 28 MM, 53 SS	56 MM, 14 SS
100	1 HH, 15 MM, 32 SS	1 HH, 32 MM, 25 SS	27 MM, 53.054 SS
120	1 HH, 2 MM, 50 SS	40 MM, 55 SS	35 MM, 15 SS
144	1 HH, 36 MM, 60 SS	45 MM, 4.024 SS	48 MM, 0.842 SS
192	31 MM, 34.777 SS	16 MM, 7.648 SS	23 MM, 7.590 SS

Table 7.4: Variation of wall clock with subdomains in 6000 ×6500 ×7 grid cell in 50m NHI SS model

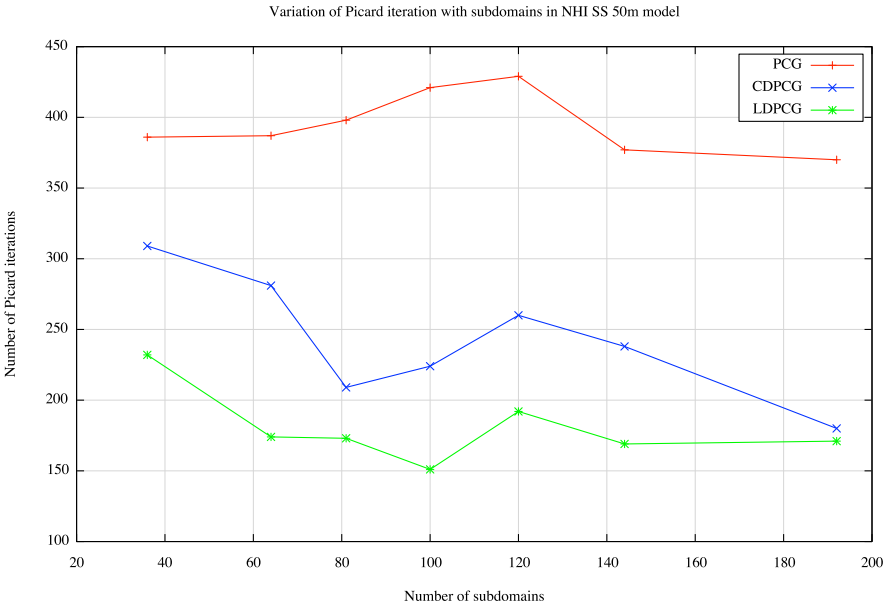
In figure 7.12a and 7.12a we observe that the iteration decreases with increasing number of subdomains in DPCG solvers. For 100 subdomains we observe a speed of about factor 4. It is the best iteration decrease that we have seen until now in all the coarser runs. From the current trends of using different cell sizes, we observe that that Deflation works better for small cell sizes. Therefore, we expect an iteration decrease by more than factor of 4 for NHI SS 25 meter model.

In figure 7.13a and 7.13b, we observe reduction in inner iteration in each Picard iteration. Like other coarser runs, we observe that Deflation wins a lot for higher number of subdomains (here 100).

In figure 7.14a, we observe that the residual norm in 2^{nd} Picard iteration is lower in PCG compared to DPCG. However, the effect seems to go away for higher Picard iterations. The figure 7.14b shows a plot of the decrease of global residual. We can see that the convergence in CDPCG and LDPCG is faster compared to the PCG method.

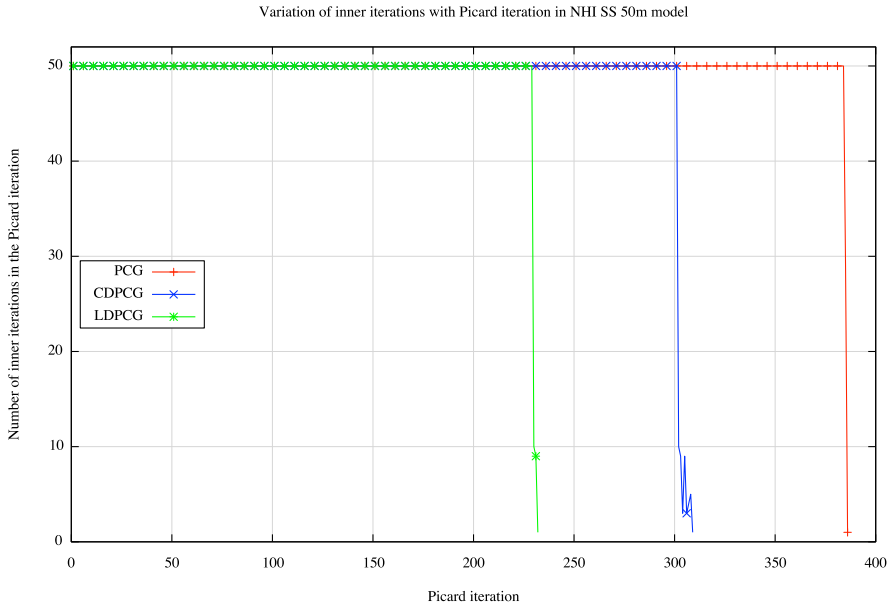


(a) Total iterations

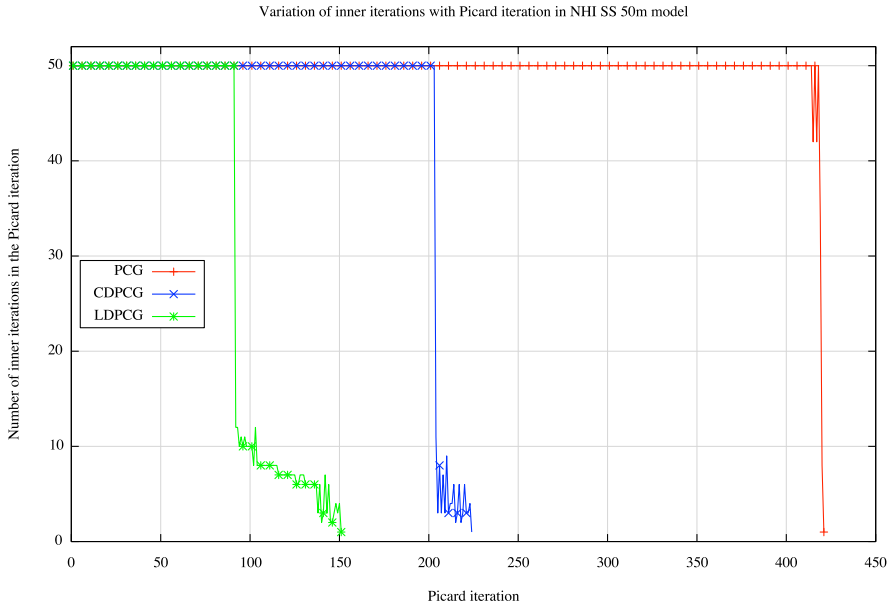


(b) Picard iterations

Figure 7.12: Variation of iterations with subdomains in 6000 ×6500 ×7 grid cell in 50m NHI SS model

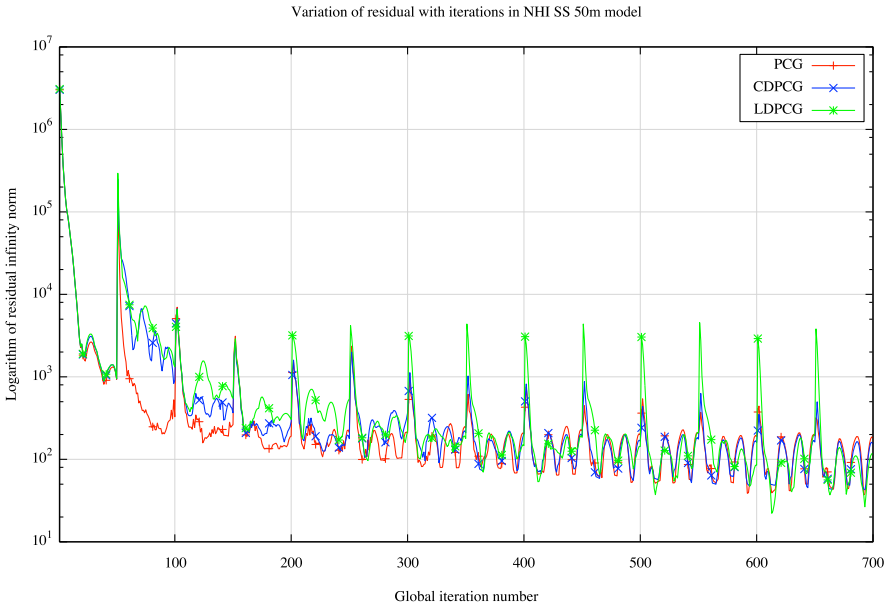


(a) 36 subdomains

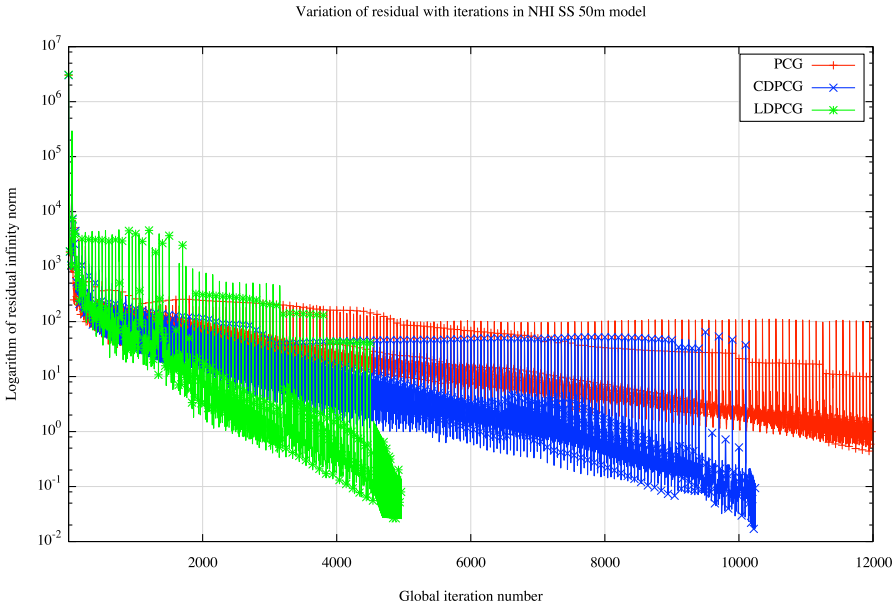


(b) 100 subdomains

Figure 7.13: Variation of inner iterations with Picard iteration in 6000 ×6500 ×7 grid cell in 50m NHI SS model



(a) global iteration: $0 - 700$

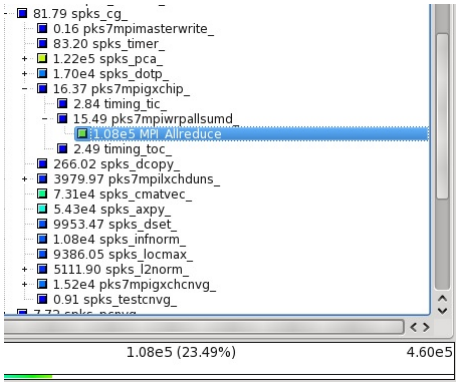


(b) Full simulation

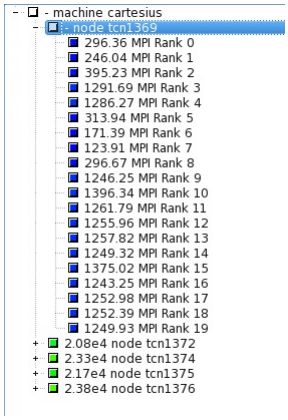
Figure 7.14: Variation of logarithm of residual norm with global iteration in 6000 ×6500 ×7 grid cell in 50m NHI SS model

7.5.1. Scalasca Profiling

To understand the wall clock time of PCG, CDPCG and LDPCG solvers, we need to look into time taken by all the subroutines. Here, we present the time measurements in the solvers for 100 subdomain.



(a) Time taken in MPI_Allreduce



(b) Communication distribution across MPI processes on one node in the MPI_Allreduce subroutine given in a)

Figure 7.15: Scalasca snapshot illustrating load imbalance in 6000 ×6500 ×7 grid cell in 50m NHI SS model in PCG method

The RCB partitioning method provides load balance by ensuring that almost equal number of active (interior) cells are assigned to all the MPI processes. However, each MPI process also includes the active cells from its ghost layer. The number of ghost layer active cells are different for each process due to irregular

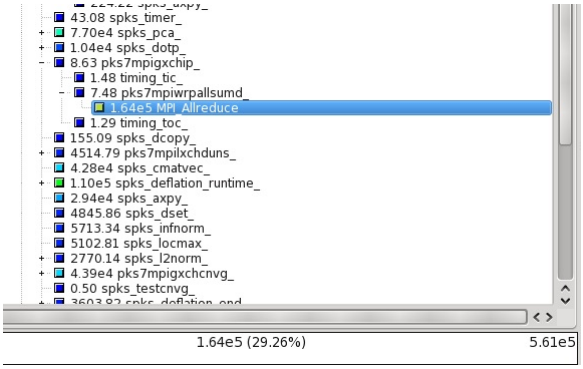
boundary of the Netherlands. Therefore, the total number of active cells (including ghost layer cells) differs across each subdomain. It creates a load imbalance. The fraction of ghost layer active cells increases for higher number of subdomains. Hence, the load imbalance is higher for higher number of subdomains. Our initial inspection for 50 m NHI model shows that 23.49% time of the entire simulation is taken in computing the global interior products of vectors using MPI_Allreduce (refer figure 7.15a. We show in the figure 7.15b that, the MPI_Allreduce communication time varies significantly across all the process on a cartesius node: from 171.39 seconds to 1396.34 seconds.

In the CDPCG and LDPCG method, construction of matrix E and vector $Z^T r^{(0)}$, $Z^T r^{(0)}$ (refer DPCG algorithm in chapter 5 for notations) introduces additional wait across the MPI processes. Therefore, we expect that load imbalance introduces overhead in wall clock time. For 100 subdomains, we notice that, the number of iterations reduces by about a factor of two in CDPCG compared to PCG method. However, the wall clock time increases from 75 minutes to 92 minutes. To investigate this behavior, we show the scalasca snapshot in figure 7.16a. We observe that MPI_Allreduce wall clock time fraction has increased upto 29.26% compared to 23.49% from PCG. Furthermore, we see from figure 7.16b that, the MPI_Allreduce communication time has become worse across all the process on a cartesius node: starting from 7.45 seconds (MPI Rank 4) to 1928.37 seconds (MPI Rank 15) .

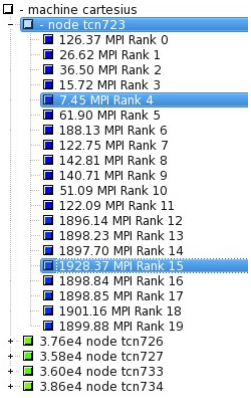
Subroutine	PCG	CDPCG	LDPCG
Perform one iteration	99.97%	97.5%	97.58%
Deflation init	NA	5.67%	14.34%
Deflation runtime	NA	19.51%	23.35%
Preconditioner	26.48%	13.71%	11.84%
Global dot product	23.5%	29.26%	23.85%
Matrix vector multiplication	15.88%	7.63%	7.33%
Vector update ($ax + by$)	11.79%	5.24%	3.72%

Table 7.5: Fraction of run times of most time consuming subroutines in $6000 \times 6500 \times 7$ grid cell in 50m NHI SS model

In table 7.4, we present % run times of most time consuming subroutines. We note that the global dot product (load imbalance) fraction has increased in CDPCG compared to the PCG. However, it remains same in the LDPCG solver since the load imbalance has moved in the Deflation init subroutine. 8.78% out of 14.34% Deflation init time is due in MPI wait (load imbalance).



(a) Time taken in MPI_Allreduce



(b) Worse communication distribution in CD-PCG compared to PCG

Figure 7.16: Scalasca snapshot illustrating 6000 ×6500 ×7 grid cell in 50m NHI SS model in CDPCG method. It is even worse than PCG method.

7.6. Miamore California Model

The description of the model parameters is given below:

- The models parameters have been calculated using Digital elevation model [3].
- The domain of the model (the rectangle in figure) represents an area of size 1227.2 km $\times 1398.7$ km with the coordinates $x_{\text{min}} = -0.1388220E + 08$ m, $x_{\text{max}} = -0.1265500E + 08$ m, $y_{\text{min}} = 3794800$ m and y_{max} is 5193500 m. Here x_{min} denotes starting coordinate in x direction and x_{max} denotes ending coordinate in x-direction, similar notation for y-direction as well. We choose 50 and 100 meter as cell size.
- For 100 meter cell size, the dimension of model grid is $12272 \times 13987 \times 1$, 171.64 million cells (active and inactive) . For 50 meter cell size, the dimension of model grid is $24544 \times 27974 \times 1$, 686.59 million cells (active and inactive).
- The model contains one layer denoted as L_1. The active modules in the model are river and recharge.
- We choose, head change termination criteria (HCLOSE) to be 10^{-3}, and residual change closing criteria (RCLOSE) to be 10^{1}.
- We use maximum 50 inner iteration in each Picard iteration.
- we have set the flag ICNVGOPT to 0 that means infinity norm termination criterion has been selected.

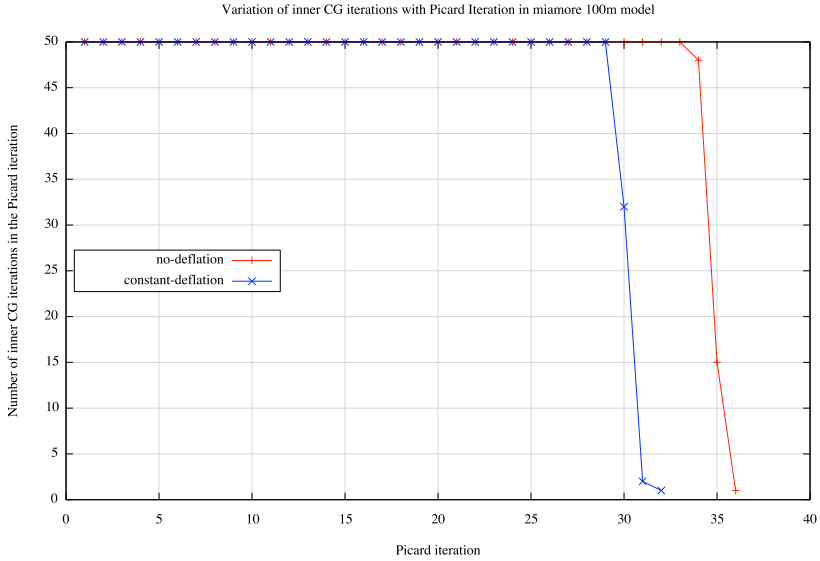
Since the domain of this model is quite large, The RCB partitioning requires a lot of memory. Therefore, we ran the jobs for 50m cell size on fat nodes. Our job did not complete successfully for 50m cell-size in for linear deflation (x and y directions).

	No deflation	Constant deflation	Linear-xy-deflation
Total iterations	1714	1485	1383
Time to run (secs)	133	132	144

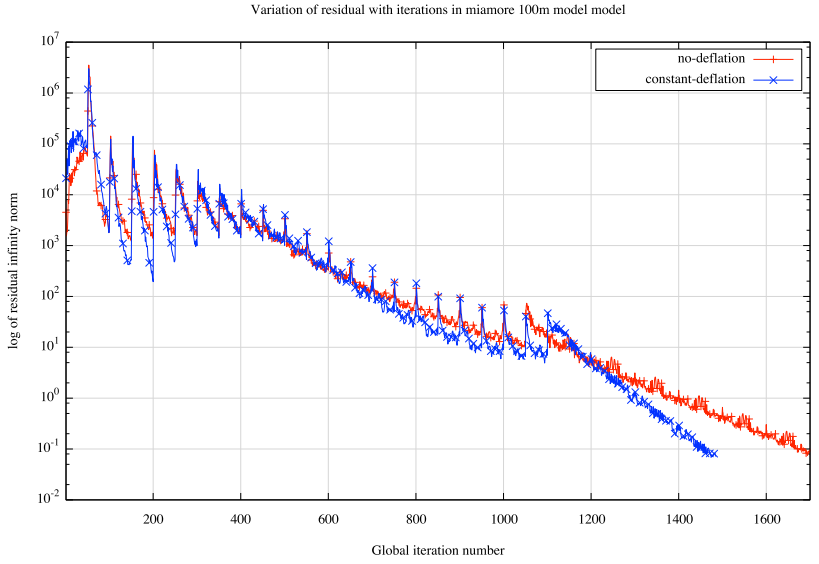
Table 7.6: Observations for miamore 100m model on 10 thin nodes (240 processes)

	No deflation	Constant deflation
Total iterations	3342	2703
Time to run	16 min, 55 secs	16 min, 33 secs

Table 7.7: Observations for miamore 50m model on 5 fat nodes (160 processes)

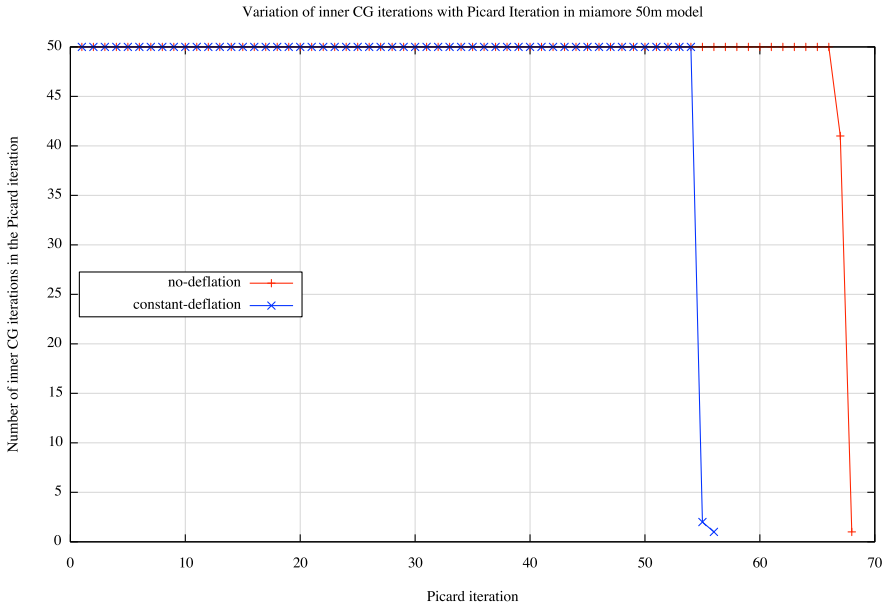


(a) Inner iteration per outer picard iteration

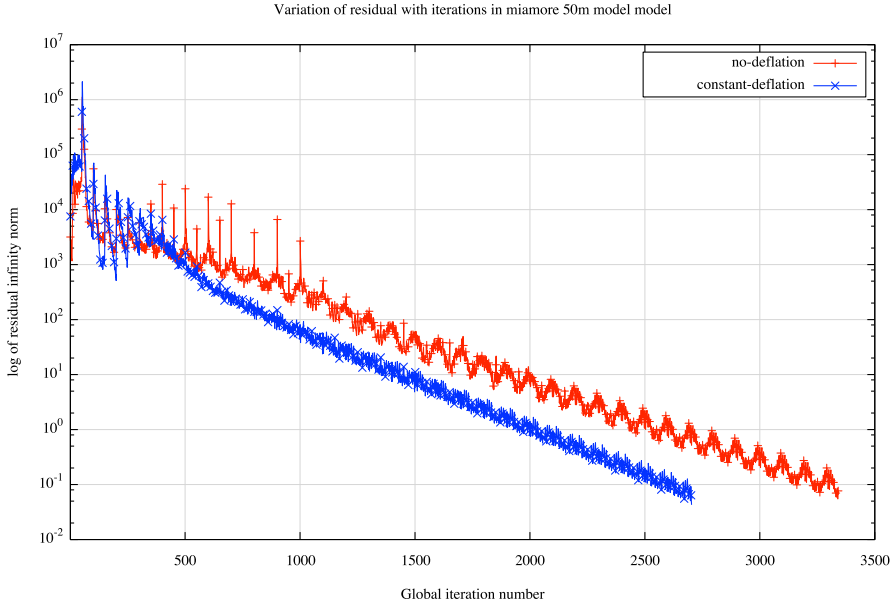


(b) Variation of the residual norm with subdomains

Figure 7.17: Observation for the 12272 ×13987 grid-cell in 100m miamore model using 240 subdomains



(a) Inner iteration per outer picard iteration



(b) Variation of the residual norm with subdomains

Figure 7.18: Observation for the 24544 ×27974 grid-cell in 50m miamore model using 160 subdomains

7.7. Results

In this section we present important results for NHI SS model.

7.7.1. Gain in iterations

We observe that the optimal number of subdomains for performance improvement of PCG using Deflation is 100. The gain in iterations for various cell sizes are tabulated below:

	PCG	CDPCG		LDPCG		LDPCG Speedup
Cell Size (m)	Iters	Iters	Speed up	Iters	Speed up	vs CDPCG Speedup
250	2527	1768	**1.43**	1496	**1.69**	1.18
100	10775	5209	**2.07**	3313	**3.25**	1.57
50	20927	10244	**2.04**	4966	**4.21**	2.06

Table 7.8: Speed up in iterations (Iters) for NHI SS model with 100 subdomains

7.7.2. Gain in wall clock time

The speed up factor of CDPCG and LDPCG with respect to PCG is shown below. We observe that the LDPCG performs better than CDPCG for higher number of subdomains in 100m NHI SS model. The improvement of Deflation is higher for intermediate number of subdomains (shown with red circle).

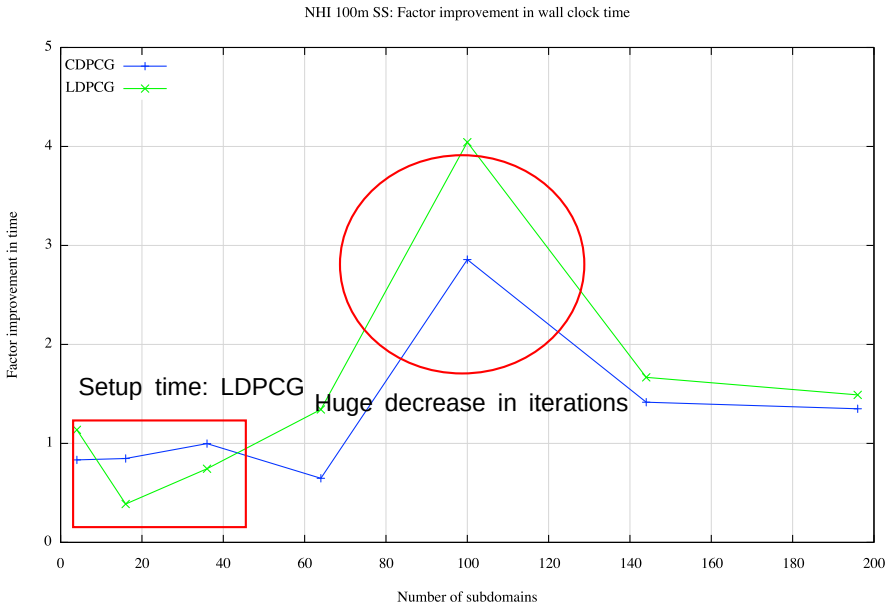


Figure 7.19: Wall clock time speed up factor in 3000 ×3250 ×7 grid cell in 50m NHI SS model

For 50m NHI SS model, the LDPCG works better than CDPCG for intermediate number of subdomains as shown below (shown with yellow circle).

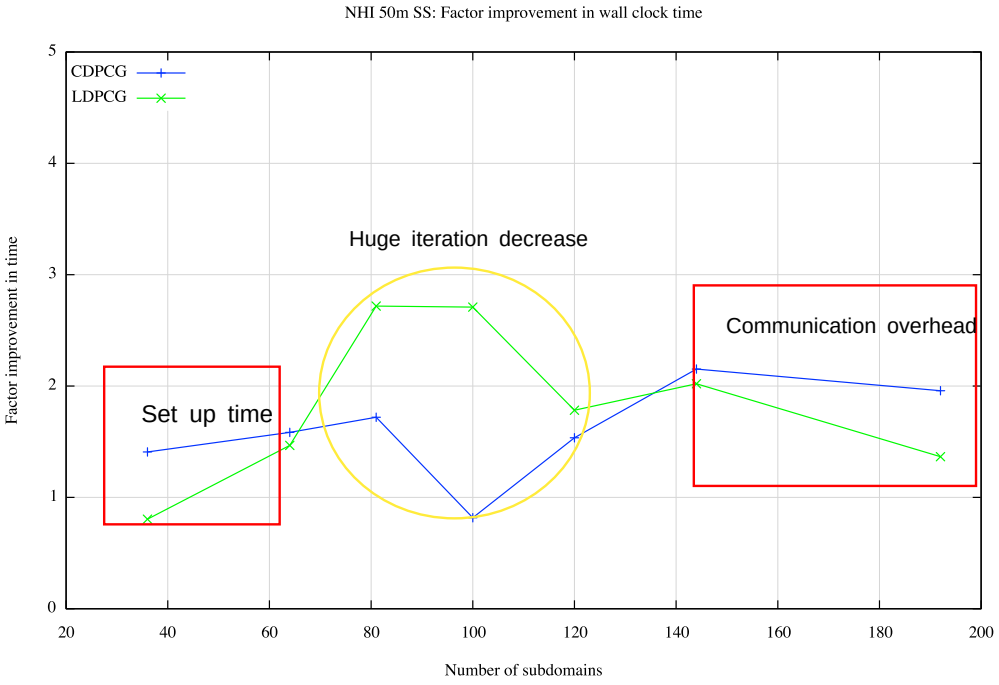


Figure 7.20: Wall clock time speed up factor in 6000 ×6500 ×7 grid cell in 50m NHI SS model

7.8. Code Optimization with Scalasca Profiler

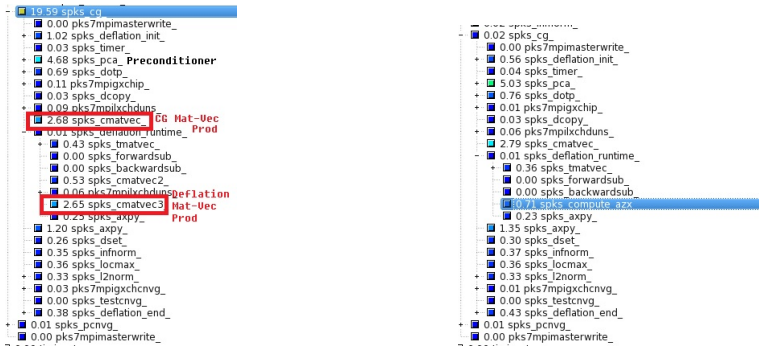
During the code development of Deflation in the PKS solver, we took several Scalasca Profiler readings to understand the code bottlenecks. We improved the run time of the deflated solver(s) after fixing the issue(s).

7.8.1. Storing AZ Matrix

A matrix AZ is obtained by multiplying the matrix A with the deflation matrix Z. AZ is used in deflation pre-processing phase and deflation run-time phase. In our initial implementation, we did not explicitly store the AZ matrix in deflation pre-processing phase and computed it again in the deflation run-time phase. Therefore, we did not gain any run time in the Deflated codes.

We see in figure 7.21a that, the time taken in CG matrix vector product (spks_cmatvec_) is 2.68 seconds, which is about same as time taken in matrix vector product used (2.65 seconds) in Deflation run time phase (spks_cmatvec3_) before the optimization. To resolve this issue, we store AZ matrix in deflation pre-processing phase. We see from figure 7.21b that, the matrix vector product time in Deflation run time phase (spks_compute_azx_) has reduced to 0.71.

We could resolve this problem only after checking the scalasca profiling report.



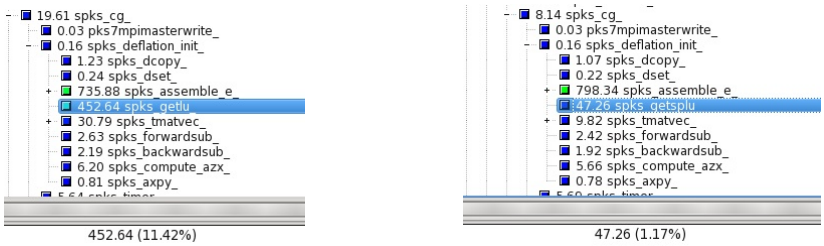
(a) Before optimization (b) After optimization

Figure 7.21: Scalasca snapshots describing runtime for $500 \times 500 \times 2$ iMOD unit case in CDPCG method

7.8.2. Sparse LU Decomposition of E

LU decomposition of matrix E is performed in Deflation pre processing phase. The entries in matrix E is zero corresponding to non-neighboring subdomains connections in CDPCG method. As the number of subdomains increases, the number of zero entries in E increases. For LDPCG method, the size of E is 4 times bigger than the size of E in CDPCG method. Therefore, matrix E in LDPCG gets bigger and becomes more sparse.

To decompose, the matrix E into L and U, we initially wrote a subroutine called `spks_getlu_` in the PKS solver. It does not utilize the sparsity in E and computes redundant pivots (whose value is zero) and redundant row updates in U. In scalasca profiler figure 7.22a, we observe that the subroutine spks_getsplu_ takes 11.42% of the whole simulation time. We wrote a subroutine called `spks_getsplu_`, in which we skip the row updates in U corresponding to the zero pivots. It has drastically reduced the LU decomposition time from 11.42% to 1.17% (see figure 7.22b).



(a) Before optimization (b) After optimization

Figure 7.22: Scalasca snapshots describing runtime for $1200 \times 1300 \times 7$ NHI SS model in LDPCG method

8

Conclusions and Recommendations

8.1. Conclusions

In our Master's project research, we have successfully implemented the CDPCG and LDPCG method in the PKS package. The primary goals have been achieved, and the research questions have been answered. In the speedup figure below, the base time in PCG, CDPCG and LDPCG is the respective serial run time, i.e serial speed up is 1 in PCG, CDPCG and LDPCG.

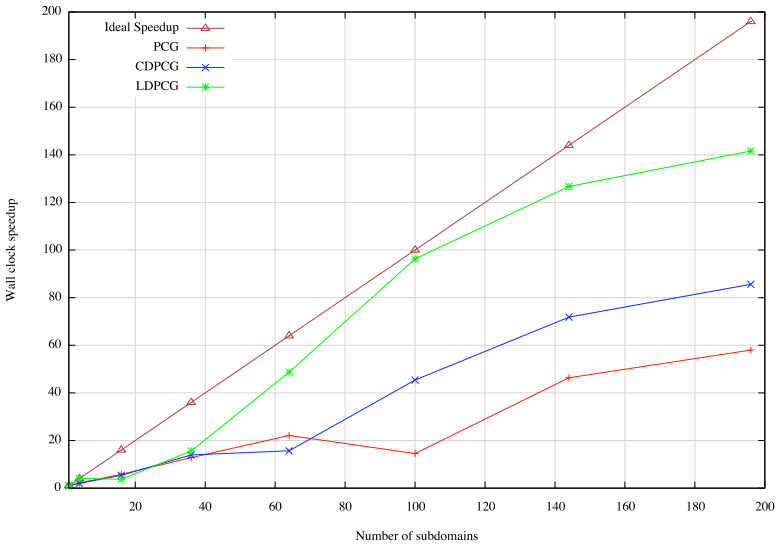


Figure 8.1: Speed up in wall clock time for 100 m NHI SS model

We conclude the following from our research:

- Deflation method works better for the higher number of subdomains. We have found that Deflation performs the best for 100 subdomains.
- The groundwater simulation time can be reduced by a factor **4** using Deflation preconditioner (using linear deflation vectors).
- The wall clock improvement is obtained due to huge decrease in iterations.
- Linear deflation vectors seems to be the optimal choice in the deflation preconditioner.

8.2. Recommendations

- Investigate the serial solver convergence: by changing the maximum number of inner iterations, checking accuracy of ILU(0) subdomain solve.
- We have considered only steady state simulation in our research. We recommend checking the performance of Deflated PCG code for NHI transient simulation.
- The number of iterations in SEAWAT (coupled groundwater and solute transport) code increase by almost a factor of two. We recommend using LDPCG method to see significant improvement in iterations.
- Construction of AZ matrix in Deflation pre-processing phase involves local communication of linear deflation vectors. The linear deflation vector entries can be constructed locally using a global index in the domain.
- Two global communications can be combined to reduce time in the Deflation pre processing phase.
- The performance of Deflation needs to be checked for 25 meter NHI SS model.
- Investigate the load imbalance in PCG and deflated PCG.
- We use both active and inactive cell indexes to calculate linear deflation vector entries in a structured grid. We recommend checking the effect LDPCG using indexes using only active cells.
- AS preconditioner has been used in the current implementation. What would be the impact when RAS is combined with Deflation?
- Two flags need to be added in input run-file to include CDPCG and LDPCG solvers in the next release of iMOD: DEFLATION and NDEFVEC. DEFLATION can be set to true or false. NDEFVEC can be set to 1 or 4 for $3D$ models to include CDPCG and LDPCG respectively.

Appendix

Matrix A

We have done some experiments on the exported large matrix A from MODFLOW. We denote A by A_{MOD}. For the Netherlands Hydrological Instrument (NHI) model, we have made a coarser model to the experiments. In the current coarser model, the grid size is $1000 \times 1000 \times 7$. The observations for 1^{st} time step and 1^{st} Picard iteration are:

- The square matrix A_{MOD} is symmetric.
- Dimension of the matrix A_{MOD} is 389429×389429.
- Largest eigenvalue $(\lambda_{max}) = 4.65 * 10^6$.
- Smallest eigenvalue $(\lambda_{min}) = 2.65 * 10^2$.
- Condition number of matrix $(\mathcal{K}) = 17520.65$.
- From above observations the matrix is SPD.

Diagonal Dominant

A square matrix A is said to be diagonally dominant if for every row of the matrix, the magnitude of the diagonal entry in a row is larger than or equal to the sum of the magnitudes of all the other (non-diagonal) entries in that row. More precisely, the matrix A_{MOD} is diagonally dominant if

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \quad \text{for all } i, \tag{8.1}$$

A Strictly diagonal dominant matrix have favorable properties, which helps to converge the iterative methods faster. We want to see the fraction of rows which are diagonally dominant in A_{MOD}, in the observations below:

- Fraction of diagonal dominant rows in Transient Simulation: 0.998041.
- Fraction of diagonal dominant rows in Steady State Simulation: 0.6825.

ILU Fill-in

Time taken and iterations to converge were noted for A_{MOD} with cg tolerance 10^{-5}, droptol denotes fill-in, droptol = 0 means full fill-in.

Case	Iterations	Time(s) to form M	Time(s) in CG loop
No Preconditioner	815	NA	12.6
ILU(0)	27	0.06	0.91
ILU, droptol = 0.002	31	2.51	0.95
ILU, droptol = 0.001	23	4.84	0.75
ILU, droptol = 0.00075	21	7.48	0.70
ILU, droptol = 0.0005	18	17.89	0.66
ILU, droptol = 0.0002	10	111.0	0.40

Table 8.1: Time and iterations for varying drop-tolerance

Overlap

For 1D Poisson's problem, we have computed the condition number of preconditioned system in Additive Schwarz (AS) and Restricted Additive Schwarz (RAS) for different overlap. We conclude that RAS performs best for higher value of overlap.

- Number of grid points: 1000
- Number of subdomains: 2

Ovelap	5	10	50	100	300
cond($M_{AS}^{-1}A$)	$7.85 * 10^3$	$4.06 * 10^3$	$1.06 * 10^3$	626	322
cond($M_{RAS}^{-1}A$)	$7.59 * 10^3$	$4.24 * 10^3$	761	351	89

Table 8.2: Effect of overlap on condition number

NHI SS 250m Model

Here, LDPCG stands for linear deflation vectors in x, y and z directions.

subdomains	PCG	CDPCG	LDPCG
1	1963	2114	2033
4	2193	2246	2178
9	2052	2055	1944
16	2192	2168	2102
25	2054	2029	1537
36	2168	2158	1720
49	2207	1776	1716
64	2377	2238	1540
81	2313	1673	1517
100	2527	1768	1496
121	2563	1607	1557
144	2504	1943	1523

Table 8.3: Total iterations in 250 m NHI SS model

subdomains	PCG	linear-X-deflation	linear-XY-deflation
1	1963	2127	2088
4	2193	2206	2178
9	2052	2074	2082
16	2192	2133	2132
25	2054	1985	1985
36	2168	1831	1900
49	2207	1912	1911
64	2377	1567	1592
81	2313	1584	1586
100	2527	1589	1606
121	2563	1589	1643

Table 8.4: Total iterations in 250 m NHI SS model

NHI SS 100m Model

subdomains	PCG	CDPCG	LDPCG
1	9967	10005	9501
4	10245	8967	7675
16	10410	10559	10561
36	10350	8811	9025
64	10770	9631	4837
100	10775	5209	3313
144	10905	6292	3420
196	11315	6658	3240

Table 8.5: Total iterations in 100 m NHI SS model

subdomains	PCG	CDPCG	LDPCG
1	5 HH, 9 MM, 3 SS	5 HH, 38 MM, 10 SS	8 HH, 26 MM, 37 SS
4	2 HH, 20 MM, 13 SS	2 HH, 48 MM, 17 SS	2 HH, 3 MM, 19 SS
16	53 MM, 2.015 SS	1 HH, 2 MM, 35 SS	2 HH, 16 MM, 52 SS
36	24 MM, 8.431 SS	24 MM, 11.912 SS	32 MM, 26.582 SS
64	13 MM, 57.191 SS	21 MM, 32.596 SS	10 MM, 23.553 SS
100	21 MM, 15.975 SS	7 MM, 26.627 SS	5 MM, 15.601 SS
144	6 MM, 39.927 SS	4 MM, 42.409 SS	3 MM, 59.860 SS
196	5 MM, 19.766 SS	3 MM, 56.918 SS	3 MM, 34.621 SS

Table 8.6: Wall clock time in 100 m NHI SS model

NHI SS 50m Model

#subdomains	PCG	CDPCG	LDPCG
36	19230	15094	11470
64	19242	13874	6537
81	19792	9978	6608
100	20927	10244	4966
120	21342	12712	7971
144	18662	11505	5468
192	18090	7390	5674

Table 8.7: Variation of total iterations with subdomains in 6000 ×6500 ×7 grid cell in 50m NHI SS model

#subdomains	PCG	CDPCG	LDPCG
36	4 HH, 25 MM	3 HH, 8 MM, 11 SS	5 HH, 29 MM, 35 SS
64	2 HH, 6 MM, 27 SS	1 HH, 19 MM, 51 SS	1 HH, 26 MM, 12 SS
81	2 HH, 32 MM, 52 SS	1 HH, 28 MM, 53 SS	56 MM, 14 SS
100	1 HH, 15 MM, 32 SS	1 HH, 32 MM, 25 SS	27 MM, 53.054 SS
120	1 HH, 2 MM, 50 SS	40 MM, 55 SS	35 MM, 15 SS
144	1 HH, 36 MM, 60 SS	45 MM, 4.024 SS	48 MM, 0.842 SS
192	31 MM, 34.777 SS	16 MM, 7.648 SS	23 MM, 7.590 SS

Table 8.8: Variation of wall clock with subdomains in 6000 ×6500 ×7 grid cell in 50m NHI SS model

Sample batch file

A sample batch file to run a PCG solver job on Cartesius on 100 MPI processes using 5 nodes is given below:

```
#!/bin/bash
#SBATCH -t 01:00:00
#SBATCH --output=nhiss_n100sca_no_def.out
#SBATCH --error=nhiss_n100sca_no_def.err
#SBATCH -p normal
#SBATCH --constraint=haswell
#SBATCH -N 5
#SBATCH -n 100
#SBATCH --distribution=block:block
module load scalasca
cp -r $HOME/imodflow_pks/only_modflow/tests/nhiss/* $TMPDIR
cd $TMPDIR
begintime=$(date +%s%N)
export SCOREP_EXECUTABLE=$HOME/imodflow_pks/only_modflow/bin/
imodflow-sca
scalasca -analyze srun $HOME/imodflow_pks/only_modflow/bin/
imodflow-nhi100m-nodef nhi_100m.run
endtime=$(date +%s%N)
echo "runtime: $(echo "scale=3;(${endtime} - ${begintime})/
(1*10^09)" | bc) seconds"
basedir=$HOME/imodflow_pks/only_modflow/tests/jobsnhiss/100m/
results
dir=no_def_100
mkdir -p ${basedir}/${dir}
cp $TMPDIR/results/head/*_l1*.idf ${basedir}/${dir}
cp $TMPDIR/results/head/*_l2*.idf ${basedir}/${dir}
cp $TMPDIR/results/mf2005_tmp/*.list* ${basedir}/${dir}
cp $TMPDIR/results/mf2005_tmp/*.dat ${basedir}/${dir}
basedir2=$HOME/imodflow_pks/only_modflow/tests/jobsnhiss/
100m/globalres
cp $TMPDIR/*RES*.dat ${basedir2}
basedir3=$HOME/imodflow_pks/only_modflow/tests/jobsnhiss/
100m/iiter_per_oiter
cp $TMPDIR/*iitr*.dat ${basedir3}
scoredir=$(echo score*)
cp -r $scoredir ${basedir}/${dir}
```

Iterations 2D Poisson Problem

Termination criteria for Residual is Inf Norm, (ICNVGOPT=0 in MODFLOW).

S.No.	Subdomains	PCG	CDPCG	$\|x\_PCG - x\_DPCG\|_\infty$
1	1	35	35	$6.1 * 10^{-7}$
2	2	55	58	$4.43 * 10^{-6}$
3	4	66	62	$3.79 * 10^{-6}$
4	8	73	72	$3.34 * 10^{-6}$
5	16	81	59	$6.56 * 10^{-6}$
6	32	94	62	$8.1 * 10^{-6}$
7	64	104	41	$7.14 * 10^{-6}$
8	128	117	42	$O(10^{-5})$
9	256	124	26	$O(10^{-5})$

Table 8.9: Variation of inner iteration for 1st Picard iteration in 100 ×100 ×1 Poisson problem

Nomenclature

Abbreviations

1D One dimensional

2D Two dimensional

3D Three dimensional

AS Additive Schwarz

ASM Additive Schwarz Method

BiCGSTAB Bi-Conjugate Gradient Stabilized

BIM Basic Iterative Method

CDPCG Deflated Preconditioned Conjugate Gradient with constant deflation vectors

CG Conjugate Gradient

DD Domain Decomposition

DPCG Deflated Preconditioned Conjugate Gradient method, general term to denote CDPCG and LDPCG method

GWF Ground Water Flow

HCLOSE Solver closing criteria using head

ICNVGOPT Option to select different norms for solver convergence in PKS

ILU Incomplete LU Decomposition

LDPCG Deflated Preconditioned Conjugate Gradient with linear deflation vectors in x, y and z direction

LHM Landelijk Hydrologisch Model

LSE Linear System of Equations

MPI Message Passing Interface

NHI Nederlands Hydrologisch Instrumentarium

PCG Preconditioned Conjugate Gradient

PDE Partial Differential Equation

PKS Parallel Krylov Solver

RAS Restrictive Additive Schwarz

RCB Recursive coordinate bisection

RCLOSE Solver closing criteria using residual

SLURM Simple Linux Utility for Resource Management

SPD symmetric positive definite

SS Steady State

USGS United States Geological Survey

References

[1] 500 most powerful commercially available computer systems. https://www.top500.org/.

[2] Computing the Cholesky Factorization of Sparse Matrices. http://www.tau.ac.il/~stoledo/Support/chapter-direct.pdf.

[3] Digital elevation model. https://www.wikiwand.com/en/Digital_elevation_model.

[4] DINOloket, the publishing portal of TNO, Geological Service Netherlands. https://www.dinoloket.nl/ondergrondmodellen.

[5] Distribution of Earth's Water. https://water.usgs.gov/.

[6] Domain decomposition methods. http://www.ddm.org/.

[7] Dutch national supecomputer cartesius. https://userinfo.surfsara.nl/systems/cartesius.

[8] Finite difference method. http://www.mathematik.uni-dortmund.de/~kuzmin/cfdintro/lecture4.pdf.

[9] Gershgorin circle theorem. https://www.wikiwand.com/en/Gershgorin_circle_theorem.

[10] iMOD, an efficient Deltares-version of MODFLOW. https://www.deltares.nl/en/software/imod/.

[11] iMOD User Manual, online version 3.6. https://content.oss.deltares.nl/imod/imod34/imod_um_html/imod-um-PKS-Parallel-Krylov-Solver-Package.html.

[12] Intel® Xeon® Processor E5-2690 v3. http://ark.intel.com/products/81713/.

[13] Landelijk Hydrologisch Model. http://www.nhi.nu/nl/index.php/toepassingen/nhi-lhm/.

[14] Scalasca Profiler. http://www.scalasca.org/.

[15] Slurm workload manager. https://slurm.schedmd.com/.

[16] Why is Groundwater So Important? http://www.brighthub.com/environment/science-environmental/articles/68744.aspx.

[17] Jarno Verkaik, Deflated Krylov-Schwarz Domain Decomposition for the Incompressible Navier-Stokes Equations on a Colocated Grid, Master Thesis 2003. http://ta.twi.tudelft.nl/nw/users/vuik/numanal/verkaik_afst.pdf, 2003.

[18] Tom Bamford. An Implementation of Domain Decomposition by Recursive Bisection, 2007.

[19] Xiao-Chuan Cai and Marcus Sarkis. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21:239–247, 1999.

[20] R.M. Dinkla. GMRES(m) with deflation applied to non-symmetric systems arising from fluid mechanics problems, 2009.

[21] Victorita Dolean, Pierre Jolivet, and Frédéric Nataf. *An introduction to domain decomposition methods*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2015. Algorithms, theory, and parallel implementation.

[22] Evridiki Efstathiou and Martin J. Gander. Why restricted additive schwarz converges faster than additive schwarz. *BIT Numerical Mathematics*, 43(5):945–959, 2003.

[23] J. Frank and C. Vuik. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing*, 23:442–462, 2001.

[24] Jason Frank and C Vuik. Parallel implementation of a multiblock method with approximate subdomain solution. *Applied numerical mathematics*, 30(4):403–423, 1999.

[25] Arlen W. Harbaugh. MODFLOW-2005, The U.S. Geological Survey Modular Ground-Water Model- The Ground-Water Flow Process. 2005.

[26] Jarno Verkaik. First Applications of the New Parallel Krylov Solver for MODFLOW on a National and Global Scale. ftp://ftp.geog.uu.nl/pub/posters/2016/First_Applications_of_the_New_Parallel_Krylov_Solver_for_MODFLOW_on_a_National_and_Global_Scale-Verkaik_Hughes_Sutanudjaja_Walsum-December2016-final.pdf.

[27] R. Nabben and C. Vuik. A comparison of Deflation and Coarse Grid Correction applied to porous media flow. *SIAM J. Numer. Anal.*, 42:1631–1647, 2004.

[28] Roy A Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *SIAM Journal on Numerical Analysis*, 24(2):355–365, 1987.

[29] L.A. Ros. L.A. Ros. Deflated CG Method for Modelling Groundwater Flow in a Layered Grid. http://ta.twi.tudelft.nl/users/vuik/numanal/ros_eng.html, 2008.

[30] Yousef Saad. *Iterative Methods for Sparse Linear Systems, Second Edition*. Society for Industrial and Applied Mathematics, 2 edition, 4 2003.

[31] Lloyd N Trefethen and David Bau III. *Numerical Linear Algebra*, volume 50. SIAM, 1997.

[32] J. Verkaik, C. Vuik, B. D. Paarhuis, and A. Twerda. *The Deflation Accelerated Schwarz Method for CFD*, pages 868–875. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[33] Jarno Verkaik. Profiling the massively parallel processing modflow code for pcr-globwb 3.0. 2013.

[34] C. Vuik and D.J.P. Lahaye. Scientific Computing Lecture Notes, TU Delft, 2015.

[35] C Vuik, A Segal, and J.A Meijerink. An efficient preconditioned cg method for the solution of a class of layered problems with extreme contrasts in the coefficients. *J. Comput. Phys.*, 152(1):385–403, June 1999.