2017

High Performance Techniques Applied in Partial Differential Equations Library

Shilei Lin
College of Saint Benedict/Saint John's University, lin.shilei@outlook.com

High Performance Techniques Applied in Partial Differential Equations Library

AN ALL COLLEGE THESIS

College of Saint Benedict | Saint John's University

In Partial Fulfillment
of the Requirements of Distinction
In the Department of *Computer Science* and *Mathematics*
by

Shilei Lin

May 2018

PROJECT TITLE:
High Performance Techniques Applied in Partial Differential Equations Library

BY: SHILEI LIN

Approved By:

Michael Heroux
Scientist in Residence of Computer Science

Michael Tangredi
Associate Professor of Mathematics

Imad Rahal
Associate Professor of Computer Science

Imad Rahal
Chair, Department of Computer Science

Robert Hesse
Chair, Department of Mathematics

Molly Ewing
Director, All College Thesis Program

ABSTRACT

This thesis explores various Trilinos packages to determine a method for updating the deal.ii library, which specializes in solving partial differential equations by finite element methods. It begins with introducing related concepts and general goals, followed by exploring computational and mathematical methods which are analytical solutions of one-dimensional Boussinesq equations and developing newer prototypes for solvers in deal.ii based on Trilinos packages. After demonstrating the methods, it indicates the reducing solving time in newer prototypes. Based on results from the prototype, similar methods are applied to update the deal.ii library. In the end, a testing program is exploited to demonstrate the improvement in performance for deal.ii.

Contents

List of Tables

List of Listings

List of Figures

1 Introduction

Real world phenomenon in physics such as heat dynamics, fluid dynamics, and quantum mechanics can mostly be represented by systems of partial differential equations (PDEs). Also, PDE systems can be enormous since they mostly describe phenomenon observed across dimensions. Therefore, PDE systems of this size are normally solved by computational software which is capable of formulating and solving large PDE systems.

Nowadays, solving large systems is still time-consuming. Thus, scientists are developing new software and hardware to improve efficiency. One approach to improve performance is by revising data structures and solving methods mainly through two high performance techniques: parallelism and templates.

1.1 Introduction to Parallel Computing

As opposed to traditional sequential computing, parallel computing takes advantage of computer architectures in which there are multiple processors. The goal of parallel computing is to reduce running time.[8]

The distributed memory model and the shared memory model are two major types of parallel computing models. For instance, the Message Passing Interface (MPI), a distributed memory model, relies on distributed networks, which is flexible and expressive. The OpenMP, a shared memory model, relies on multi-core processors, which is easier to program and debug[11].[1]

1.2 Introduction to Template Programming

'Templates' are a distinct feature in the C++ programming language that allows functions and classes to handle more data types or structures. Adopting 'template' programming has several benefits on my project, one of which reduces the repetition in codes and allows programs to accept user-made data types besides the predefined ones.

1.3 Introduction to Boussinesq Equation

In partial differential equations (PDEs), Boussinesq equations are often found in fluid dynamics, describing groundwater flow through an aquifer as a nonlinear parabolic PDE. The equation is named after Joseph Boussinesq, who first derived it in response to John Scott Russell's observation of the "wave of translation" (also known as a solitary wave or soliton). Boussinesq's 1872 paper introduces the equations now known as the Boussinesq equations[2].

[1]This project needs MPI-able environment.

2 Contributions to this Topic

This thesis intends to provide several contributions to the scientific computing field. First, it provides a detailed analytical solution for one-dimensional Boussinesq equations (flow equation). Secondly, it demonstrates reasons to update deal.ii by examining differences between various data services and solver abilities. Lastly, it provides the potential conversions and modifications of deal.ii with more recent Trilinos packages.

3 Problem Statement

When trying to solve a large partial differential system by using computers, a powerful software can be highly effective. Hardware technology advances rapidly, however, software takes years to catch up. When modifying and updating codes, we need to reflect the current computer architectures and programming languages.[8]

In this thesis, I propose to update the Trilinos wrapper classes in deal.ii. Currently, deal.ii (version 8.4.1) uses *Epetra* for its vector and matrix representations, *ML* and *MueLu*(partially) as its preconditioning, and *AztecOO* as its iterative linear solver. Throughout the years, Trilinos (version 12.10) has seen a newer version for data representation named *Tpetra*, and a newer version of linear solver named *Belos*.

To demonstrate the improvement of runtime performance, my intention is to use one of the tutorial programs in deal.ii 'step-32' which are embedded with Trilinos wrapper classes of deal.ii. The step-32 constructs and solves multi-dimensional Boussinesq equations. Thus, to have a better understanding of Boussinesq equations, my project explores analytical solution of one-dimensional Boussinesq equations as discussed in a 1984 paper written by Christos D. Tzimopoulos Panagiotis K. Tolikas, and Epaminondas G. Sidiropoulos.

4 Background

4.1 One-Dimensional Boussinesq Equation

In general, the one-dimensional Boussinesq Equation has the form

$$\frac{\partial h}{\partial t} = \frac{K}{S}\frac{\partial}{\partial x}(h\frac{\partial h}{\partial x}) \tag{4.1}$$

where K (in meters per second) and S (dimensionless) are hydraulic conductivity and specific yield of the aquifer, respectively; h (in meters) is the depth of water from the impermeable substratum; x is the horizontal distance from the origin; and t is the time. The boundary conditions for the equation are given by

$$x = 0 \qquad t > 0 \qquad h = h_1 \tag{4.2}$$

$$x > 0 \qquad t = 0 \qquad h = h_0 \tag{4.3}$$

where h_0 the initial depth of the water from the impermeable substratum, and h_1 the depth of water from the impermeable substratum at the origin. We consider this case as an abrupt raising of piezometric head at the origin, i.e., $h_1 > h_0$ [12]. We can approximate the solution as following:

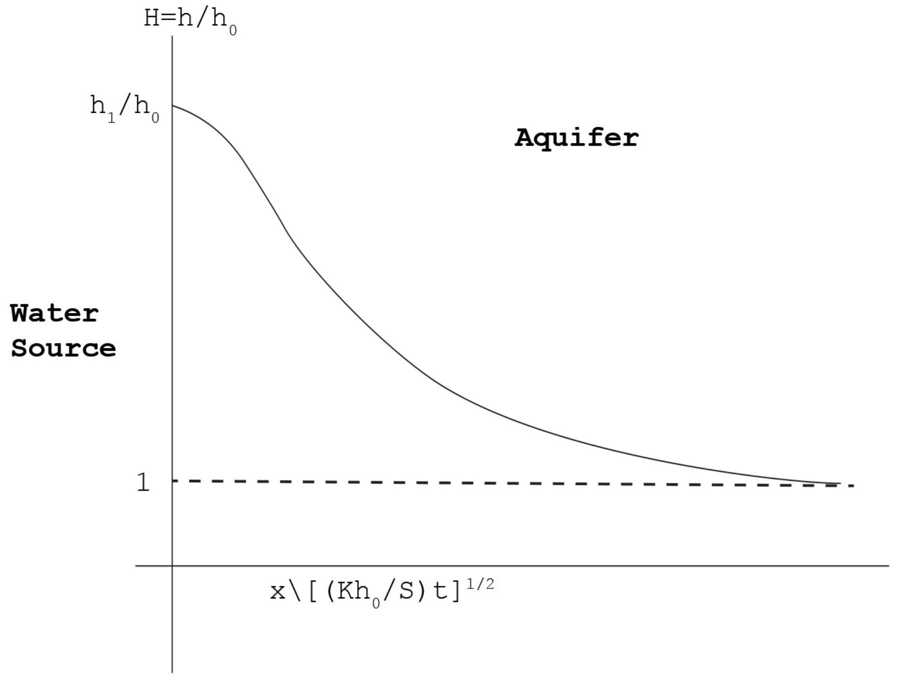

Figure 1: Expected piezometric solution curve for H. [12]

4.2 The deal.ii

'Deal' is the abbreviation of **D**ifferential **E**quations **A**nalysis **L**ibrary and 'II' indicates that it is the successor of 'Deal'. The deal.ii library is an open source C++ library that aims to solve partial differential equations by finite element methods. It allows rapid development of modern finite element codes via providing simple meshes programming content. The deal.ii library was introduced at Universität Heidelberg, Germany, based on works by the Numerical Methods Group, which concentrated in finite element methods and error estimation.[6]

4.3 Trilinos Packages

'Trilinos' is a collection of open-source software libraries, called *packages*, intended to be used as building blocks for the development of scientific applications. The word 'Trilinos' is from Greek and means 'a string of pearls', suggesting a number of software packages linked together by a common infrastructure. Trilinos was developed at Sandia National Laboratories from a core group of existing algorithms and utilizes the functions of software interfaces such as the BLAS, LAPACK, and MPI.[5]

4.3.1 Petra Packages

Petra is the Greek word for 'foundation'. Therefore in Trilinos, Epetra, Tpetra and Xpetra are representers for basic mathematical structures such as vectors, matrices, and graphs. In general, all Petra packages are capable of parallel execution on distributed memory machines.

1. **Epetra**
   The 'E' stands for 'essential' and defines the basic classes for distributed matrices and vectors, linear operators and linear problems. All packages in Trilinos support Epetra, which means that each package accepts Epetra objects as input[2][10]. Epetra supports double-precision numbers, floating point data, which can be extended to 64-bit indices. This allows powerful combinations among the various Trilinos functions. Epetra provides a high level of portability and stability. [8]

2. **Tpetra**
   The 'T' stands for 'template'. Tpetra can create *templated distributed linear Algebra objects*. Because Tpetra uses templates technique, it allows Tpetra to accept more data types than Epetra. In particular, Tpetra is based on two major data types: *ordinal type* and *scalar type*. The *ordinal type* is used for storing countable items, i.e. the number of non-zero elements in a matrix. The *scalar type* is the type of stored data, which can be varying from a complex number to a small 2×2 matrix. [10]

[2]The MueLu package only accepts Epetra or Tpetra objects wrapped by Xpetra.

3. **Xpetra**
   Xpetra is a wrapper interface which accepts both Epetra and Tpetra objects. The Xpetra also uses template technique like Tpetra. Xpetra enables algorithm developers to write to a single interface but be able to use either Epetra or Tpetra. Most importantly, Xpetra is used by MueLu preconditioner. [7]

4.3.2 Solver Packages

AztecOO and Belos are both iterative solver packages in Trilinos. The linear system has the form $Ax = B$, where A is the left-hand side n by n matrix, X is th solution, and B is the right-hand side.

1. **AztecOO**
   AztecOO is a linear solver package based on preconditioned Krylov methods. It only supports Epetra objects. [10]

2. **Belos**
   Belos provides next-generation iterative linear solvers and a powerful linear solver developer framework. Belos supports both Epetra and Tpetra objects. Belos also enables template, because user can benefit from Belos-defined abstract base classes, which can be considered as more efficient.[10]

4.3.3 Preconditioning Packages

ML and Muelu are both preconditioner packages in Trilinos.

1. **ML**
   ML is the algebraic (M)ulti(L)evel preconditioner package which has scalable preconditioning capabilities for a variety of problems. It can be used as the preconditioner for both AztecOO and Belos solvers, however, it only supports Epetra objects.[10]

2. **MueLu**
   MueLu is an extensible algebraic multi-grid library that is part of the Trilinos packages. MueLu only works with Xpetra interface which wraps either Epetra or Tpetra objects as long as the program is consistent with its data structure. The library is written in C++ and enables templated traits. Like Tpetra, The MueLu package allows for different ordinal and scalar types. Also, the MueLu package is designed to support various computer architectures from supercomputers to personal computers.

4.3.4 Finite Element and Matrix Generation Package

Galeri is the Greek word for *Gallery*. The Galeri can generate a variety of distributed linear systems. The Galeri packages can also generate several well-known finite element and finite difference matrices.

4.3.5 Summary of Trilinos Packages

Name	Description	Usage
Epetra	Essential package for building mathematical structures. Objects based on Epetra can be used in many other solver/preconditioning packages.	It is currently used in deal.ii-8.4.1.
Tpetra	A template based kernel for building mathematical structures. Also, it is a newer generation of Epetra.	To replace Epetra in deal.ii-8.4.1
Xpetra	An interface wraps E/Tpetra.	It is used by MueLu. In deal.ii ⌋ `preconditioningAMGMulue` class, Epetra objects are wrapped by Xpetra in order to use MueLu as preconditioner.
ML	Multigrid preconditioner (MGP). Main multigrid preconditioner package in Trilinos	Currently used by deal.ii.
MueLu	Newer generation of MGP. MueLu is a flexible, high-performance multigrid solver library.	Currently, it is in deal.ii but not supporting Tpetra.
AztecOO	Iterative solver. It allows flexible construction of matrix and vector arguments via Epetra matrix and vector classes.	Currently, deal.ii uses AztecOO as wrapper solver for Trilinos objects.
Belos	A next-generation iterative linear solvers and a powerful linear solver developer framework.	To replace AztecOO in deal.ii
Galeri	Galeri contains a suite of utilities and classes to generate a variety of (distributed) linear systems.	Prototypes use Galeri to generate problems. Not used in deal.ii

Table 1: Trilinos Packages.[8]

5 Methods

5.1 Solving the One Dimensional Flow Equation Analytically

In Section 4.1,the partial differential equation to be solved is

$$\frac{\partial h}{\partial t} = \frac{K}{S}\frac{\partial}{\partial x}\left(h\frac{\partial h}{\partial x}\right) \tag{5.1}$$

The boundary conditions of this partial differential equations are defined as

$$x = 0 \qquad t > 0 \qquad h = h_1 \tag{5.2}$$

$$x > 0 \qquad t = 0 \qquad h = h_0 \tag{5.3}$$

In order to solve this partial differential equation analytically, it needs to be reduced into an ordinary differential equation by substituting x and t as,

$$\eta = \frac{x}{[(Kh_0/S)t]^{1/2}} \tag{5.4}$$

Based on (5.2), (5.3) and (5.4), rewrite $h(x,t)$ as $H(\eta)$ as

$$h(x,t) = H(\eta) = H\left(\frac{x}{[(Kh_0/S)t]^{1/2}}\right) \tag{5.5}$$

$$H = h/h_0 \quad \mu = h_1/h_0 \tag{5.6}$$

Then, we can reduce (5.1) into ordinary differential equations with boundary condition (5.6) by applying chain-rule. It follows

$$\begin{aligned}
\frac{\partial h}{\partial t} &= \frac{K}{S}\frac{\partial}{\partial x}\left(h\frac{\partial h}{\partial x}\right)\\
\frac{\mathrm{d}H}{\mathrm{d}\eta}\cdot\frac{\partial \eta}{\partial t} &= \frac{K}{S}\frac{\mathrm{d}}{\mathrm{d}\eta}\left(h\cdot\frac{\mathrm{d}H}{\mathrm{d}\eta}\cdot\frac{\partial \eta}{\partial x}\right)\frac{\partial \eta}{\partial x}\\
\frac{\mathrm{d}H}{\mathrm{d}\eta}\cdot\frac{-x}{2t\sqrt{\frac{kth_o}{S}}} &= \frac{K}{S}\frac{\mathrm{d}}{\mathrm{d}\eta}\left(h\cdot\frac{\mathrm{d}H}{\mathrm{d}\eta}\cdot\frac{1}{\sqrt{\frac{Kth_0}{S}}}\right)\cdot\frac{1}{\sqrt{\frac{Kth_0}{S}}}\\
-\frac{\mathrm{d}H}{\mathrm{d}\eta}\cdot\frac{\eta}{2t} &= \frac{K}{S}\frac{\mathrm{d}}{\mathrm{d}\eta}\left(h\cdot\frac{\mathrm{d}H}{\mathrm{d}\eta}\right)\cdot\frac{S}{Kt\cdot h_0}\\
-\frac{\mathrm{d}H}{\mathrm{d}\eta}\cdot\frac{\eta}{2t} &= \frac{\mathrm{d}}{\mathrm{d}\eta}\left(h\cdot\frac{\mathrm{d}H}{\mathrm{d}\eta}\right)\frac{1}{t\cdot h_0}\\
-\frac{\mathrm{d}H}{\mathrm{d}\eta}\cdot\frac{\eta}{2} &= \frac{\mathrm{d}}{\mathrm{d}\eta}\left(\frac{h}{h_0}\cdot\frac{\mathrm{d}H}{\mathrm{d}\eta}\right)\\
-\frac{\mathrm{d}H}{\mathrm{d}\eta}\cdot\frac{\eta}{2} &= \frac{\mathrm{d}}{\mathrm{d}\eta}\left(H\cdot\frac{\mathrm{d}H}{\mathrm{d}\eta}\right)
\end{aligned} \tag{5.7}$$

Thus, the result of simplification produces the ordinary differential equation with boundary conditions (5.2) and (5.3) as

$$-\frac{\eta}{2}\frac{\mathrm{d}H}{\mathrm{d}\eta}=\frac{\mathrm{d}}{\mathrm{d}\eta}\left(H\frac{\mathrm{d}H}{\mathrm{d}\eta}\right) \tag{5.8}$$

$$\eta=0 \qquad H=\mu \tag{5.9}$$

$$\eta=\infty \qquad H=1 \tag{5.10}$$

Before we take integral to solve (5.8), we need to extract some properties from the differential equations first. From (5.8), at the origin using product rule, it follows

$$\frac{\mathrm{d}}{\mathrm{d}\eta}\left(H\frac{\mathrm{d}H}{\mathrm{d}\eta}\right)=\frac{\mathrm{d}H}{\mathrm{d}\eta}\cdot\frac{\mathrm{d}H}{\mathrm{d}\eta}+H\cdot\frac{\mathrm{d}}{\mathrm{d}\eta}\left(\frac{\mathrm{d}H}{\mathrm{d}\eta}\right) \tag{5.11}$$

Since we are looking at the origin when $\eta=0$, by simplifying (5.11) we have:

$$0=H\frac{\mathrm{d}^2H}{\mathrm{d}\eta^2}+\left(\frac{\mathrm{d}H}{\mathrm{d}\eta}\right)^2 \tag{5.12}$$

Theorem 1. *There exists an inflection point in $H(\eta)$.*

Remark. *Let $\eta=L\to\infty$, which L is the so-called penetration distance of the H [12].*

Proof. When $\eta=L\to\infty$, $H(\eta)$ approaches the horizontal asymptote $H(L)=1$, given by its boundary (5.10) and (5.12). At the origin the second derivate of $H(\eta)$ is $-\left(\frac{\mathrm{d}H}{\mathrm{d}\eta}\right)^2\Big/H<0$. Therefore, to allow $H(\eta)\to 1$ when $\eta\to\infty$, an inflection point must exist. □

Let η_k be the inflection point, where the second derivative of $H(\eta)$ is zero. Thus, from (5.8) at the inflection point η_k it follows

$$-\frac{\eta}{2}\frac{\mathrm{d}H}{\mathrm{d}\eta}=H\frac{\mathrm{d}^2H}{\mathrm{d}\eta^2}+\left(\frac{\mathrm{d}H}{\mathrm{d}\eta}\right)^2=0+\left(\frac{\mathrm{d}H}{\mathrm{d}\eta}\right)^2 \tag{5.13}$$

or

$$-\frac{\eta_k}{2}=\frac{\mathrm{d}H}{\mathrm{d}\eta}\Big|_{\eta=\eta_k} \tag{5.14}$$

Thus, let $H(\eta_k)=v$ and the main characteristics of $H(\eta)$ are

$$H(\eta_k)=v \qquad -\frac{\eta_k}{2}=\frac{\mathrm{d}H}{\mathrm{d}\eta}\Big|_{\eta=\eta_k} \qquad 0=\frac{\mathrm{d}^2H}{\mathrm{d}\eta^2}\Big|_{\eta=\eta_k} \qquad H(0)=\mu \quad H(L)=1 \tag{5.15}$$

Then, in the interval $\eta_k\le\eta\le L$, a way to express $H(\eta)$ by interpolating H approximately as a polynomial. We need to fit this polynomial respects characteristics above. Thus, we have the form

$$H=v-\frac{\eta_k}{2}(\eta-\eta_k)+A(\eta-\eta_k)^\lambda \tag{5.16}$$

where the unknowns are λ,η_k,v,A,L,μ.

5.1.1 The Important Properties of $H(\eta)$

This subsection demonstrates details in solving the ordinary differential equation form of H by find the properties of the unknowns via various algebraic manipulations.

- **Property 1: A+C=B+C**
  The following figure from previous section (Figure 1) is modified to highlight the area under the postulated solution curve. We extract the first property from the this figure.

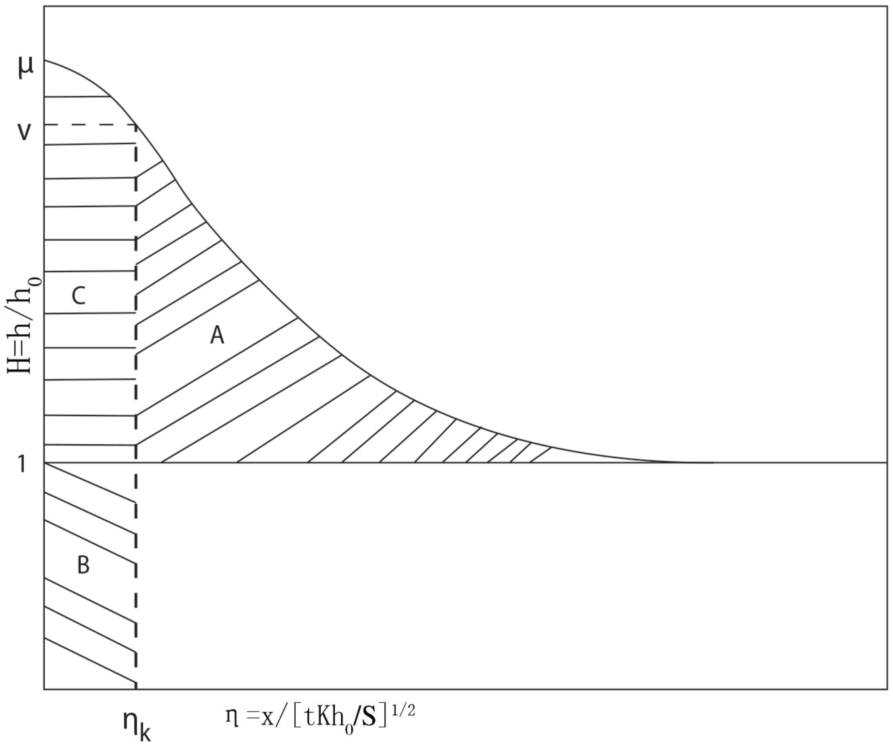

Figure 2: Modified postulated piezometric solution curve for H. [12]

Integrate (5.8) from $\eta = \eta_k$ to $L \to \infty$ yields

$$\int_{\eta_k}^{L} -\frac{\eta}{2}\frac{\mathrm{d}H}{\mathrm{d}\eta}d\eta = \int_{\eta_k}^{L} \frac{\mathrm{d}}{\mathrm{d}\eta}\left(H\frac{\mathrm{d}H}{\mathrm{d}\eta}\right)d\eta \tag{5.17}$$

Applying integration by parts, the left-hand side of (5.17) yields

$$\begin{aligned}\int_{\eta_k}^{L} -\frac{\eta}{2}\frac{\mathrm{d}H}{\mathrm{d}\eta}d\eta &= -\frac{\eta}{2}\cdot H\Big|_{\eta_k}^{L} + \frac{1}{2}\int_{\eta_k}^{L} Hd\eta \\ &= -\frac{L\cdot 1}{2} + \frac{\eta_k \cdot v}{2} + \frac{1}{2}\int_{\eta_k}^{L} Hd\eta\end{aligned} \tag{5.18}$$

Similarly, the right-hand side of (5.17) yields[3]

$$\int_{\eta_k}^{L} \frac{\mathrm{d}}{\mathrm{d}\eta}\left(H\frac{\mathrm{d}H}{\mathrm{d}\eta}\right) d\eta = H\frac{\mathrm{d}H}{\mathrm{d}\eta}\Big|_{\eta_k}^{L} = 1\cdot 0 - v\cdot\left(-\frac{\eta_k}{2}\right) = \frac{v\eta_k}{2} \tag{5.19}$$

Thus, putting (5.18) and (5.19) together, it follows

$$\begin{aligned} -\frac{L\cdot 1}{2} + \frac{v\eta_k}{2} + \frac{1}{2}\int_{\eta_k}^{L} H d\eta &= \frac{v\eta_k}{2} \\ \eta_k - L + \int_{\eta_k}^{L} H d\eta &= \eta_k \\ \int_{\eta_k}^{L}(-1)d\eta + \int_{\eta_k}^{L} H d\eta &= \eta_k \end{aligned} \tag{5.20}$$

or

$$\int_{\eta_k}^{L}(H-1)d\eta = \eta_k \tag{5.21}$$

This equation can be interpreted as the area of A is the same as B from Figure 2. Thus, the area bounded by the H solution curve and the lines $H = 1$ and $\eta = \eta_k$ is equal to η_k.

Theorem 2. *In Figure 2, Area$(C + B)$ = Area$(C + A)$*

Proof. Based on Calculus knowledge, (5.21) yields,

$$\int_{\eta_k}^{L}(H-1)d\eta = \int_{0}^{L}(H-1)d\eta - \int_{0}^{\eta_k}(H-1)d\eta \tag{5.22}$$

or

$$\begin{aligned} \int_{0}^{L}(H-1)d\eta &= -\int_{0}^{\eta_k}(H-1)d\eta + \int_{\eta_k}^{L}(H-1)d\eta \\ &= \int_{0}^{\eta_k}(H-1)d\eta + \eta_k \\ &= \int_{0}^{\eta_k}(H-1)d\eta + \int_{0}^{\eta_k}1d\eta \\ &= \int_{0}^{\eta_k}H d\eta \end{aligned} \tag{5.23}$$

Thus, we have

$$\int_{0}^{\infty}(H-1)d\eta = \int_{0}^{\eta_k}H d\eta \tag{5.24}$$

It suggests that the area between the H solution curve and the line $H = 1$ equals the area bounded by the H curve, the axes $H = 0$ and $\eta = 0$, and the line $\eta = \eta_k$. Thus,Area$(C + B)$ = Area$(C + A)$[12]. □

[3]The first derivative of H equals to 0 when $\eta \to \infty$, for H approaches horizontal asymptote.

- **Property 2: Two Integral Values at Inflection Points**

  Then, multiply (5.8) by η and integrate it from $\eta = 0$ to $\eta = \eta_k$ yields

$$\int_0^{\eta_k} -\frac{\eta^2}{2}\frac{\mathrm{d}H}{\mathrm{d}\eta} d\eta = \int_0^{\eta_k} \eta \frac{\mathrm{d}}{\mathrm{d}\eta}\left(H\frac{\mathrm{d}H}{\mathrm{d}\eta}\right) d\eta \tag{5.25}$$

  Apply integration by parts on left-hand sides of (5.25), it follows

$$\int_0^{\eta_k} -\frac{\eta^2}{2}\frac{\mathrm{d}H}{\mathrm{d}\eta} d\eta = -\frac{\eta^2}{2}\cdot H\Big|_0^{\eta_k} + \int_0^{\eta_k} \eta H d\eta = -\frac{v\eta_k^2}{2} + \int_0^{\eta_k} \eta H d\eta \tag{5.26}$$

  Similarly, right-hand sides it follows

$$\begin{aligned}\int_0^{\eta_k} \eta \frac{\mathrm{d}}{\mathrm{d}\eta}\left(H\frac{\mathrm{d}H}{\mathrm{d}\eta}\right) d\eta &= H\eta\frac{\mathrm{d}H}{\mathrm{d}\eta}\Big|_0^{\eta_k} - \int_0^{\eta_k}\left(H\frac{\mathrm{d}H}{\mathrm{d}\eta}\right) d\eta \\ &= -\frac{v\eta_k^2}{2} - \frac{H^2}{2}\Big|_0^{\eta_k} \\ &= -\frac{v\eta_k^2}{2} - \frac{v^2-\mu^2}{2}\end{aligned} \tag{5.27}$$

  Then, putting (5.26) and (5.27) together, we have

$$\int_0^{\eta_k} \eta H d\eta = \frac{\mu^2 - v^2}{2} \tag{5.28}$$

  Similarly, multiply (5.8) by η and integrate it from $\eta = \eta_k$ to $\eta = \infty$ yields

$$\int_{\eta_k}^{\infty} -\frac{\eta^2}{2}\frac{\mathrm{d}H}{\mathrm{d}\eta} d\eta = \int_{\eta_k}^{\infty} \eta \frac{\mathrm{d}}{\mathrm{d}\eta}\left(H\frac{\mathrm{d}H}{\mathrm{d}\eta}\right) d\eta \tag{5.29}$$

  Apply integration by parts on (5.29), it yields

$$\begin{aligned}\int_{\eta_k}^{\infty} -\frac{\eta^2}{2}\frac{\mathrm{d}H}{\mathrm{d}\eta} d\eta &= \int_{\eta_k}^{\infty} \eta \frac{\mathrm{d}}{\mathrm{d}\eta}\left(H\frac{\mathrm{d}H}{\mathrm{d}\eta}\right) d\eta \\ \left(\frac{\eta_k^2}{2}\right) - \frac{L^2}{2} + \int_{\eta_k}^{\infty} \eta H d\eta &= -\frac{1+v^2}{2} + \left(\frac{\eta_k^2}{2}\right) \\ \int_{\eta_k}^{\infty} -\eta d\eta + \int_{\eta_k}^{\infty} \eta H d\eta &= -\frac{1+v^2+\eta_k^2}{2}\end{aligned} \tag{5.30}$$

  or

$$\int_{\eta_k}^{\infty} \eta(H-1) d\eta = \frac{v^2 - 1 + \eta_k^2}{2} \tag{5.31}$$

5.1.2 Polynomial Approximation Of the Solution

Now, we are ready to find all the unknowns of

$$H = v - \frac{\eta_k}{2}(\eta - \eta_k) + A(\eta - \eta_k)^{\lambda} \tag{5.32}$$

We need to determine constants A and λ. Based on all boundary conditions, we can make following boundary conditions based on (5.32)

$$H(L) = 1 = v - \frac{\eta_k}{2}(L - \eta_k) + A(\eta - \eta_k)^{\lambda} \tag{5.33}$$

$$\frac{\mathrm{d}H}{\mathrm{d}\eta}\Big|_{\eta=L} = 0 = -\frac{\eta_k}{2} + A \cdot \lambda (L - \eta_k)^{\lambda-1} \tag{5.34}$$

In (5.21), we can substitute $H(\eta)$ by the expression in (5.32). On the left-hand side of (5.21) we have

$$\begin{aligned}
\int_0^{\infty} (H-1)\,d\eta &= \int_0^{\infty} \left(v - \frac{\eta_k}{2}(\eta - \eta_k) + A(\eta-\eta_k)^{\lambda} - 1\right) d\eta \\
&= \left(-\eta + v\eta - \frac{\eta^2\eta_k}{4} + \frac{\eta{\eta_k}^2}{2} + A(\eta-\eta_k)^{\lambda}\left(\frac{\eta}{1+\lambda} - \frac{\eta_k}{1+\lambda}\right)\right)\Big|_0^{\infty} \\
&= -\eta\Big|_0^{\infty} + v\eta\Big|_0^{\infty} - \frac{\eta^2\eta_k}{4}\Big|_0^{\infty} + \frac{\eta{\eta_k}^2}{2}\Big|_0^{\infty} + \frac{A(\eta-\eta_k)^{1+\lambda}}{1+\lambda}\Big|_0^{\infty} \\
&= -L + \mathrm{vL} - \frac{L^2\eta_k}{4} + \frac{L{\eta_k}^2}{2} + \frac{A(L-\eta_k)^{1+\lambda}}{1+\lambda}
\end{aligned} \tag{5.35}$$

while on the right-hand side we have

$$\begin{aligned}
\int_0^{\eta_k} H\,d\eta &= \int_0^{\eta_k} \left(v - \frac{\eta_k}{2}(\eta - \eta_k) + A(\eta-\eta_k)^{\lambda}\right) d\eta \\
&= \left(v\eta - \frac{\eta^2\eta_k}{4} + \frac{\eta{\eta_k}^2}{2} + A(\eta-\eta_k)^{\lambda}\left(\frac{\eta}{1+\lambda} - \frac{\eta_k}{1+\lambda}\right)\right)\Big|_0^{\eta_k} \\
&= v\eta\Big|_0^{\eta_k} - \frac{\eta^2\eta_k}{4}\Big|_0^{\eta_k} + \frac{\eta{\eta_k}^2}{2}\Big|_0^{\eta_k} \\
&= v\eta_k + \frac{{\eta_k}^3}{4}
\end{aligned} \tag{5.36}$$

So put (5.35) and (5.36) together

$$\begin{aligned}
(\eta_k) - L + \mathrm{vL} - \frac{L^2\eta_k}{4} + \frac{L{\eta_k}^2}{2} + \frac{A(L-\eta_k)^{1+\lambda}}{1+\lambda} &= v\eta_k + \frac{{\eta_k}^3}{4}(+\eta_k) \\
(v-1)(L-\eta_k) - \frac{\eta_k(L-\eta_k)^2}{4} &= \eta_k
\end{aligned} \tag{5.37}$$

Similarly, from (5.32), and (5.21),

$$\begin{aligned}
\int_{\eta_k}^{\infty}\eta(H-1)d\eta &= \int_{\eta_k}^{\infty}\eta(H-1)\,d\eta - \int_{\eta_k}^{\infty}\eta_k(H-1)\,d\eta + \int_{\eta_k}^{\infty}\eta_k(H-1)\,d\eta \\
&= \eta_k^2 + \int_{\eta_k}^{\infty}(\eta-\eta_k)(H-1)d\eta \\
&= \eta_k^2 + \int_{\eta_k}^{\infty}(\eta-\eta_k)\left(v - \frac{\eta_k}{2}(\eta-\eta_k) + A(\eta-\eta_k)^{\lambda} - 1\right)d\eta \\
&= \eta_k^2 + \left(\frac{A(\eta-\eta_k)^{\lambda+2}}{\lambda+2} - \frac{\eta^3\eta_k}{6} + \frac{\eta^2\eta_k^2}{2} - \frac{\eta^2}{2} - \frac{\eta\eta_k^3}{2} + \eta\eta_k + \frac{\eta^2 v}{2} - \eta\eta_k v\right)\Big|_{\eta_k}^{\infty} \\
&= \eta_k^2 + \left(\frac{A(L-\eta_k)^{\lambda+2}}{\lambda+2} - \frac{L^3\eta_k}{6} + \frac{L^2\eta_k^2}{2} - \frac{L^2}{2} - \frac{L\eta_k^3}{2} + L\eta_k + \frac{L^2 v}{2} - L\eta_k v\right) \\
&\quad - \left(-\frac{\eta_k^4}{6} + \frac{\eta_k^2}{2} - \frac{\eta_k^2 v}{2}\right) \\
&= \frac{A\left(\eta-\eta_k\right)^{\lambda+2}}{\lambda+2} - \frac{1}{6}\eta_k\left(L-\eta_k\right)^3 + \frac{1}{2}(v-1)\left(L-\eta_k\right)^2 + \eta_k^2
\end{aligned} \tag{5.38}$$

From(5.31), we can find such equality

$$\frac{A\left(\eta-\eta_k\right)^{\lambda+2}}{\lambda+2} - \frac{1}{6}\eta_k\left(L-\eta_k\right)^3 + \frac{1}{2}(v-1)\left(L-\eta_k\right)^2 = -\frac{\eta_k^2}{2} + \frac{v^2}{2} - \frac{1}{2} \tag{5.39}$$

Also, in the interval of $0 \le \eta \le \eta_k$, the area under the $H(\eta)$ is taken approximated as a linear function on interval $[0, \eta_k]$. From the Figure 2, we learn that $H(\eta)$ across points $(0, \mu)$ and (η_k, v). Put two points into a linear function, we have

$$H = \frac{v-\mu}{\eta_k}\eta + \mu \tag{5.40}$$

Thus, left hand side of (5.28) can be reduced into

$$\int_0^{\eta_k}\left(\frac{v-\mu}{\eta_k}\eta^2 + \mu\right)d\eta = \eta_k^2 \cdot \frac{2v+\mu}{6} \tag{5.41}$$

and consequently (5.28) yields

$$\eta_k^2 = 3\frac{\mu^2 - v^2}{\mu + 2v} \tag{5.42}$$

Equations (5.33),(5.34),(5.37),(5.39) and (5.28) construct a five parameters systems. The unknowns are $\lambda, \eta_k, v, A, L, \mu$. Then, after some algebraic manipulations, we have the iterative system with 4 intermediate variables D_1, D_2, D_3 and D_4, as following

$$\begin{aligned} D_1 &= \frac{8\lambda}{\lambda-1} - \frac{4\lambda^2}{(\lambda-1)^2} + \frac{8\lambda^2}{(\lambda-1)^2}\left(\frac{1}{\lambda(\lambda+1)}\right) \\ D_2 &= \frac{v-1}{\eta_k} = \frac{2}{\sqrt{D_1}} \\ D_3 &= L - \eta_k = \frac{2\lambda}{\lambda-1} D_2 \\ D_4 &= D_2 D_3^2 - \frac{D_3^3}{3} + \frac{D_3^3}{\lambda(\lambda+2)} \\ \eta_k &= (D_4 - 2D_2)(-1 + {D_2}^2)^{-1} \\ \frac{A}{\eta_k} &= \frac{1}{2\lambda D_3^{\lambda-1}} \end{aligned} \tag{5.43}$$

Up to this point, we have all the parameters worked out and the iteration can be carried out by choosing an initial value for $\lambda > 2$. simplifying it further, we have all the parameters from (5.43)

$$\begin{aligned} \eta_k &= (D_4 - 2D_2)\left(-1 + D_2^2\right)^{-1} \\ v &= \frac{2\eta_k}{\sqrt{D_1}} + 1 \\ L &= \frac{2\lambda}{\lambda-1} D_2 + \eta_k \\ A &= \frac{\eta_k}{2\lambda D_3^{\lambda-1}} \end{aligned} \tag{5.44}$$

Also, from (5.44) we have

$$3\mu^2 - \eta_k^2\mu - 3v^2 - 2v\eta_k^2 = 0 \tag{5.45}$$

To solve μ, we need to use quadratic formula:

$$\mu = \left(-\eta_k^2 + \sqrt{\eta_k^4 + 12\left(3v^2 + 2\eta_k^2 v\right)}\right)/6 \tag{5.46}$$

I use *Matlab* to assist for solving the system. The *Matlab* code is listed in Section 8.4. Also, the result is demonstrated in the Section 6.1.

5.2 Prototype Development

In previous work conducted by Emily Furst, she created three prototypes and compared them in different problem sizes: 50^3, 120^3, 190^3, 260^3,330^3 and 400^3. The results in her paper suggested that *MueLu + Belos*[4] are a suitable way to update deal.ii, which currently uses *ML + AztecOO*. When she used *Xpetra* objects, however, she instantiated *Xpetra* to *Epetra*. Thus, in my prototype, I instantiate *Xpetra* to *Tpetra* moving a step forward.

Thus, to show the differences, three different prototypes are developed, which each one solves a similar problem generated by *Galeri*. The mathematics problem represents by *Xpetra* interface, which builds a(n) either *Epetra* or *Tpetra* expressed by the command `Xpetra::UseEpetra` for *Epetra* or `Xpetra::UseTpetra` for *Tpetra*. In the following code, *MueLu* is used for instantiating *Xpetra* objects into *Epetra* or *Tpetra* objects.

Listing 1: Problem Generation for Prototypes[9].

```
typedef double     scalar_type;
typedef int       local_ordinal_type;
typedef int       global_ordinal_type;
typedef KokkosClassic::DefaultNode::DefaultNodeType        node_type;
typedef Tpetra::Map<local_ordinal_type,global_ordinal_type,node_type>
↪  driver_map_type;
RCP<const Map>    xpetraMap =
↪  MapFactory::Build(Xpetra::UseEpetra,matrixParameters.GetNumGlobalElements(),
↪  indexBase, comm);
RCP<GaleriXpetraProblem> Pr = Galeri::Xpetra::BuildProblem<SC, LO, GO, Map,
↪  CrsMatrixWrap, MultiVector>(matrixParameters.GetMatrixType(), xpetraMap,
↪  matrixParameters.GetParameterList());
RCP<Matrix>         xpetraA  = Pr->BuildMatrix();
RCP<crs_matrix_type>       A  = MueLuUtilities::Op2NonConstEpetraCrs(xpetraA);
const driver_map_type   map  = MueLuUtilities::Map2EpetraMap(*xpetraMap);
```

Then, first comparison is between *MLAztecOOEpetra* and *MLBelosEpetra*. They both use *Epetra* as its data service, *ML* as its preconditioner, but first one uses *AztecOO* as its solver and second one uses *Belos* as the solver. Then, second comparison is between *MLBelosEpetra* and *MueLuBelosTpetra*. The *MueLuBelosTpetra* is adopted from an example from *MueLu* packages that uses *Tpetra*, *MueLu* as its preconditioner, and *Belos*. These two comparisons indicate improvements in solving time. Details of comparisons results are in Section 6.2.

[4]It means that the program uses *MueLu* as the preconditioner and *Belos* as the solver.

5.3 Updating deal.ii

The comparison results from prototypes suggests that *Tpetra+MueLu+Belos* has the best performance in large size problems. Thus, I expect a similar imporvement in performance after updating deal.ii..

To begin this updating process, I take three steps. Since *Epetra* is currently used by deal.ii, replacing all the data types and functionalities at once is infeasible. Furthermore, as I mention before, currently, `deal.ii::TrilinosWrappers` is using *AztecOO* as its linear solver and *ML* as its preconditioner, which both are not compatible with *Tpetra*.

Thus, I decide to start with updating the linear solver *AztecOO*, which means by updating *AztecOO* to *Belos*, since *Belos* is compatible with both *Epetra* and *ML*.

Next, after I successfully update the linear solver, I turn to convert *Epetra* objects to *Tpetra* objects.

5.3.1 Updating Solver from AztecOO to Belos

To update the linear solver in `deal.ii::TrilinosWrappers` from *AztecOO* to *Belos*, we need to complete a list of things. It starts with modifying 'header file' *trilinos_solver.h*. While using Trilinos package to solve problem, a `linearProblem` object must be created first. Also, to create a Belos solver, a new `SolverManager` object and a new `SolverFactory` object need to be created. When those are finished, we can fully replace `AztecOO` solver. The new solver can choose one of the *Belos* solver algorithms, i.e. block GMRES, block CG, pseudo-block CG, pseudo-block GMRES. [1]

The following list provides the process of modification on *trilinos_solver.h*.

1. Using new namespaces and abbreviating some variable names;

Listing 2: New Namespaces and Type Abbreviations in trilinos_solver.h

```
//namespace:
using Teuchos::ParameterList;
using Teuchos::RCP;
using Teuchos::rcp;
using Teuchos::rcpFromRef;
//Type Abbreviations or Shortname
typedef double                    ST;
typedef Epetra_MultiVector        MV;
typedef Epetra_Operator           OP;
```

Listing 3: The trilinos_solver.h before modification

```
std_cxx11::shared_ptr<Epetra_LinearProbl
↪   em >
↪   linear_problem;
AztecOO solver;
```

Listing 4: The trilinos_solver.h after modification

```
RCP<Belos::LinearProblem<ST,MV,OP>>
↪   linear_problem;
RCP<Belos::SolverManager<ST,MV,OP>>
↪   newSolver;
RCP<Belos::SolverFactory<ST,MV,OP>>
↪   factory;
```

2. Change `Epetra_LinearProblem` into `Belos::linearProblem<ST,MV,OP>`;
3. Change `AztecOO` into `Belos::SolverManager<ST,MV,OP>` and `Belos::SolverFactory<ST,MV,OP>`.
4. Change some functions return type from `void` to `double`, to examine residuals.

Next, modify *trilinos_solver.cc*. Before solving the problem, deal.ii uses a set of different setup functions, `solve`, which initializes the `LinearPrblem`. The following code show modifications on one of thos setup functions.

Remark (Highlights)**.** *The highlighted codes indicate important modifications.*

Listing 5: `solve` in trilinos_solver.cc before modification

```
void SolverBase::solve (
  Epetra_Operator          &A,
  VectorBase               &x,
  const VectorBase         &b,
  const PreconditionBase &preconditioner)
{
  linear_problem.reset();

  linear_problem.reset(
    new Epetra_Linearproblem(&A,
      &x.trilinos_vector(),
      const_cast<Epetra_MultiVector *>
      (&b.trilinos_vector())));

  do_solve(preconditioner);
}
```

Listing 6: `solve` in trilinos_solver.cc after modification

```
void SolverBase::solve(
  Epetra_Operator          &a,
  VectorBase               &x,
  const VectorBase         &b,
  const PreconditionBase &preconditioner)
{
  linear_problem.reset();
  //Transfer std_cxx11::shared_ptr object
  RCP<const OP> A =
  ↪   rcpFromRef(*(const_cast<OP *>(&a)));
  RCP<MV> X = rcpFromRef(
    *(const_cast<Epetra_FEVector *>
    (&x.trilinos_vector())));
  RCP<MV> B = rcpFromRef(*(const_cast<MV
  ↪   *>(&b.trilinos_vector())));
  //create Belos linear_problem object
  linear_problem = rcp(new
  ↪   Belos::LinearProblem<ST, MV, OP>(A,
  ↪   X, B));
  return do_solve(preconditioner);
}
```

Finally, we can update the solving function `do_solve`. The following code shows modifications. This completes the modification on the linear solver in `deal.ii::TrilinosWrappers` from *AztecOO* to *Belos*.

Listing 7: `do_solve` in trilinos_solver.cc after modification

```
void
SolverBase::do_solve(const PreconditionBase &preconditioner){

  factory = rcp(new Belos::SolverFactory<ST,MV,OP>);
  int max_iters = solver_control.max_steps();
  double tol = solver_control.tolerance();

  RCP<ParameterList> solverParams = rcp(new ParameterList());

  solverParams->set("Maximum Iterations", max_iters);
  solverParams->set("Convergence Tolerance", solver_control.tolerance());

  switch (solver_name) {
    case cg:
      //before: solver.SetAztecOption(AZ_solver, AZ_cg);
      newSolver = factory->create("CG", solverParams);
    break;
    case cgs:
      //before: solver.SetAztecOption(AZ_solver, AZ_cgs);
      newSolver = factory->create("Block CG", solverParams);
    break;
    case gmres:
      //before: solver.SetAztecOption(AZ_solver, AZ_gmres);
      //before: solver.SetAztecOption(AZ_kspace,
      ↪  additional_data.gmres_restart_parameter);
      solverParams->set("Maximum Restarts",
      ↪  additional_data.gmres_restart_parameter);
      newSolver = factory->create("GMERS", solverParams);
    break;
    case bicgstab:
      //before: solver.SetAztecOption(AZ_solver, AZ_bicgstab);
      newSolver = factory->create("bicgstab", solverParams);
    break;
    case tfqmr:
      //before: solver.SetAztecOption(AZ_solver, AZ_tfqmr);
      newSolver = factory->create("TFQMR", solverParams);
    break;
    default:
      Assert(false, ExcNotImplemented());
  }
  /*Before:
  if (preconditioner.preconditioner.use_count()!=0){
    ierr = solver.SetPrecOperator(const_cast<Epetra_Operator
↪   *>(preconditioner.preconditioner.get()));
    AssertThrow (ierr == 0, ExcTrilinosError(ierr));
  }
  else
    solver.SetAztecOption(AZ_precond,AZ_none);
  */
  if (preconditioner.preconditioner.use_count() != 0) {
    RCP<Epetra_Operator>MLPrec = rcpFromRef(*(const_cast<Epetra_Operator
    ↪           *>(preconditioner.preconditioner.get())));
    RCP<Belos::EpetraPrecOp> RP = rcp(new Belos::EpetraPrecOp(MLPrec));
    linear_problem->setRightPrec(RP);
  }
  linear_problem->setProblem();
  //before: solver.SetProblem(*linear_problem);
  newSolver->setProblem(linear_problem);
  /* Before:
     ierr = solver.Iterate (solver_control.max_steps(),
↪  solver_control.tolerance());
  */
  Belos::ReturnType result = newSolver->solve();
  /*Before:
   switch (ierr){
      case -1:
      AssertThrow (false, ExcMessage("AztecOO::Iterate error code -1: "
      "option not implemented"));
      case -2:
      AssertThrow (false, ExcMessage("AztecOO::Iterate error code -2: "
      "numerical breakdown"));
      case -3:
      AssertThrow (false, ExcMessage("AztecOO::Iterate error code -3: "
      "loss of precision"));
      case -4:
      AssertThrow (false, ExcMessage("AztecOO::Iterate error code -4: "
      "GMRES Hessenberg ill-conditioned"));
      default:
      AssertThrow (ierr >= 0, ExcTrilinosError(ierr));
   }
  */

  if (result == Belos::Unconverged)
    AssertThrow(false, ExcMessage("Belos::ReturnType Unconverged!"));

  //before: solver_control.check (solver.NumIters(), solver.achievedTol());
  solver_control.check(newSolver->getNumIters(), actTol);

  if (solver_control.last_check() != SolverControl::success)
    AssertThrow(false, SolverControl::NoConvergence(solver_control.last_step(),
      solver_control.last_value()));
}
```

5.3.2 Converting Epetra to Tpetra

Converting Epetra to Tpetra, we not only need to change data types, but also their corresponding functions. Therefore, I create the Table 2 and Table 3 to provide more details. There are few remarks for the tables.

Remark (Arguments)**.** *The tables ignore all function arguments*

Remark (Highlights)**.** *The **blue** functions suggests a direct conversion is missing. The **red** functions suggests that those are the functions with the same name in different structures.*

Remark (Italic Form)**.** *The italic form is only using for `Epetra_FECrsGraph` and `Epetra_FECrsMatrix`, which only implement in Epetra. During conversions, they are converted into `Tpetra::CrsGraph` and `Tpetra::CrsMatrix` respectively.*

Remark (Template Parameters for Trilinos)**.** *The following table are data types specifications for Tpetra Structure.*

Listing 8: Data types for Tpetra Structure

```
typedef double                                           SC;
typedef int                                              LO;
typedef int                                              GO;
typedef KokkosClassic::DefaultNode::DefaultNodeType     NT;
```

The followling table is convension table for the data types.[3][4][10]

Structure	Epetra Structure	Tpetra Structure
Maps	Epetra_Map Epetra_BlockMap	Tpetra::Map<SC, LO, GO>
	Epetra_LocalMap	Tpetra::LocalMap<SC, LO, GO>
Export	Epetra_Export	Tpetra::Export< SC, LO, GO >
Import	Epetra_Import	Tpetra::Import<SC, LO, GO>
Operator	Epetra_Operator	Tpetra::Operator<SC, LO, GO, NT>
Matrices	Epetra_CrsMatrix Epetra_FECrsMartix	Tpetra::CrsMatrix<SC, LO, GO, NT>
	Epetra_RowMatrix	Tpetra::RowMatrix<SC, LO, GO, NT>
Graphs	Epetra_CrsGraph Epetra_FECrsGraph	Tpetra::CrsGraph<SC, LO, GO, NT>
Vectors	Epetra_Vector	Tpetra::Vector<SC, LO, GO, NT>
	Epetra_MultiVector	Tpetra::MultiVector<SC, LO, GO, NT>
MPI	Epetra_Comm	Teuchos::RCP<Teuchos::Comm<int>>
	Epetra_MpiComm	Tpetra::MpiPlatform<int>
	Epetra_SerialComm	Tpetra::SerialPlatform<int>

Table 2: *Epetra* to *Tpetra* Data Type Conversion

The followling table is the convension table for functions. [3][4][10]

E/Tpetra Structure	Epetra Function	Tpetra Function
Map BlockMap	MinAllGID()	getMinAllGlobalIndex ()
	MaxAllGID()	getMaxAllGlobalIndex ()
	MinMyGID()	getMinGlobalIndex ()
	MaxMyGID()	getMaxGlobalIndex ()
	MinLID()	getMinLocalIndex()
	MaxLID()	getMaxLocalIndex ()
	NumMyElements()	getNodeNumElements ()
	NumGlobalElements()	getGlobalNumElements ()
	IndexBase()	getIndexBase()
	DistributedGlobal()	isDistributed()
	LinearMap()	isContiguous ()
	MyGlobalElements()	getMyGlobalIndices ()
	MyGID()	isNodeGlobalElement()
	MyLID()	isNodeLocalElement()
	LID()	getLocalElement()
	GID()	getGlobalElement()
	IsOneToOne()	isOnetoOne()
	UniqueGIDs()	
CrsGraph CrsMatrix *FECrsGraph* *FECrsMatrix*	RangeMap()	getRangeMap()
	RowMap()	getRowMap()
	DomainMap()	getDomainMap()
	GlobalLength()	getGlobalLength()
	ColMap()	getColMap()
	graph()	getCrsGraph()
	FillComplete()	fillComplete()
	Filled()	isFillComplete()
	IndexBase()	getIndexBase ()
	MaxNumIndices()	getNodeMaxNumRowEntries()
	NumMyRows ()	getNodeNumRows ()
	NumMyIndices()	getNumEntriesInLocalRow()
	NumGlobalCols ()	getGlobalNumCols()
	NumGlobalIndices()	getNumEntriesInGlobalRow ()
	GlobalMaxNumIndices()	getGlobalMaxNumRowEntries()
	ReplaceGlobalValues()	replaceGlobalValues()
	OperatorRangeMap()	getRangeMap()
	OperatorDomainMap()	getDomainMap()
	MaxNumEntries()	getNodeMaxNumRowEntries()
	NumMyEntries()	getNumEntriesInLocalRow()
	LRID()	
	GRID()	
FECrs_Graph *FECrs_Matrix*	InsertGlobalValues()	insertGlobalValues()
	GlobalAssmeble()	
Vector MultiVector	PutScalar()	putScalar()
	Map()	getMap()
	Norm1()	norm1()
	Norm2()	norm2()
	MeanValue()	meanValue()
	Update()	update()

	PutScalar()
	MinValue()
	MaxValue()
	ExtractView()
		putScalar()

Operator	OperatorRangeMap()
	OperatorDomainMap()
	Apply()
	ApplyInverse()
		getRangeMap()
		getDomainMap()
		Apply()
		Apply()

Solver	reset()
	Iterate()
	NumIters()
	TrueResidual()
		setProblem()
		solve()
		getNumIters()
		achievedTol()

Table 3: Epetra to Tpetra Functions Conversion

6 Results

6.1 One-Dimensional Boussinesq Equation Result

In section 5.1 we have all the parameters for $H(\eta)$ which defined in (5.32) as

$$H(\eta) = v - \frac{\eta_k}{2}(\eta - \eta_k) + A(\eta - \eta_k)^{\lambda}, \text{where } \eta = \frac{x}{[(Kh_0/S)t]^{1/2}} \tag{6.1}$$

Then we need to find all the parameters by iterations. Thus, iterated through (5.44), we have the following table:[5]

H	λ	v	η_k	μ	A	L
H_1	2.1000	1.2571	0.2923	1.2996	0.0184	3.6498
H_2	2.2000	1.5283	0.5931	1.7010	0.0326	3.8590
H_3	2.3000	1.8135	0.9033	2.2082	0.0435	4.0898
H_4	2.4000	2.1127	1.2240	2.8258	0.0519	4.3408

Table 4: Values of the Parameters for $1 < \mu < 3$.[12]

Then plot the $H(\eta)$ by those parameters in Table 4 we have

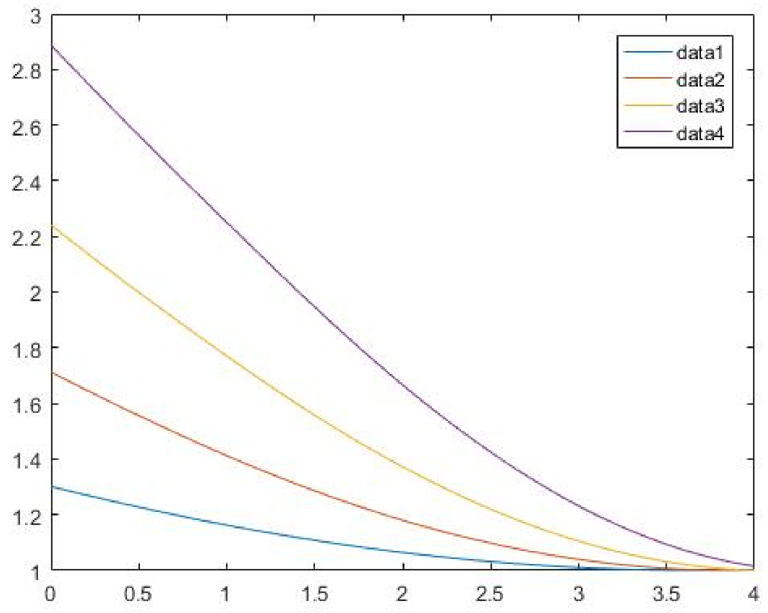

Figure 3: Piezometric H versus η for various values of μ.[12]

The following Figure 4, 5 and 6 are representing numerical results for all the basis parameters v, η_k, λ, A,and$L$ versus μ, $1 < \mu < 3$.

[5]The code for iterations is listed in Section 8.4

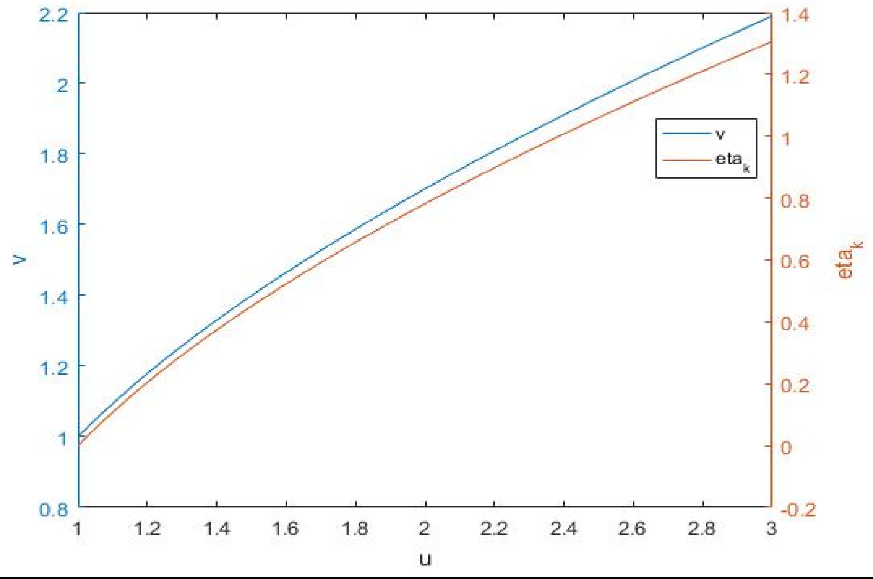

Figure 4: Values of the parameters v and η_k versus μ.[12]

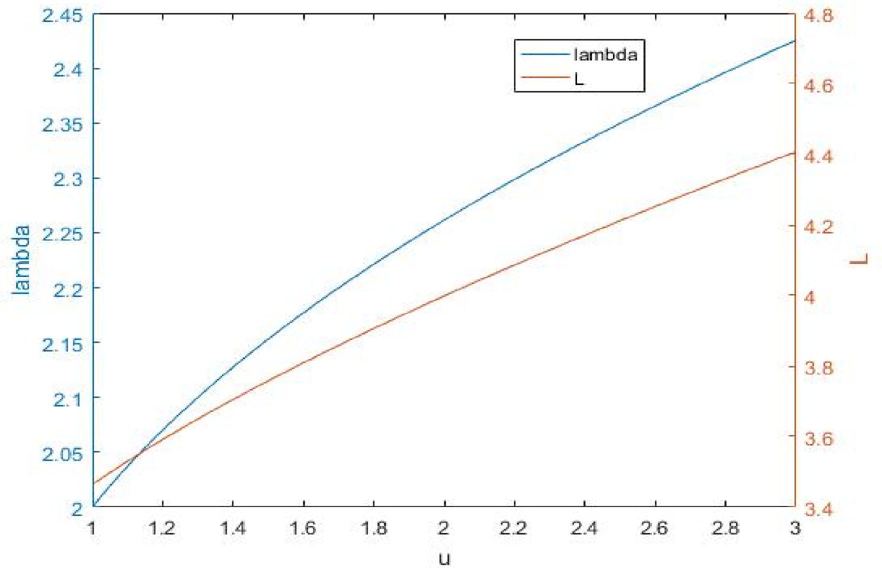

Figure 5: Values of the parameters λ and L versus μ.[12]

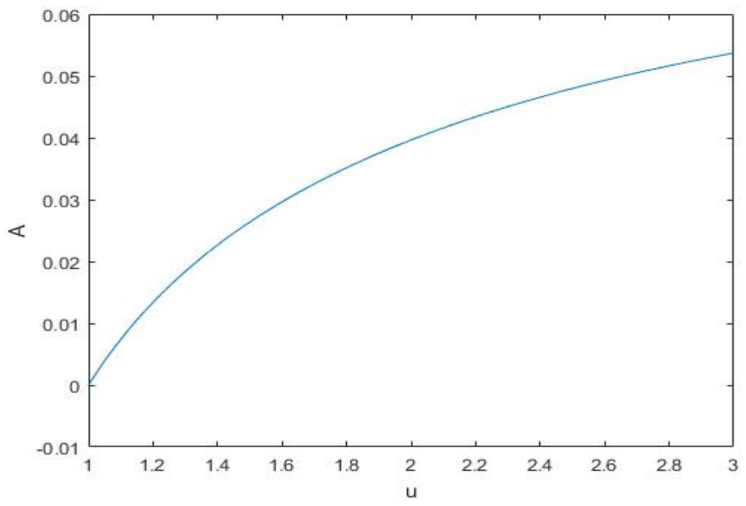

Figure 6: Values of the parameters λ and A versus μ.[12]

6.2 Prototypes Comparison

6.2.1 Number of Iterations Comparison

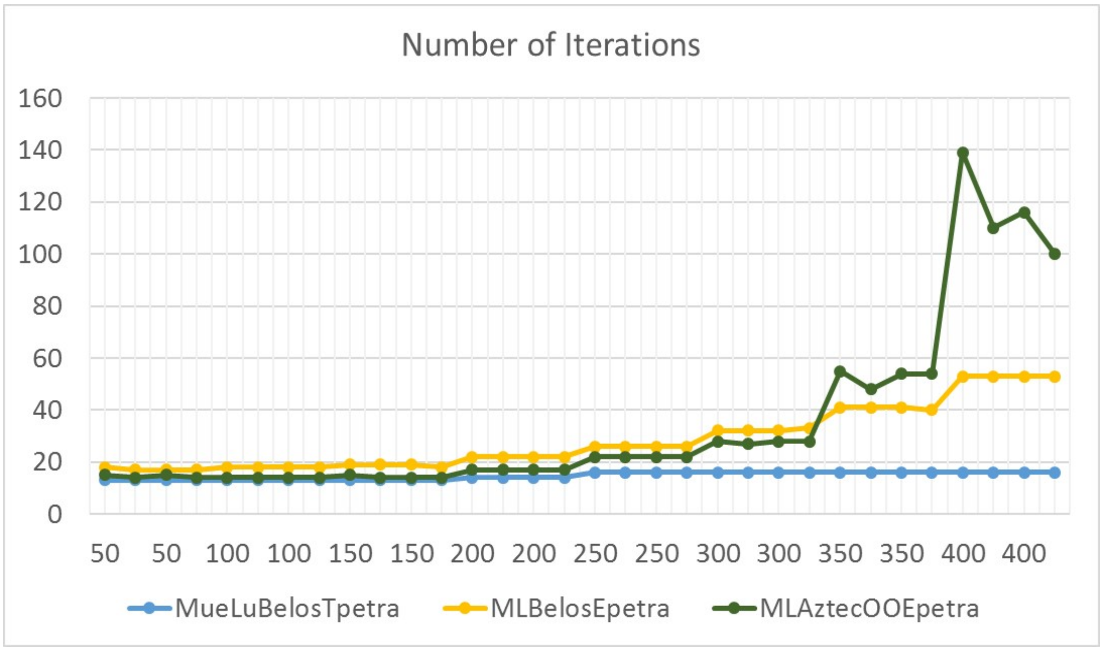

Figure 7: A line chart of all trials for *MLAztecOOEpetra.cc*, *MLBelosEpetra.cc*, and *MueLuBelosTpetra.cc* with problem sizes from 50^3 to 400^3. This plot shows the range in number of iterations for each file.

6.2.2 Solving Time Comparison

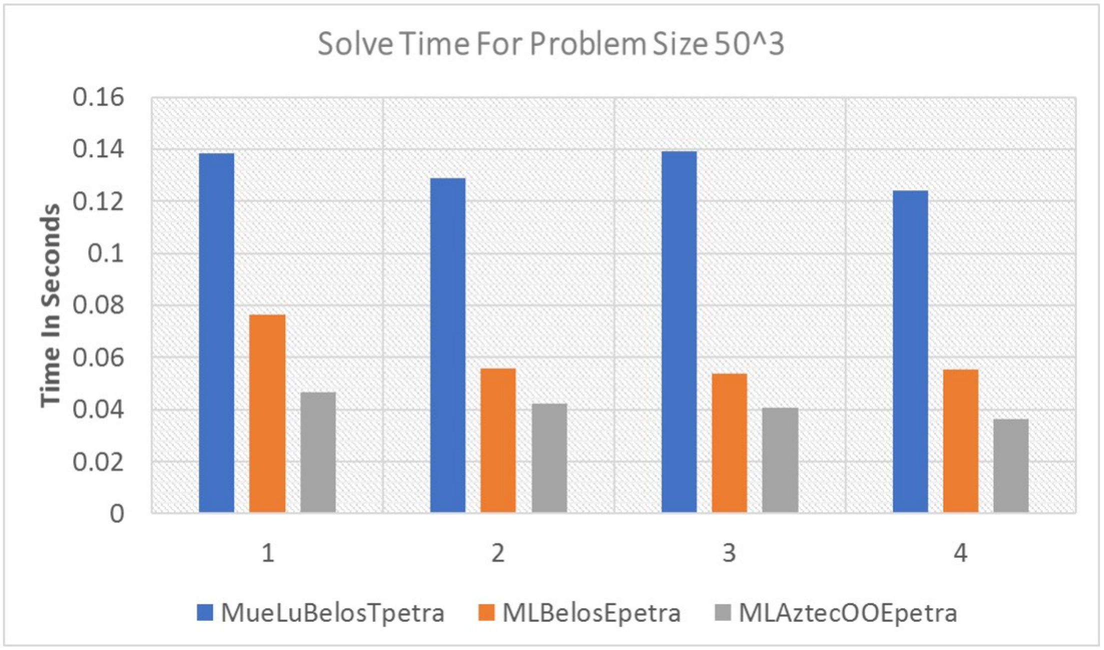

Figure 8: A column chart of 4 time trials for *MLAztecOOEpetra.cc*, *ML-BelosEpetra.cc*, and *MueLuBelosTpetra.cc* with a problem size of 50^3.

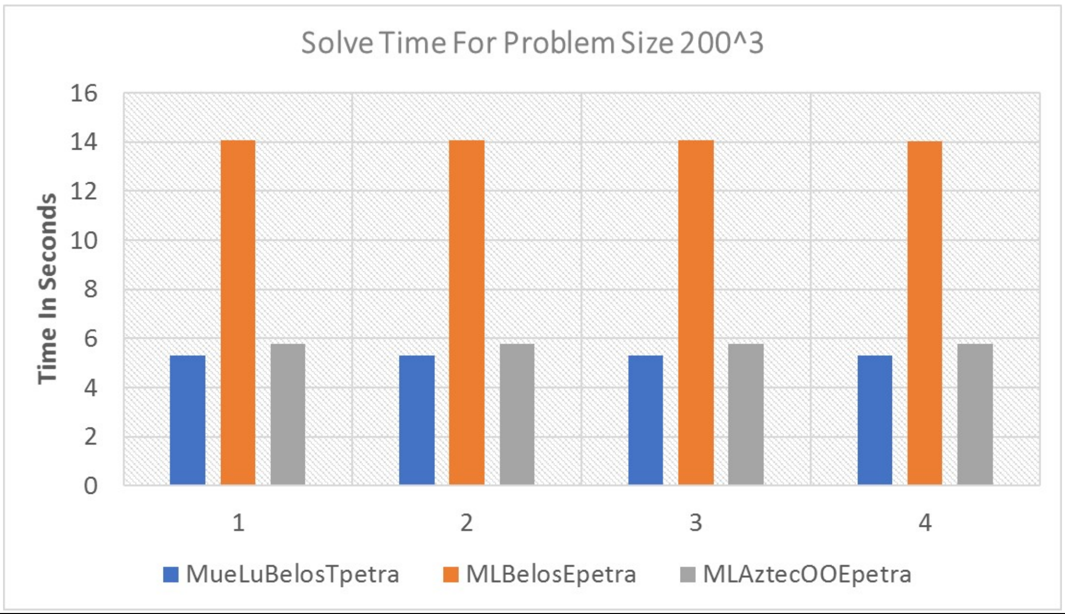

Figure 9: A column chart of 4 time trials for *MLAztecOOEpetra.cc*, *ML-BelosEpetra.cc*, and *MueLuBelosTpetra.cc* with a problem size of 200^3.

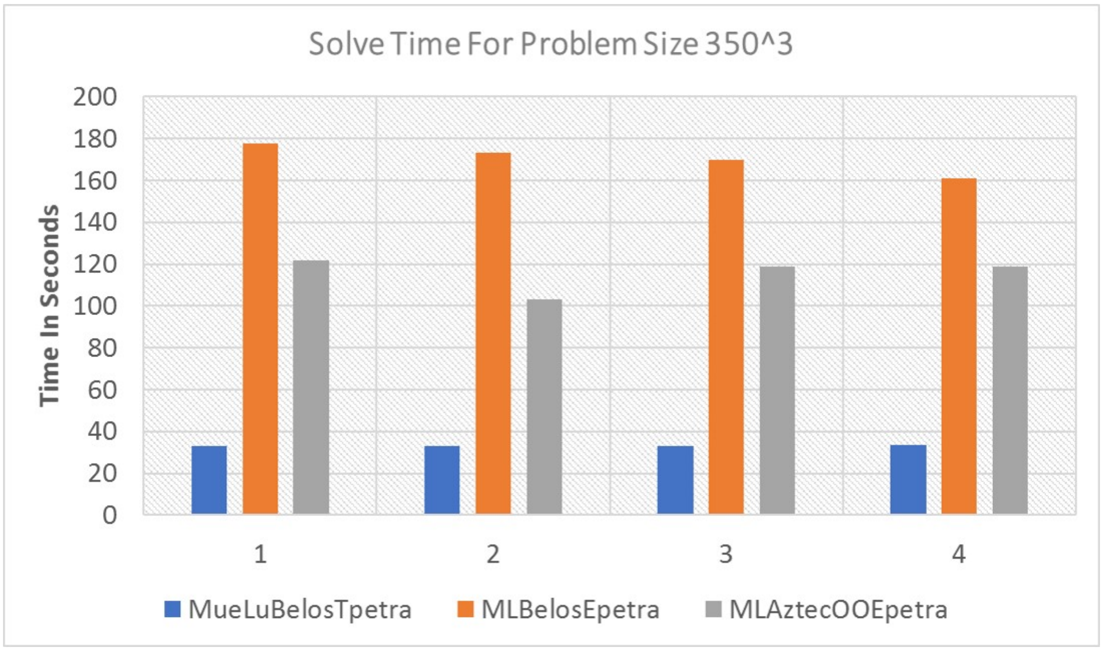

Figure 10: A column chart of 4 time trials for ***MLAztecOOEpetra.cc***, ***ML-BelosEpetra.cc***, and ***MueLuBelosTpetra.cc*** with a problem size of 350^3.

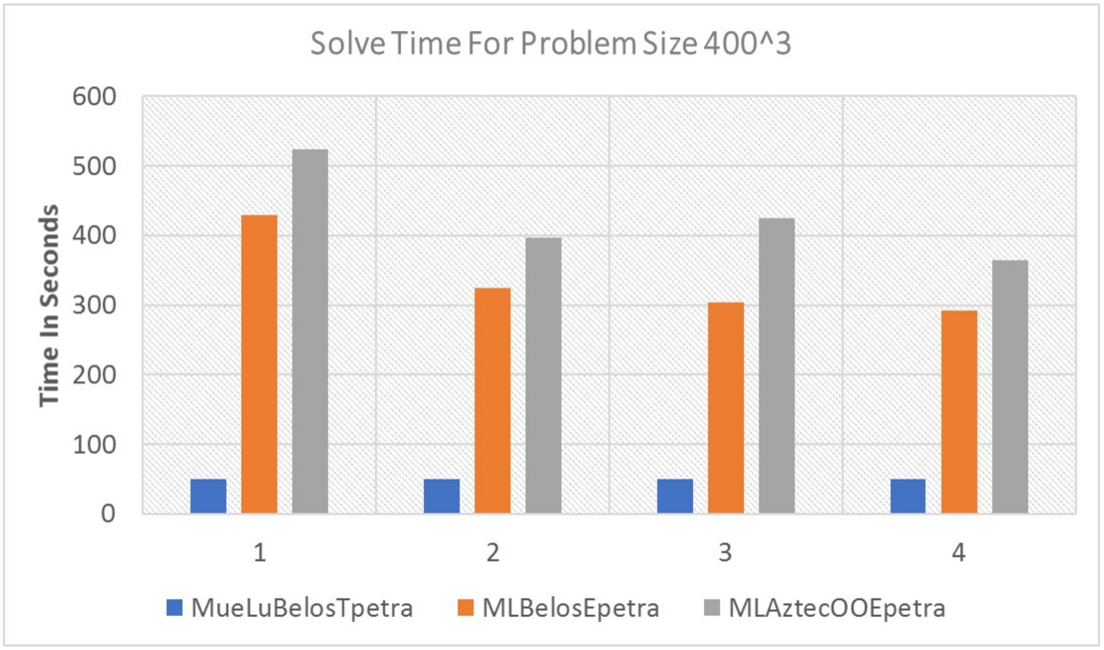

Figure 11: A column chart of all trials for ***MLAztecOOEpetra.cc***, ***ML-BelosEpetra.cc***, and ***MueLuBelosTpetra.cc*** with a problem size from 50^3 to 400^3.

6.3 Comparison of Trilinos Linear Solvers in deal.ii

6.3.1 Solving time Comparison

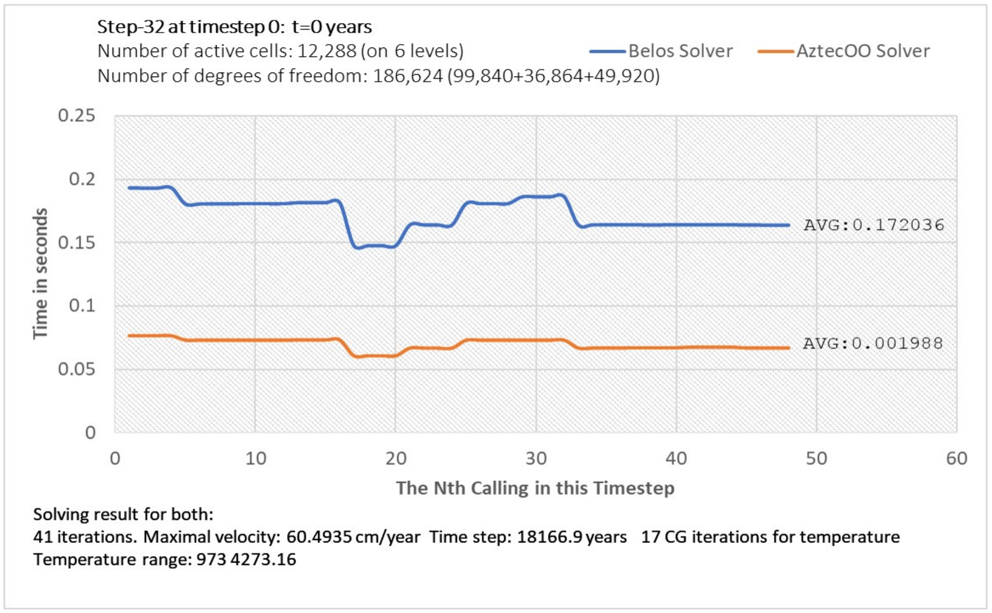

Figure 12: A scatter chart at Timestep 0 for *step-32.cc*. The measured time is solving time. This plot shows the range in time performance for *Belos* and *AztecOO* solver. The solving results listed at the bottom of this chart have the same value for both solvers.

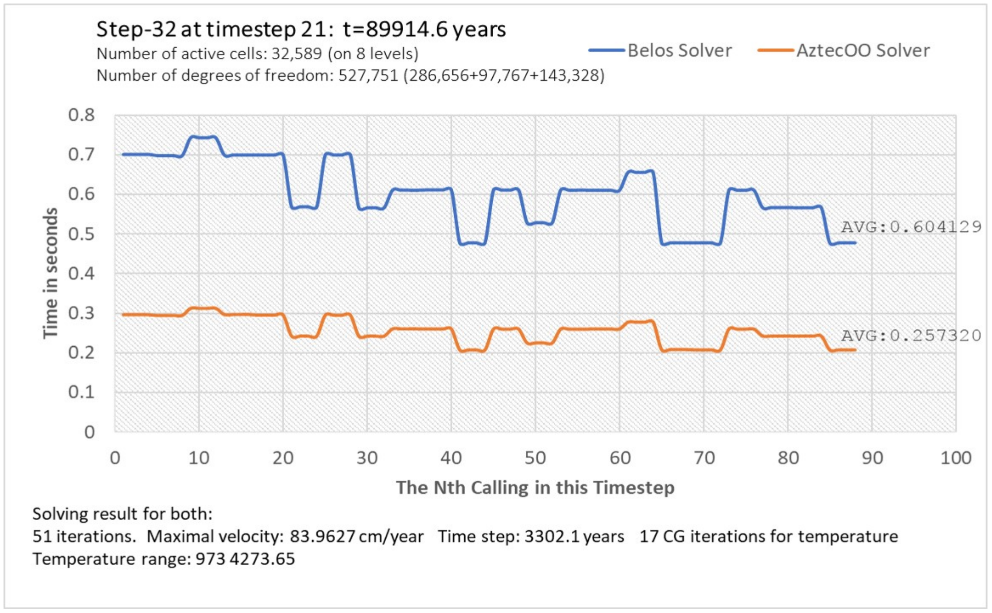

Figure 13: A scatter chart at Timestep 21 for *step-32.cc*. The measured time is solving time. This plot shows the range in time performance for *Belos* and *AztecOO* solver. The solving results listed at the bottom of this chart have the same value for both solvers.

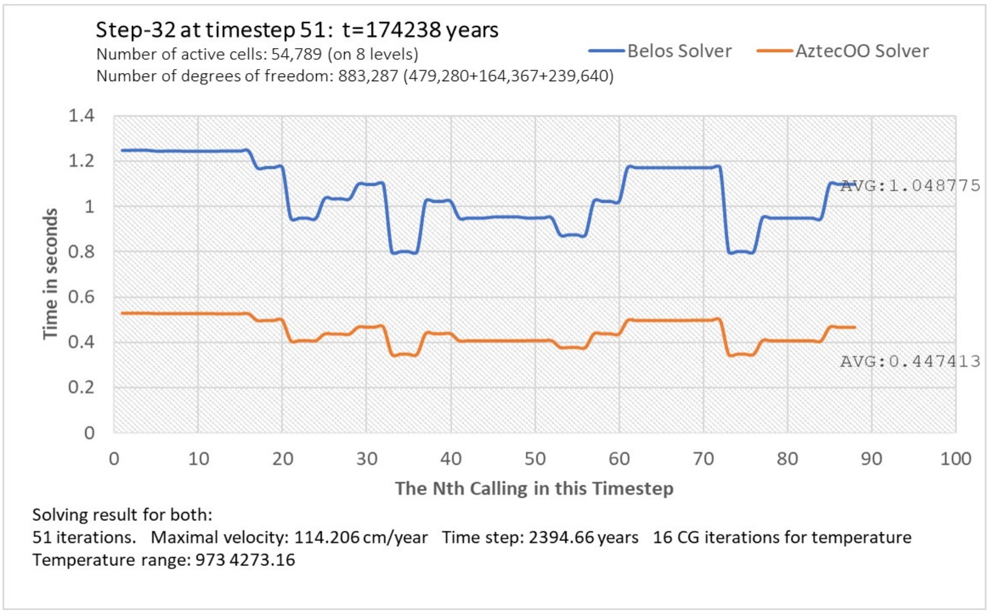

Figure 14: A scatter chart at Timestep 51 for *step-32.cc*. The measured time is solving time. This plot shows the range in time performance for *Belos* and *AztecOO* solver. The solving results listed at the bottom of this chart have the same value for both solvers.

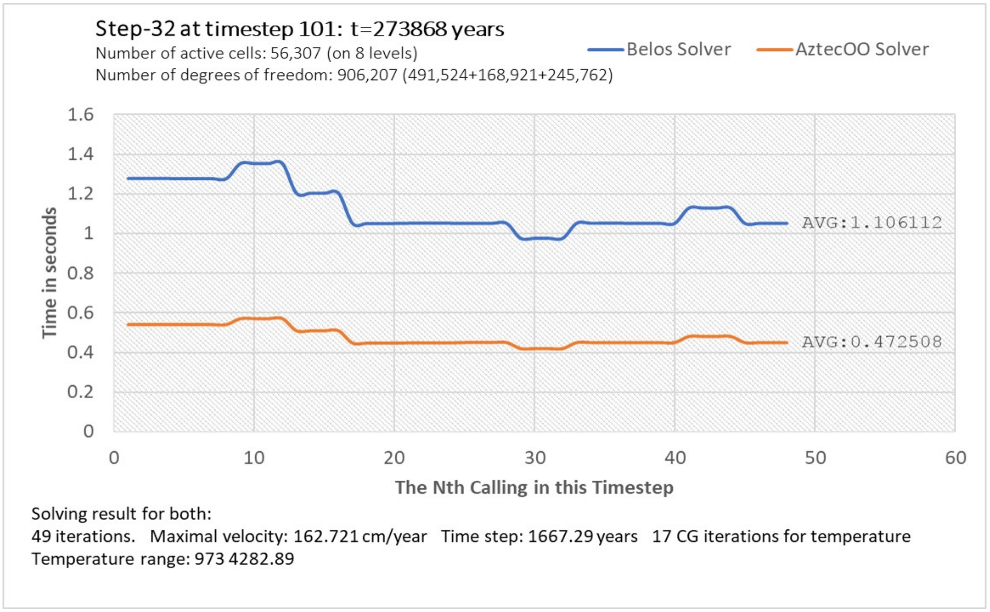

Figure 15: A scatter chart at Timestep 101 for *step-32.cc*. The measured time is solving time. This plot shows the range in time performance for *Belos* and *AztecOO* solver. The solving results listed at the bottom of this chart have the same value for both solvers.

6.3.2 Number of Solver Iteration Comparison

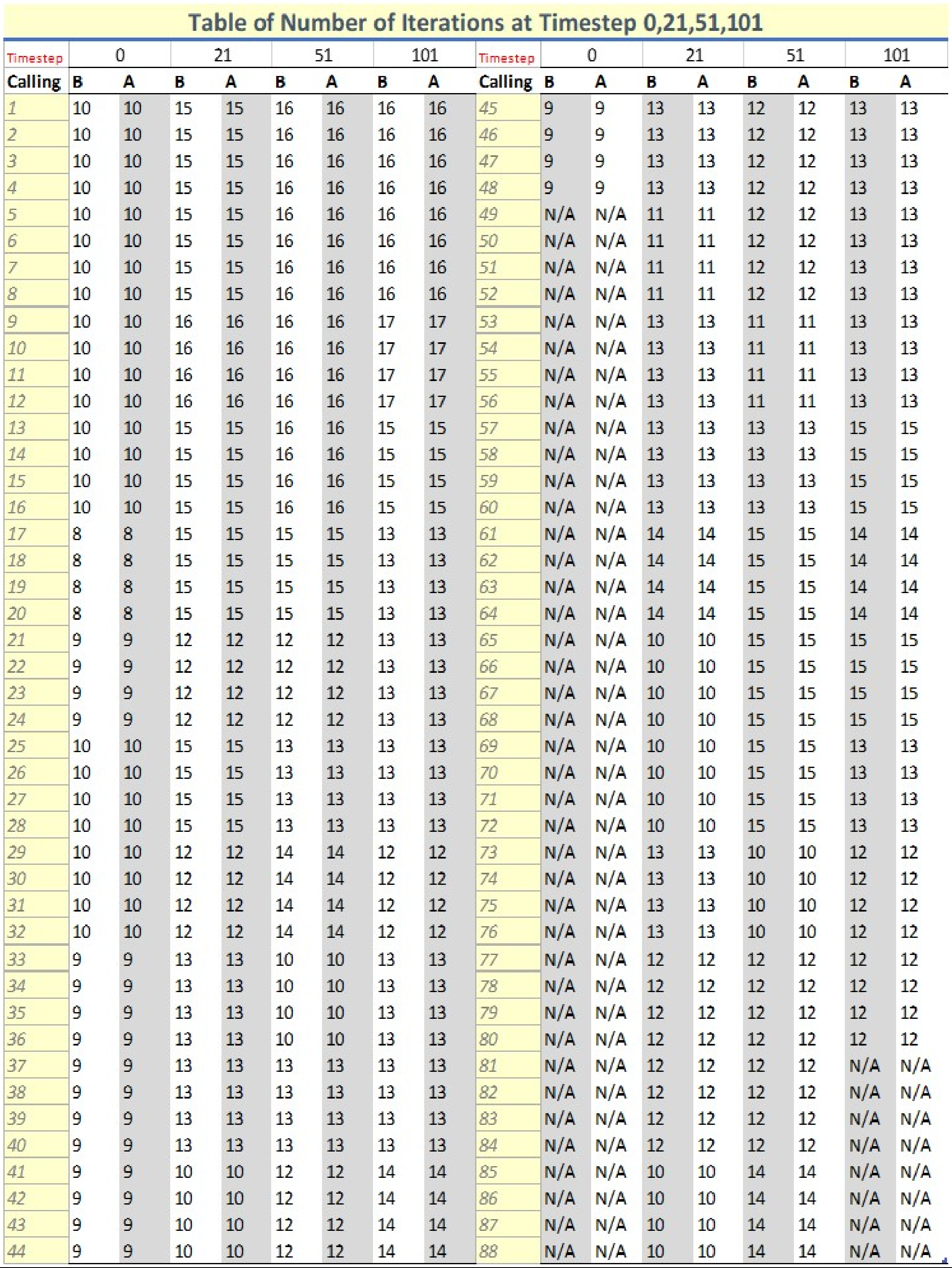

Table of Number of Iterations at Timestep 0,21,51,101

Timestep	0		21		51		101	
Calling	B	A	B	A	B	A	B	A
1	10	10	15	15	16	16	16	16
2	10	10	15	15	16	16	16	16
3	10	10	15	15	16	16	16	16
4	10	10	15	15	16	16	16	16
5	10	10	15	15	16	16	16	16
6	10	10	15	15	16	16	16	16
7	10	10	15	15	16	16	16	16
8	10	10	15	15	16	16	16	16
9	10	10	16	16	16	16	17	17
10	10	10	16	16	16	16	17	17
11	10	10	16	16	16	16	17	17
12	10	10	16	16	16	16	17	17
13	10	10	15	15	16	16	15	15
14	10	10	15	15	16	16	15	15
15	10	10	15	15	16	16	15	15
16	10	10	15	15	16	16	15	15
17	8	8	15	15	15	15	13	13
18	8	8	15	15	15	15	13	13
19	8	8	15	15	15	15	13	13
20	8	8	15	15	15	15	13	13
21	9	9	12	12	12	12	13	13
22	9	9	12	12	12	12	13	13
23	9	9	12	12	12	12	13	13
24	9	9	12	12	12	12	13	13
25	10	10	15	15	13	13	13	13
26	10	10	15	15	13	13	13	13
27	10	10	15	15	13	13	13	13
28	10	10	15	15	13	13	13	13
29	10	10	12	12	14	14	12	12
30	10	10	12	12	14	14	12	12
31	10	10	12	12	14	14	12	12
32	10	10	12	12	14	14	12	12
33	9	9	13	13	10	10	13	13
34	9	9	13	13	10	10	13	13
35	9	9	13	13	10	10	13	13
36	9	9	13	13	10	10	13	13
37	9	9	13	13	13	13	13	13
38	9	9	13	13	13	13	13	13
39	9	9	13	13	13	13	13	13
40	9	9	13	13	13	13	13	13
41	9	9	10	10	12	12	14	14
42	9	9	10	10	12	12	14	14
43	9	9	10	10	12	12	14	14
44	9	9	10	10	12	12	14	14
45	9	9	13	13	12	12	13	13
46	9	9	13	13	12	12	13	13
47	9	9	13	13	12	12	13	13
48	9	9	13	13	12	12	13	13
49	N/A	N/A	11	11	12	12	13	13
50	N/A	N/A	11	11	12	12	13	13
51	N/A	N/A	11	11	12	12	13	13
52	N/A	N/A	11	11	12	12	13	13
53	N/A	N/A	13	13	11	11	13	13
54	N/A	N/A	13	13	11	11	13	13
55	N/A	N/A	13	13	11	11	13	13
56	N/A	N/A	13	13	11	11	13	13
57	N/A	N/A	13	13	13	13	15	15
58	N/A	N/A	13	13	13	13	15	15
59	N/A	N/A	13	13	13	13	15	15
60	N/A	N/A	13	13	13	13	15	15
61	N/A	N/A	14	14	15	15	14	14
62	N/A	N/A	14	14	15	15	14	14
63	N/A	N/A	14	14	15	15	14	14
64	N/A	N/A	14	14	15	15	14	14
65	N/A	N/A	10	10	15	15	15	15
66	N/A	N/A	10	10	15	15	15	15
67	N/A	N/A	10	10	15	15	15	15
68	N/A	N/A	10	10	15	15	15	15
69	N/A	N/A	10	10	15	15	13	13
70	N/A	N/A	10	10	15	15	13	13
71	N/A	N/A	10	10	15	15	13	13
72	N/A	N/A	10	10	15	15	13	13
73	N/A	N/A	13	13	10	10	12	12
74	N/A	N/A	13	13	10	10	12	12
75	N/A	N/A	13	13	10	10	12	12
76	N/A	N/A	13	13	10	10	12	12
77	N/A	N/A	12	12	12	12	12	12
78	N/A	N/A	12	12	12	12	12	12
79	N/A	N/A	12	12	12	12	12	12
80	N/A	N/A	12	12	12	12	12	12
81	N/A	N/A	12	12	12	12	N/A	N/A
82	N/A	N/A	12	12	12	12	N/A	N/A
83	N/A	N/A	12	12	12	12	N/A	N/A
84	N/A	N/A	12	12	12	12	N/A	N/A
85	N/A	N/A	10	10	14	14	N/A	N/A
86	N/A	N/A	10	10	14	14	N/A	N/A
87	N/A	N/A	10	10	14	14	N/A	N/A
88	N/A	N/A	10	10	14	14	N/A	N/A

Figure 16: A table lists the number of iterations where 'B' stands for 'Belos' and 'A' stands for 'AztecOO'. 'Calling' represents the n^{th} time of calling `BlockSchurPreconditioner::vmult` function in *step-32.cc*.

7 Conclusion

Overall, this project demonstrates the analytical solution of an one-dimensional Boussinesq equation, a workable prototype for updating deal.ii, a new solver for deal.ii and a conversion table for converting Epetra objects to Tpetra objects.

7.1 Performance of Prototypes

The results from Section 6.2 confirms that using MueLu as the preconditioner, Belos as solver and Tpetra as basic mathematical structure result in shorter solving time and a smaller number of iterations. A real world problem that deals.ii solves can potentially have the large size of matrices and vectors. Thus, I believe the results from prototypes suggest that improvement can be expected after successfully updating deal.ii as proposed.

Noticeably, the intermediate prototype, *MLEpetraBelos*, does not result in faster solving time with smaller sizes of the problems. Thus, the same results apply to the current updated solver version of deal.ii-8.4.1. Since I have only updated the solver, and 'step-32' uses ML as preconditioner, the program result in a longer runtime with such a combination.

7.2 Performance of Updated deal.ii

The results from Section 6.3 reflect Section 6.2, that ML as preconditioner, Belos as solver with Epetra do not produce shorter solving time and a smaller number of iterations. Actually, the table in Figure 16 suggests that there is no difference in the number of iterations at Timestep 0,21,51, and 101. Since this version of deal.ii only updated its Trilinos solver, based on the results from 6.3, I believe that improvement can be expected after a complete update.

7.3 Future Work

Currently, this project finishes updating the deal.ii Trilinos solver. The next phase of converting *Epetra* to *Tpetra* is still ongoing. Also, the conversion table is not entirely complete, which means that there is still much to do.

For starters, the conversion tables are not yet completed, maintaining the conversion table is critical for the future of this project. The table needs regular updates since Trilinos packages are constantly renewed. Also, it may be possible to encapsulate some Xpetra objects into the project because Xpetra objects are more friendly when working with MueLu.

8 Appendix

8.1 Source Code

8.1.1 Sourc Code for Prototype

The complete source code of the prototypes can be found at the git repository `https://github.com/s1lin/ug_thesis_prototypes` or clone as `https://github.com/s1lin/ug_thesis_prototypes.git`.

8.1.2 Source Code for Updated deal.ii

The source code of the ongoing updating process of deal.ii-8.4.1 can be found at the git repository `https://github.com/s1lin/deal.ii-8.4.1-Tpetra.git` or clone as `https://github.com/s1lin/deal.ii-8.4.1-Tpetra.git`.
The complete source code for deal.ii with 'Belos' solver can be checked out as a branch.

8.2 Computing Specifications

In this project, the computing node has a *Dual Intel Xeon CPU E5-2420 Sandy Bridge @ 1.90GHz 15MB L3 Cache 95W Six-Core* (12 cores per node, with 2 HW thread states per core). The following are the detailed specifications.

Listing 9: Specification

```
vendor_id       : GenuineIntel
cpu family      : 6
model     : 45
model name      : Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz
stepping        : 7
microcode       : 0x710
cpu                   : 1201.156
cache size      : 15360 KB
physical id     : 1
siblings        : 12
core id         : 5
cpu cores       : 6
apicid    : 43
initial apicid  : 43
fpu       : yes
fpu_exception         : yes
cpuid level     : 13
wp        : yes
flags     : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
 ↪  pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
 ↪  rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
 ↪  nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx
 ↪  smx est tm2 ssse3 cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic popcnt
 ↪  tsc_deadline_timer aes xsave avx lahf_lm ida arat epb pln pts dtherm
 ↪  tpr_shadow vnmi flexpriority ept vpid xsaveopt
```

```
bogomips        : 3796.05
clflush size    : 64
cache_alignment     : 64
address sizes       : 46 bits physical, 48 bits virtual
```

8.3 Prototypes Detailed Comparison Table

Remark. *In following table 'I' stands for number of iterations, 'size' stands for the problem size, and 'Residual' stands for the result residual. The time is measuring in seconds.The actual problem size is the cube power to the number in the 'size' column.*

	MueLuBelosTpetra			**MLBelosEpetra**			**MLAztecOOEpetra**		
Size	**Time**	**I**	**Residual**	**Time**	**I**	**Residual**	**Time**	**I**	**Residual**
50	0.13832	13	1.12E-07	0.07621	18	5.34E-07	0.05029	15	9.88E-07
50	0.12876	13	1.12E-07	0.05581	17	1.40E-07	0.04205	14	6.E-07
50	0.13895	13	1.12E-07	0.05359	17	8.64E-08	0.04043	15	7.01E-07
50	0.12423	13	1.12E-07	0.05536	17	1.74E-07	0.03625	14	1.25E-06
100	0.56452	13	1.17E-05	1.27561	18	1.13E-05	0.42046	14	1.28E-05
100	0.57198	13	1.17E-05	0.96262	18	3.89E-05	0.41381	14	1.72E-05
100	0.56596	13	1.17E-05	0.95775	18	2.53E-05	0.42866	14	2.10E-05
100	0.55718	13	1.17E-05	0.96385	18	9.40E-06	0.40586	14	1.75E-05
150	2.03450	13	5.53E-05	4.41518	19	3.00E-04	1.93984	15	2.37E-04
150	2.00938	13	5.53E-05	4.41267	19	1.43E-04	1.79286	14	1.26E-04
150	2.01293	13	5.53E-05	4.41494	19	1.38E-04	1.79128	14	1.07E-04
150	2.02474	13	5.53E-05	4.09582	18	3.62E-05	1.76792	14	9.00E-05
200	5.30855	14	1.81E-03	14.0855	22	6.98E-04	5.77289	17	5.17E-04
200	5.30083	14	1.81E-03	14.0809	22	3.46E-04	5.79151	17	1.42E-03
200	5.29882	14	1.81E-03	14.0751	22	9.63E-04	5.76752	17	8.37E-03
200	5.28961	14	1.81E-03	14.0423	22	3.83E-04	5.77199	17	1.62E-03
250	12.0674	16	4.28E-03	37.7575	26	3.12E-03	16.0191	22	6.94E-03
250	12.0437	16	4.28E-03	37.6013	26	7.63E-03	16.0310	22	4.58E-03
250	12.0514	16	4.28E-03	37.5571	26	3.57E-03	16.0286	22	1.81E-02
250	12.0406	16	4.28E-03	37.5510	26	6.68E-03	16.0724	22	2.51E-03
300	20.7442	16	4.28E-03	82.4861	32	5.23E-03	39.5045	28	4.97E-03
300	20.7682	16	8.02E-03	85.9746	32	4.83E-03	37.5331	27	1.08E-02
300	20.7546	16	8.02E-03	87.8332	32	8.47E-03	39.4962	28	9.38E-03
300	20.7720	16	8.02E-03	89.5426	33	1.54E-03	39.3823	28	1.11E-02
350	33.0793	16	2.52E-02	177.653	41	1.67E-02	121.813	55	1.14E-02
350	32.8929	16	2.52E-02	173.226	41	1.62E-02	103.102	48	1.03E-02
350	33.1230	16	2.52E-02	169.977	41	2.80E-02	119.006	54	2.19E-03
350	33.4639	16	2.52E-02	161.084	40	2.70E-02	119.055	54	9.74E-03
400	50.0645	16	6.24E-02	429.628	53	4.38E-02	523.259	139	6.67E-02
400	49.9835	16	6.24E-02	324.990	53	3.14E-02	396.836	110	5.95E-03
400	49.6237	16	6.24E-02	303.448	53	3.14E-02	424.181	116	2.30E-02
400	49.4960	16	6.24E-02	291.795	53	3.46E-02	363.891	100	1.21E-02

Table 5: Prototypes Detailed Comparison Table

8.4 Code for Iteration

Listing 10: Iteration.m

```
function[u,v,nk,l,L] = Solve()
  l(1) = 2;
  m = 2;
  index = 1;
  tol = 1e-6;
  while(m<3)
    l(index) = m;
    D1 = 8*m/(m-1)-4*m*m/(m-1)^2+(8*m*m/(m-1)^2)*(1/(m*(m+1)));
    D3 = 2/sqrt(D1);
    D4 = D3*2*m/(m-1);
    D2 = D3*D4^2-D4^3/3+D4^3/(m*(m+2));
    nk(index) = (D2 - 2*D3)*(-1+D3^2)^(-1);
    v(index)  = D3*nk(index)+1;
    L(index)  = D3*2*m/(m-1) + nk(index);
    A(index)  = nk(index)/(2*m*(L(index)-nk(index))^(m-1));
    b  = -nk(index)^2;
    delta = b^2-12*(-3*v(index)^2-2*v(index)*nk(index)^2);
    u(index) = (-b + sqrt(delta))/6;
    m = l(index) + 1e-8;
    index = index + 1;
  end
end
```

8.5 Installation Notes and Scripts

8.5.1 The Trilinos Packages

The following is the CMake script to install Trilinos. [6] The lastest Trilinos version is 12.10, which can be downloaded from the public Github repository. [7]

Listing 11: Trilinos Installation and CMake Script

```
mkdir trilinos_build
cd trilinos_build

cmake                                        \
-DTrilinos_ENABLE_ALL_OPTIONAL_PACKAGES=OFF  \
-DTrilinos_ENABLE_Amesos=ON                  \
-DTrilinos_ENABLE_Epetra=ON                  \
-DTrilinos_ENABLE_Tpetra=ON                  \
-DTrilinos_ENABLE_Xpetra=ON                  \
-DTrilinos_ENABLE_EpetraExt=ON               \
```

[6] To successfully install Trilinos with MPI gFortran is required. The installation guide is listed in Section 8.5.3

[7] To get Trilinos, clone from `https://github.com/trilinos/Trilinos.git`

```
-DTrilinos_ENABLE_Ifpack=ON                         \
-DTrilinos_ENABLE_Ifpack2=ON                        \
-DTrilinos_ENABLE_AztecOO=ON                        \
-DTrilinos_ENABLE_Sacado=ON                         \
-DTrilinos_ENABLE_Teuchos=ON                        \
-DTrilinos_ENABLE_MueLu=ON                          \
-DTrilinos_ENABLE_Belos=ON                          \
-DTrilinos_ENABLE_ML=ON                             \
-DTrilinos_ENABLE_Kokkos=ON                         \
-DTrilinos_ENABLE_Teuchos=ON                        \
-DTrilinos_ENABLE_Galeri=ON                         \
-DTPL_ENABLE_MPI=ON                                 \
-DBUILD_SHARED_LIBS=ON                              \
-DCMAKE_BUILD_TYPE=RELEASE                          \
-DCMAKE_INSTALL_PREFIX:PATH=/path/to/install/dir    \
/path/to/source/folder

make -j 16 install
```

8.5.2 The deal.ii Library

The following is a CMake script to install the deal.ii, version 8.4.1 [8].

Listing 12: deal.ii Installation and CMake Script

```
mkdir dealii_build
cd dealii_build

cmake \
-DDEAL_II_WITH_MPI=ON              \s
-DDEAL_II_WITH_TRILINOS=ON            \
-DDEAL_II_WITH_P4EST=ON             \
-DTRILINOS_DIR=/path/to/trilinos          \
-DP4EST_DIR=/path/to/p4est            \
-DCMAKE_INSTALL_PREFIX=/path/to/install/dir    \
/path/to/source/folder

make -j 16 install
```

8.5.3 Project Required Libraries

1. **gFortran**

[8]Installation of Trilinos are required in reproducing this project and should be installed prior to deal.ii installation.

2. **BLAS, LAPACK** [9]

3. **P4EST** [10]

[9]BLAS: Basic Linear Algebra Subprograms and LAPACK: Linear Algebra PACKage

[10]The P4EST enables the dynamic management of a collection of adaptive octrees, conveniently called a forest of octrees. `http://www.p4est.org`

References

[1] Belos: An iterative linear solvers package. `https://trilinos.org/docs/dev/packages/belos/doc/html/index.html`.

[2] Boussinesq approximation (water waves). `https://en.wikipedia.org/wiki/Boussinesq_approximation_(water_waves)`.

[3] Epetra. `https://trilinos.org/packages/tpetra/`.

[4] Tpetra. `https://trilinos.org/packages/epetra/`.

[5] Trilinos. `https://en.wikipedia.org/wiki/Trilinos`.

[6] What is deal.ii? `https://www.dealii.org/about.html`.

[7] Xpetra. `https://trilinos.org/packages/xpetra/`.

[8] Emily Furst. Parallel preconditioners for finite element computations. (66).

[9] Jonathan Hu. Muelu examples. `https://github.com/trilinos/Trilinos/blob/master/packages/muelu/example/basic/Simple.cpp`.

[10] James M. Willenbring Michael A. Heroux, David M. Day. *Trilinos Tutorial*. Sandia National Laboratories.

[11] Genilhs Myson. Introduction to mpi and openmp. `http://www.ee.ryerson.ca/~courses/ee8218/mpi_openmp.pdf`.

[12] Christos D. Tzimopoulos Panagiotis K. Tolikas, Epaminondas G. Sidiropoulos. A simple analytical solution for the boussinesq one-dimensional groundwater flow equation. *Water Resources Researchs*, 20(1):24–28.