THE UNIVERSITY OF CHICAGO

CRITICAL POINTS IN SIMULATION TECHNOLOGY

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY

HANNAH MORGAN

CHICAGO, ILLINOIS

JUNE 2018

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Special thanks to Ridgway Scott, Matt Knepley, Patrick Sanan, Oana Marin, Karl Rupp, Vivak Patel, Barry Smith, and Gordon Kindlmann. Big thanks to my family and friends.

ABSTRACT

Computer simulation is a powerful tool for exploring real-world processes. Special considerations need to be made at each point to ensure an accurate and efficient simulation. This thesis focuses on topics at different points in the simulation process. We present a finite element algorithm for solving the Stokes system based on the Scott-Vogelius technique together with the iterated penalty method. By projecting the Scott-Vogelius pressure onto a continuous function space, we can recover an accurate, continuous pressure solution while retaining a divergence-free velocity. This algorithm could be of particular interest when modeling incompressible complex fluids where there is a strong dependence on the pressure. Full details of our implementation in the automated finite element software FEniCS are described and numerical results are given. Next, we derive a stochastic performance model for traditional and pipelined Krylov linear solvers. By using a description that accounts for stochastic noise, modeled by an analytical probability distribution, we are able to clarify a folk theorem that pipelined methods can only result in a speedup of $2\times$ over the naive implementation. Examining repeated runs, we are also able to study machine noise in depth. These results are particularly applicable in unpredictable computing environments, such as computing platforms with shared resources. Details of our experiments in the scientific computing software PETSc are given.

CHAPTER 1

INTRODUCTION

Simulation is the computer imitation of some real-world event. Through simulation, scientists are able to better understand and predict physical processes. For example, the ubiquitous Navier-Stokes equations can be used in medicine to model aneurysms or cerebrospinal fluid flow and simulations help researchers study these events.

Computer simulation can be thought of as several distinct steps, shown as a pipeline in Figure 1.1. Continuous physical models take the form of differential equations. Computers assist in the simulation of these models because obtaining analytical solutions can be difficult due to the domain, external forces, or equations themselves. An approximation algorithm or discretization scheme is applied to transform a continuous problem into a discrete algebraic equation that must be solved. Numerical linear algebra is an extensive field within computational science and care must be taken in the choice of algorithm and its implementation to ensure an efficient solution. Finally, the solution can be applied to a particular application of interest.

This thesis contains topics at different points in the simulation pipeline. In Chapter 2, we introduce a discretization scheme for the Stokes equations using finite elements. The finite element method is a powerful tool that uses variational calculus to transform continuous differential equations into finite dimensional problems. Its success stems from its generality, flexibility, and rigorous mathematical underpinnings (e.g. a priori error analysis). Furthermore, it can be thought of as a black box, making it amenable to automation. A flowchart of the steps of the finite element method and related topics are shown in Figure 1.2. See [38]

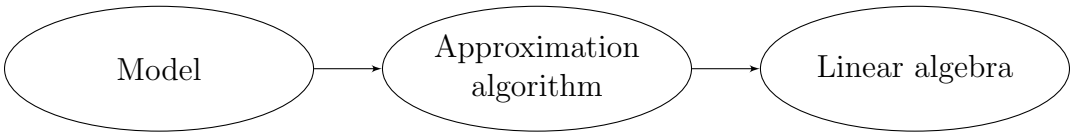


Figure 1.1: Overview of the simulation pipeline.

for a brief introduction and [9] for a mathematical approach.

In this work, we will use the user-friendly FEniCS Project framework for the implementation of finite elements. In FEniCS, we define a mesh, function spaces and functions, and a variational problem using a syntax that looks similar to handwritten mathematics. FEniCS automates basis function generation and finite element assembly and interfaces with linear

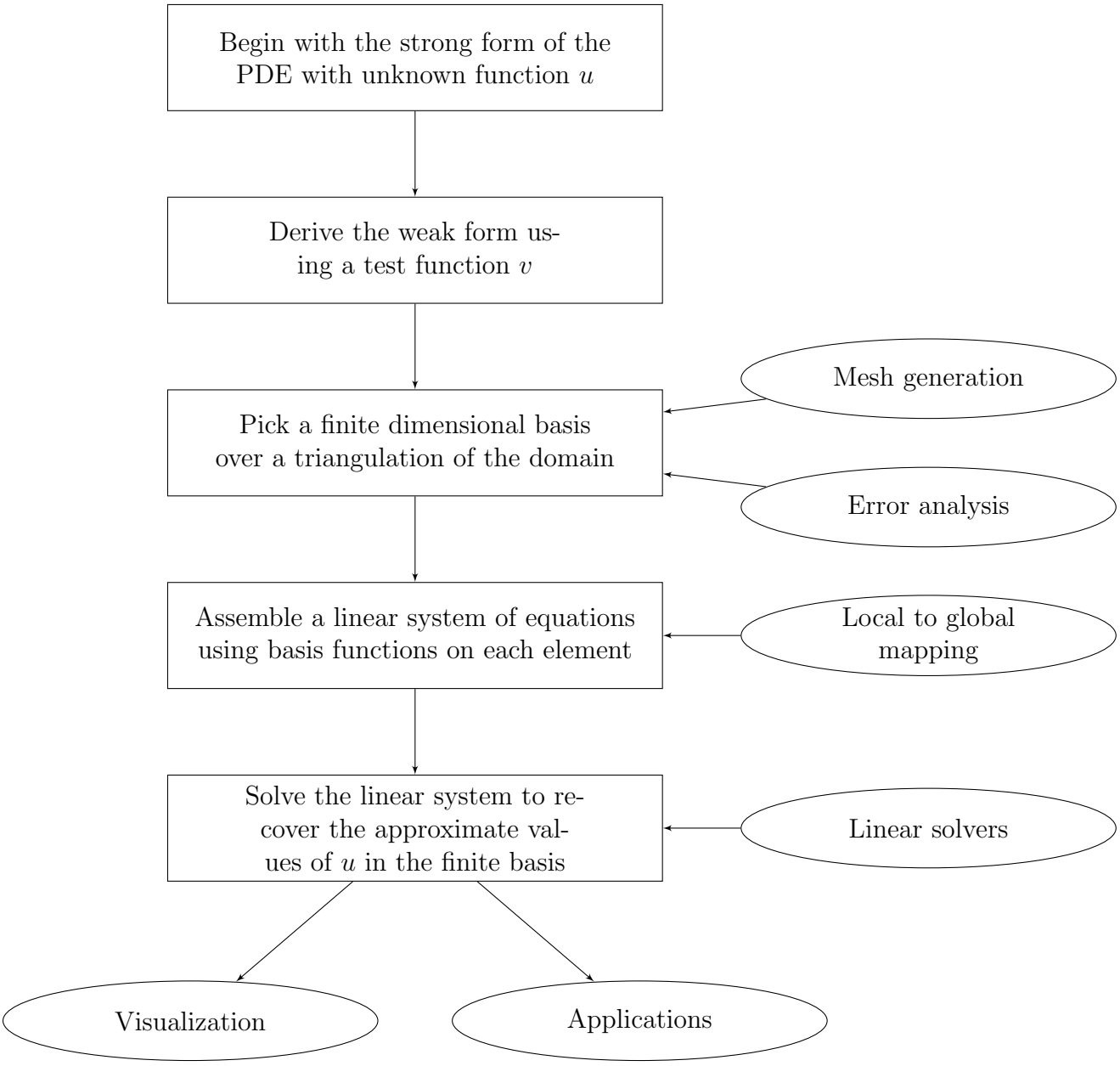


Figure 1.2: Overview of the finite element method.

algebra libraries such as PETSc (Portable, Extensible Toolkit for Scientific Computation) [4, 3, 5] and Trilinos [22] to solve the discrete problem. FEniCS can also handle nonlinear and time dependent problems, complicated boundary conditions, and complex meshes and subdomains.

The finite element method produces an algebraic equation that must be solved. Special care must be taken when picking a linear solver because the system could be ill-conditioned, sparse, or very large with the need to scale on parallel computers. In high performance settings, it is difficult to judge tradeoffs between different solvers. Iterative Krylov solvers are often employed because they use less memory and are easier to parallelize than direct solvers based on Gaussian Elimination. However, they do not scale well on massively parallel machines because of global communications used in vector normalizations and orthogonalizations. To mitigate these overheads, algebraically equivalent "pipelined" methods have been introduced which rearrange the algorithms so that communication and computation can overlap. Still, it is difficult to describe the performance of these methods on HPC machines. In Chapter 3, we introduce a stochastic performance model for Krylov and pipelined Krylov methods that captures a realistic representation of the performance of these methods.

For experiments in this work, we will use Krylov solvers implemented in the scientific computing tool PETSc on supercomputers such as the Cray XC40 at the Argonne Leadership Computing Facility. PETSc is a collection of data structures and associated functions for the numerical solution of differential equations (e.g. matrices, vectors, linear solvers, and preconditioners) efficiently implemented and designed for use on high performance computers.

CHAPTER 2

TOWARDS A UNIFIED FINITE ELEMENT FOR THE STOKES EQUATIONS

2.1 Introduction

The Stokes equations describe the low Reynold's number limit of the Navier-Stokes equations for steady, incompressible fluids, such as paint or lava. The Stokes system takes the form

$$-\Delta \mathbf{u} + \nabla p = \mathbf{f} \tag{2.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2.2}$$

for velocity $\mathbf{u}$ and pressure p with force term $\mathbf{f}$ in a domain $\Omega \in \mathbb{R}^d$ with, say, $\mathbf{u} = \mathbf{0}$ on $\partial\Omega$.

Finding stable finite element schemes for the Stokes system is difficult because mixed formulations (methods that require more than one approximation space) must satisfy the Ladyzhenskaya-Babuška-Brezzi or inf-sup condition [10]. Furthermore, it is difficult to satisfy the mass conservation (2.2) and the discrete variational form results in a saddle point problem that requires special care to solve.

The Stokes system has been studied extensively and there are many well known stable finite element methods suitable for simulation. Taylor and Hood introduced a widely used finite element that uses continuous polynomials of degree k for the velocity space and continuous polynomials of degree $k-1$ for the pressure with $k \geq 2$ [45, 8]. The Crouzeix-Raviart non-conforming element lets the velocity space be integral moments of order k over cell edges and uses a discontinuous pressure space of order $k-1$ [13]. Arnold, Brezzi, and Fortin used piecewise continuous linear functions enriched with bubbles for the velocity space [2]. Stabilization techniques have also been introduced that add a stabilization terms with parameter δ to the bilinear forms, see [16] for a discussion. Finally, Scott and Vogelius introduced an iterative method that uses a higher order continuous approximation space for the velocity

and is exactly divergence free [41].

Here, we introduce a "unified" finite element that combines the strengths of the Taylor-Hood and Scott-Vogelius methods and eliminates the weaknesses. In Sections 2.2 and 2.3 we give a review of Taylor-Hood and the Scott-Vogelius iterated penalty method, including convergence results and show that this method does not result in "locking". In Sections 2.4 and 2.5 we introduce the unified Stokes algorithm and give an implementation in FEniCS. In Sections 2.6 and 2.7 we test our algorithm in a variety of domains and compare the results to implementations of Taylor-Hood and Scott-Vogelius. We conclude our work in Section 2.8.

2.2 Taylor-Hood

The variational problem for the Stokes equations uses the bilinear forms

$$\begin{aligned} a(\mathbf{u},\mathbf{v}) &:= \int_\Omega \nabla\mathbf{u} : \nabla\mathbf{v}\, dx \\ b(\mathbf{v},p) &:= \int_\Omega (\nabla\cdot\mathbf{v})\, p\, dx \\ (\mathbf{f},\mathbf{v}) &:= \int_\Omega \mathbf{f}\cdot\mathbf{v}\, dx \end{aligned} \tag{2.3}$$

and can be written: Find $\mathbf{u}\in\mathbf{V}$ and $p\in\Pi$ such that

$$\begin{aligned} a(\mathbf{u},\mathbf{v}) - b(\mathbf{v},p) &= (\mathbf{f},\mathbf{v}) \quad \forall\mathbf{v}\in\mathbf{V} \\ b(\mathbf{u},q) &= \mathbf{0} \quad \forall q\in\Pi. \end{aligned} \tag{2.4}$$

Let $V_{h,k}$ denote the space of C^0 piecewise polynomials of degree k on a simplicial triangulation $\mathcal{T}_h$ of size h of Ω. The Taylor-Hood method consists of applying (2.4) with $\mathbf{V} = \{\mathbf{v}\in V_{h,k}^d \,|\, \mathbf{v}=\mathbf{0} \text{ on } \partial\Omega\}$ and $\Pi = V_{h,k-1}/\mathbb{R}$ for $k\geq 2$. In practice, the two equations in (2.4) are solved simultaneously. Although the Taylor-Hood pressure approximation is continuous, the mass conservation (2.2) is enforced weakly over each element in the domain.

Some effects related to the lack of mass conservation with Taylor-Hood are described in [29].

2.3 Scott-Vogelius

The Scott-Vogelius method consists of applying (2.4) with the same $\mathbf{V}$ as Taylor-Hood and $\Pi = \nabla \cdot \mathbf{V}$, typically for $k \geq 4$ in two dimensions [40, 41] and $k \geq 6$ in three dimensions [47]. The solution of the Scott-Vogelius system is complicated by the fact that the pressure space depends on the singular vertices in the mesh. This can be avoided by using the iterated penalty algorithm for the Scott-Vogelius element [9, Equation 13.1.4], which uses the bilinear form

$$a_r(\mathbf{u}, \mathbf{v}) := \int_\Omega \nabla \mathbf{u} : \nabla \mathbf{v} + r(\nabla \cdot \mathbf{u})(\nabla \cdot \mathbf{v})\, dx \tag{2.5}$$

to enforce the divergence constraint in (2.2). The variational form $a_r(\cdot, \cdot)$ in (2.5) is also advocated as a preconditioner in [31] and a stabilizer in [33].

The Scott-Vogelius iterated penalty method can be written as: find $\mathbf{u}^n \in \mathbf{V}$ such that

$$\begin{aligned} a_r(\mathbf{u}^n, \mathbf{v}) &= F(\mathbf{v}) - b(\mathbf{v}, p^n) \ \text{ for all } \mathbf{v} \in \mathbf{V} \\ p^{n+1} &= p^n - r \nabla \cdot \mathbf{u}^n, \end{aligned} \tag{2.6}$$

where a typical choice is $r = 1000$ and we can start with $p^0 = 0$. Equivalently, the pressure can be determined by

$$p^{n+1} = \nabla \cdot \mathbf{w}^{n+1}, \quad \mathbf{w}^{n+1} = \mathbf{w}^n - r\mathbf{u}^n, \tag{2.7}$$

where we start with $\mathbf{w}^0 = \mathbf{0}$.

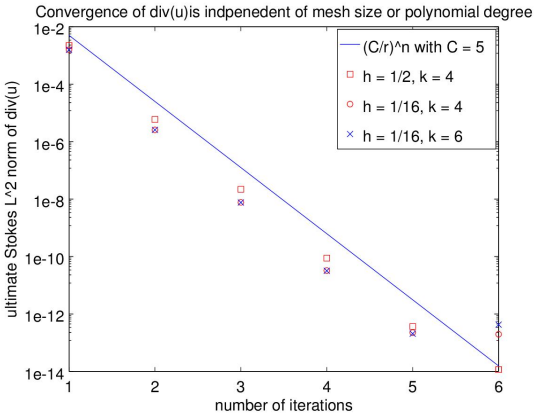


Figure 2.1: Unified Stokes and Scott-Vogelius $\|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)}$ converge geometrically.

When there are no nearly-singular vertices, it is known that

$$\sup_{\mathbf{v} \in V_{h,k}^d} \frac{b(\mathbf{v}, q)}{\|\mathbf{v}\|_{H^1(\Omega)}} \geq \beta^{\mathrm{SV}} \|q\|_{L^2(\Omega)/\mathbb{R}} \quad \forall q \in \Pi = \nabla \cdot \mathbf{V}, \tag{2.8}$$

provided $k \geq 4$ in two dimensions [40, 41] and $k \geq 6$ in three dimensions [47]. We discuss extensions to smaller values of k for special meshes in section 6. Convergence of the iterated penalty solution algorithm is given in [9, Section 13.2] as follows.

Theorem 1. *Suppose that* $(\mathbf{u}, p)$ *represents the solution of the Stokes system* (2.1)*, and let* $(\mathbf{u}_{h,k}^{SV}, p_{h,k}^{SV})$ *denote the solution to* (2.4) *with* $\mathbf{V} = \{\mathbf{v} \in V_{h,k}^d \,|\, \mathbf{v} = \mathbf{0} \text{ on } \partial\Omega\}$ *and* $\Pi = \nabla \cdot \mathbf{V}$ *for* $k \geq 4$ *in two dimensions and* $k \geq 6$ *in three dimensions. Finally, let let* $\mathbf{u}^n \in \mathbf{V}$ *be defined by* (2.6) *and let* $p^n \in \nabla \cdot \mathbf{V}$ *be equivalently defined by* (2.7) *or* (2.6)*, with* $r \geq 1$*. Then*

$$\begin{aligned}
\left\|\mathbf{u} - \mathbf{u}_{h,k}^{SV}\right\|_{H^1(\Omega)} + \left\|p - p_{h,k}^{SV}\right\|_{L^2(\Omega)/\mathbb{R}} &\leq Ch^k \|\mathbf{u}\|_{H^{k+1}(\Omega)} \\
\left\|\mathbf{u}^n - \mathbf{u}_{h,k}^{SV}\right\|_{H^1(\Omega)} + \frac{1}{r}\left\|p^n - p_{h,k}^{SV}\right\|_{L^2(\Omega)/\mathbb{R}} &\leq C \|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)} \\
\|\mathbf{u}^n - \mathbf{u}\|_{H^1(\Omega)} &\leq Ch^k \|\mathbf{u}\|_{H^{k+1}(\Omega)} + C \|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)} \\
\|p^n - p\|_{L^2(\Omega)/\mathbb{R}} &\leq Ch^k \|\mathbf{u}\|_{H^{k+1}(\Omega)} + Cr \|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)},
\end{aligned} \tag{2.9}$$

where the constant C *depends only on* k*,* β^{SV} *and* Ω*. Moreover,* $\|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)} \leq (C/r)^n$*,*

that is, it converges to zero geometrically with a rate C/r.

One consequence of the theorem is that $\|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)}$ can be used to monitor convergence of (2.6). In [27], direct solvers were used for the linear systems related to (2.6) involving $a_r(\cdot,\cdot)$.

2.3.1 No locking

The paper by Linke et al. [30] shows that for high-degree piecewise polynomials, adding a divergence term to a form does not cause locking. More precisely, they consider the Taylor-Hood finite element solution $\mathbf{u}_h^{\mathrm{TH}}$ to (2.4) but using the form a_r instead of a. The Scott-Vogelius solution $\mathbf{u}^{\mathrm{SV}}$ is the same whether a_r or a is used. Then for $k \geq 4$ in two dimensions [40, 41] and $k \geq 6$ in three dimensions [47], it is shown that [30]

$$\left\| \nabla(\mathbf{u}_{h,k}^{\mathrm{TH}} - \mathbf{u}_{h,k}^{\mathrm{SV}}) \right\|_{L^2(\Omega)} \leq \frac{C}{r}, \tag{2.10}$$

at least under suitable mesh restrictions for which the Scott-Vogelius method satisfies the inf-sup condition. The result (2.10) shows that locking does not occur.

2.3.2 Velocity and pressure are independent

The conventional way of thinking about both the Taylor-Hood and Scott-Vogelius methods is that they simultaneously solve for $\mathbf{u}_h$ and p_h. It is known that the approaches have complementary positives and negatives [46]. In particular, the Scott-Vogelius method can produce a $\mathbf{u}_h$ whose divergence is zero to within roundoff error, but the pressure p_h is discontinuous. By contrast, the Taylor-Hood pressure is quite smooth, but the divergence of the velocity is only $\mathcal{O}(h^k)$. Here we define an algorithm that combines the best of these two behaviors and eliminates the worst.

The pressure and velocity approximations can be completely decoupled [9]. For example,

we can define

$$Z_{h,k}^{\mathrm{TH}} = \left\{ \mathbf{v} \in \mathbf{V} \,\middle|\, b(\mathbf{v}, q) = 0 \; \forall q \in V_{h,k-1}/\mathbb{R} \right\}, \quad Z_{h,k}^{\mathrm{SV}} = \left\{ \mathbf{v} \in \mathbf{V} \,\middle|\, \nabla \cdot \mathbf{v} = 0 \right\}. \tag{2.11}$$

We have $Z_{h,k}^{\mathrm{SV}} \subset Z_{h,k}^{\mathrm{TH}}$. Then $\mathbf{u}_{h,k}^{\mathrm{TH}}$ and $\mathbf{u}_{h,k}^{\mathrm{SV}}$ in $\mathbf{V}$ can be characterized by

$$a_0(\mathbf{u}_{h,k}^*, \mathbf{v}) = F(\mathbf{v}) \; \forall \mathbf{v} \in Z_{h,k}^*, \tag{2.12}$$

where $*$ stands for either SV or TH. Once $\mathbf{u}_{h,k}^*$ has been obtained, then $p_{h,k}^{\mathrm{TH}} \in V_{h,k-1}/\mathbb{R}$ and $p_{h,k}^{\mathrm{SV}} \in \Pi = \nabla \cdot \mathbf{V}_{h,k}^d$ can be determined via

$$b(\mathbf{v}, p_{h,k}^*) = F(\mathbf{v}) \; \forall \mathbf{v} \in \left(Z_{h,k}^*\right)^{\perp}, \tag{2.13}$$

where $\left(Z_{h,k}^*\right)^{\perp}$ denotes the space orthogonal to $Z_{h,k}^*$ in $\mathbf{V}_{h,k}^d$ with respect to the inner product $a_0(\cdot, \cdot)$. The existence of $p_{h,k}^{\mathrm{TH}}$ is determined by the inf-sup condition

$$\sup_{\mathbf{v} \in V_{h,k}^d} \frac{b(\mathbf{v}, q)}{\|\mathbf{v}\|_{H^1(\Omega)}} \geq \beta^{\mathrm{TH}} \|q\|_{L^2(\Omega)/\mathbb{R}} \; \forall q \in V_{h,k-1}/\mathbb{R}. \tag{2.14}$$

It is known that $\beta^{\mathrm{TH}} > 0$ can be chosen independent of h with only mild mesh restrictions [8]. Similarly, the existence of $p_{h,k}^{\mathrm{SV}}$ is determined by the inf-sup condition (2.8). Unfortunately, the characterization (2.13) does not provide a useful computational algorithm, but it does establish that the pressure and velocity can be viewed as independent. This is the motivating idea behind the unified Stokes algorithm.

2.4 Unified Stokes algorithm

The unified Stokes approach is to first solve for $\mathbf{u}_{h,k}^{\mathrm{SV}}$ via (2.6), and then to define $p_{h,k}^{\mathrm{US}} \in V_{h,k-1}/\mathbb{R}$ via the L^2 projection of $p_{h,k}^{\mathrm{SV}}$ onto $V_{h,k-1}/\mathbb{R}$:

$$(p_{h,k}^{\mathrm{US}}, q) = (p_{h,k}^{\mathrm{SV}}, q) \; \forall q \in V_{h,k-1}/\mathbb{R}. \tag{2.15}$$

More precisely, we define the pair $(\mathbf{u}_{h,k}^{\mathrm{SV}}, p_{h,k}^{\mathrm{US}})$ to be the unified Stokes approximation of the Stokes system.

Note that we can approximate $p_{h,k}^{\mathrm{US}}$ defined by:

$$(p_{h,k}^{\mathrm{US},n}, q) = (\nabla \cdot \mathbf{w}^n, q) = -(\mathbf{w}^n, \nabla q) \; \forall q \in V_{h,k-1}/\mathbb{R}, \tag{2.16}$$

where $\mathbf{w}^n$ is defined via the iterated penalty method (2.7). Thus there are three mathematically equivalent ways to compute $p_{h,k}^{\mathrm{US},n} \in V_{h,k-1}/\mathbb{R}$:

$$\begin{aligned} (p_{h,k}^{\mathrm{US},n}, q) &= (\nabla \cdot \mathbf{w}^n, q) \; \forall q \in V_{h,k-1}/\mathbb{R}, \\ (p_{h,k}^{\mathrm{US},n}, q) &= (p^n, q) \; \forall q \in V_{h,k-1}/\mathbb{R}, \\ (p_{h,k}^{\mathrm{US},n}, q) &= -(\mathbf{w}^n, \nabla q) \; \forall q \in V_{h,k-1}/\mathbb{R}, \end{aligned} \tag{2.17}$$

where $p^n \in V_{h,k-1}^{\mathrm{disc}}/\mathbb{R}$, the space of discontinuous piecewise polynomials of order $k-1$, is defined by (2.6) and $\mathbf{w}^n \in \mathbf{V}$ is defined by (2.7).

Mathematically, $p_{h,k}^{\mathrm{US}} = \mathcal{P} p_{h,k}^{\mathrm{SV}}$, where $\mathcal{P}$ is the L^2 projection onto the space $V_{h,k-1}/\mathbb{R}$.

$r=1$	L^2	maximum
$\left\|p_1^{\mathrm{US}}-p_2^{\mathrm{US}}\right\|$	2.76e-16	1.33e-15
$\left\|p_1^{\mathrm{US}}-p_3^{\mathrm{US}}\right\|$	7.57e-14	3.51e-13
$\left\|p_2^{\mathrm{US}}-p_3^{\mathrm{US}}\right\|$	7.57e-14	3.52e-13
$r=1000$		
$\left\|p_1^{\mathrm{US}}-p_2^{\mathrm{US}}\right\|$	2.16e-12	1.09e-11
$\left\|p_1^{\mathrm{US}}-p_3^{\mathrm{US}}\right\|$	7.29e-11	3.34e-10
$\left\|p_2^{\mathrm{US}}-p_3^{\mathrm{US}}\right\|$	7.34e-11	3.40e-10

Table 2.1: Equivalence of the expressions in (2.17) after one iteration with $h = 1/2$ and $k = 4$ for various values of r.

More precisely, $p_{h,k}^{\mathrm{US},n} = \mathcal{P}p^n$. Thus

$$\begin{aligned}
\left\|p-p_{h,k}^{\mathrm{US},n}\right\|_{L^2(\Omega)/\mathbb{R}}^2 &= \|p-\mathcal{P}p^n\|_{L^2(\Omega)/\mathbb{R}}^2 \\
&= \left\|p-\mathcal{P}p+\mathcal{P}\big(p-p^n\big)\right\|_{L^2(\Omega)/\mathbb{R}}^2 \\
&= \left\|\mathcal{P}^{\perp}p\right\|_{L^2(\Omega)/\mathbb{R}}^2 + \left\|\mathcal{P}\big(p-p^n\big)\right\|_{L^2(\Omega)/\mathbb{R}}^2 \\
&= \|p-\mathcal{P}p\|_{L^2(\Omega)/\mathbb{R}}^2 + \left\|\mathcal{P}\big(p-p^n\big)\right\|_{L^2(\Omega)/\mathbb{R}}^2 \\
&\le \|p-\mathcal{P}p\|_{L^2(\Omega)/\mathbb{R}}^2 + \|p-p^n\|_{L^2(\Omega)/\mathbb{R}}^2 \\
&\le \big(\,\|p-\mathcal{P}p\|_{L^2(\Omega)/\mathbb{R}} + \|p-p^n\|_{L^2(\Omega)/\mathbb{R}}\,\big)^2.
\end{aligned}$$

Thus

$$\left\|p-p_{h,k}^{\mathrm{US},n}\right\|_{L^2(\Omega)/\mathbb{R}} \le \|p-\mathcal{P}p\|_{L^2(\Omega)/\mathbb{R}} + \|p-p^n\|_{L^2(\Omega)/\mathbb{R}}\,. \tag{2.18}$$

Using Theorem 1 for the Scott-Vogelius algorithm, we have thus proved the following.

Theorem 2. *Suppose that* $(\mathbf{u},p)$ *represents the solution of the Stokes system* (2.5). *Let* $\mathbf{u}_h \in \mathbf{V}$ *be defined by the iteration* (2.6) *with* $r \ge 1$, *and let* $p_{h,k}^{US,n} \in V_{h,k-1}/\mathbb{R}$ *be defined by*

(2.17). *Then*

$$\left\|p - p_{h,k}^{US,n}\right\|_{L^2(\Omega)/\mathbb{R}} \leq \inf_{q \in V_{h,k-1}/\mathbb{R}} \|p - q\|_{L^2(\Omega)/\mathbb{R}} + Ch^k \|\mathbf{u}\|_{H^{k+1}(\Omega)} + Cr \|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)}, \tag{2.19}$$

where the constant C depends only on k, β^{SV} and Ω. Moreover, $\|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)} \leq (C/r)^n$.

Proof. From (2.9), we have

$$\|p - p^n\|_{L^2(\Omega)/\mathbb{R}} \leq Ch^k \|\mathbf{u}\|_{H^{k+1}(\Omega)} + Cr \|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)}. \tag{2.20}$$

From (2.18) and (2.20), we have

$$\begin{aligned} \left\|p - p_{h,k}^{\mathrm{US},n}\right\|_{L^2(\Omega)/\mathbb{R}} &\leq \|p - \mathcal{P}p\|_{L^2(\Omega)/\mathbb{R}} + \|p - p^n\|_{L^2(\Omega)/\mathbb{R}} \\ &\leq \|p - \mathcal{P}p\|_{L^2(\Omega)/\mathbb{R}} + Ch^k \|\mathbf{u}\|_{H^{k+1}(\Omega)} + Cr \|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)}. \end{aligned} \tag{2.21}$$

Since $\|p - \mathcal{P}p\|_{L^2(\Omega)/\mathbb{R}} = \inf_{q \in V_{h,k-1}/\mathbb{R}} \|p - q\|_{L^2(\Omega)/\mathbb{R}}$, the proof is complete. □

This demonstrates that the velocity and pressure approximations are fully decoupled.

2.5 Implementation

In this section, we will discuss our implementation of the unified Stokes algorithm in the automated finite element software, FEniCS, as well as implementations of Scott-Vogelius and Taylor-Hood for comparison. This will be followed by numerical results.

The flow of the unified Stokes algorithm (as well as Scott-Vogelius and Taylor-Hood for comparison) is shown in Figure 2.2, where

$$V_{h,k-1}^* = \begin{cases} V_{h,k-1}^{\mathrm{disc}} & \mathrm{SV} \\ V_{h,k-1} & \mathrm{US} \end{cases}.$$

There are several details worthy of noting in our implementations. We advocate the use of the first form in (2.17) so that the iterated penalty method becomes: Find $\mathbf{u}^n \in \mathbf{V}$ such that

$$a_r(\mathbf{u}^n, \mathbf{v}) = F(\mathbf{v}) - b(\mathbf{v}, \nabla \cdot \mathbf{w}^n) \text{ for all } \mathbf{v} \in \mathbf{V}$$

and update $\mathbf{w}^n \in \mathbf{V}$ using

$$\mathbf{w}^{n+1} = \mathbf{w}^n - r\mathbf{u}^n.$$

Note that we use the substitution shown in (2.7) instead of using the pressure explicitly and compute the pressure only when it is needed. This could be particularly useful in time dependent or nonlinear iterations where the pressure is not explicitly needed at each step. On the other hand, it requires retention of an additional velocity field which increases storage requirements.

We choose to stop the iterated penalty method when $\|\nabla \cdot \mathbf{u}\|_{L^2(\Omega)} \leq$ 10e-10 and thus need to compute this value in each iteration. There are many ways to do this, but we find it efficient to compute $\sqrt{\int_\Omega (\nabla \cdot \mathbf{u})^2 \, dx}$ directly using the `assemble` function in Dolfin as opposed to, say, finding a function g such that $g = \nabla \cdot \mathbf{u}$ and calculating the L^2 norm. A full implementation of unified Stokes in FEniCS with $h = \frac{1}{16}$ and $k = 4$ can be found in Program 2.3.

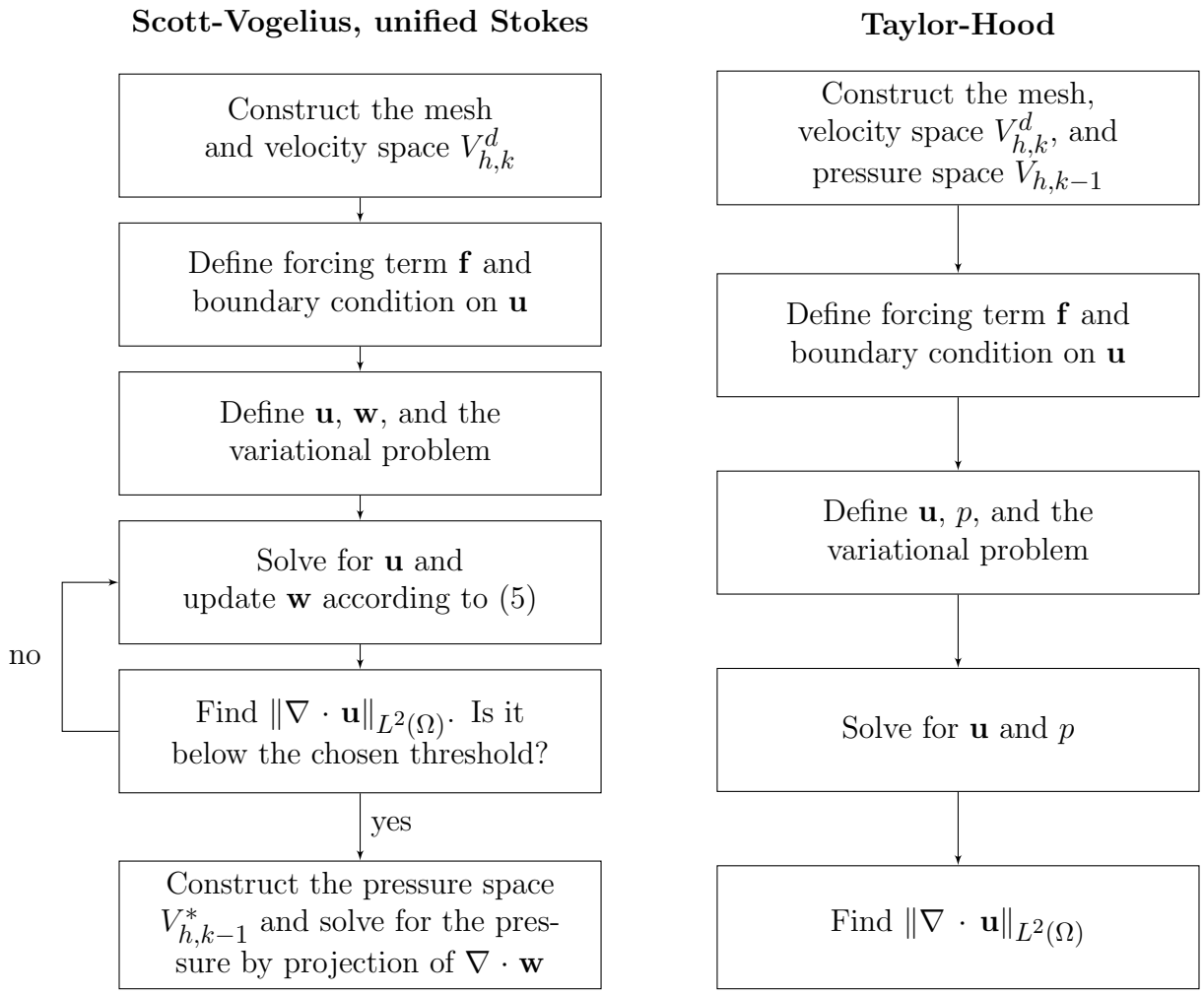


Figure 2.2: Algorithm flow for Scott-Vogelius and unified Stokes (left) and Taylor-Hood (right).

```
# create mesh and define velocity and pressure function spaces
mesh = UnitSquareMesh(16, 16, "crossed")
k = 4
V = VectorFunctionSpace(mesh, "Lagrange", k)

# define boundary condition
boundary_exp = Expression(("sin(4*pi*x[0])*cos(4*pi*x[1])",
                           "-cos(4*pi*x[0])*sin(4*pi*x[1])"))
bc = DirichletBC(V, boundary_exp, "on_boundary")

# set the parameters
f = Expression(("28*pow(pi, 2)*sin(4*pi*x[0])*cos(4*pi*x[1])",
                "-36*pow(pi, 2)*cos(4*pi*x[0])*sin(4*pi*x[1])"))
r = 1.0e3

# define test and trial functions, and function that is updated
u = TrialFunction(V)
v = TestFunction(V)
w = Function(V)

# set the variational problem
a = inner(grad(u), grad(v))*dx + r*div(u)*div(v)*dx
b = -div(w)*div(v)*dx
F = inner(f, v)*dx

u = Function(V)
pde = LinearVariationalProblem(a, F - b, u, bc)
solver = LinearVariationalSolver(pde)

# Scott-Vogelius iterated penalty method
iters = 0; max_iters = 100; div_u_norm = 1
while iters < max_iters and div_u_norm > 1e-10:

    # solve and update w
    solver.solve()
    w.vector().axpy(-r, u.vector())

    # find the L^2 norm of div(u) to check stopping condition
    div_u_norm = sqrt(assemble(div(u)*div(u)*dx(mesh)))
    iters += 1

# unified Stokes solution
p_US = project(div(w), FunctionSpace(mesh, "Lagrange", k - 1))
```

Figure 2.3: Unified Stokes algorithm

2.6 Numerical experiments

2.6.1 Manufactured solution problem

First, we will use the method of manufactured solutions so that we can easily measure the error of our algorithm. We choose our data so that the exact solution is

$$\mathbf{u} = \begin{bmatrix} \sin(4\pi x)\cos(4\pi y) \\ -\cos(4\pi x)\sin(4\pi y) \end{bmatrix} \tag{2.22}$$

$$p = \pi \cos(4\pi x)\cos(4\pi y). \tag{2.23}$$

Differentiating, we find

$$\mathbf{f} = \begin{bmatrix} 28\pi^2 \sin(4\pi x)\cos(4\pi y) \\ -36\pi^2 \cos(4\pi x)\sin(4\pi y) \end{bmatrix}.$$

Let the domain Ω be the two dimensional unit square with a crossed computation mesh and constant $r = 10^3$.

First, we numerically validate Theorem 1.1, which states that $\|\nabla \cdot \mathbf{u}\|_{L^2(\Omega)} \leq (C/r)^n$. Figure 2.1 shows that $\|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)}$ converges independently of mesh size or polynomial order with $r = 10^3$, which is in agreement with Theorem 1.1.

Similarly, we want to show that all terms in (2.17) are equivalent numerically. We do one loop of the iterated penalty method using each of the three forms (call them p_1^{US}, p_2^{US}, and p_3^{US}) and compare their L^2 and maximum differences. The results are given in Table 2.1. The differences are within roundoff error when $r = 1$ and small when $r = 1000$. We choose to use $r = 1000$ in our experiments since the magnitude of this constant is the coefficient of the penalty term in the iterated penalty method and dictates the convergence rate of the method. A choice of r with a smaller magnitude will take more iterations until convergence.

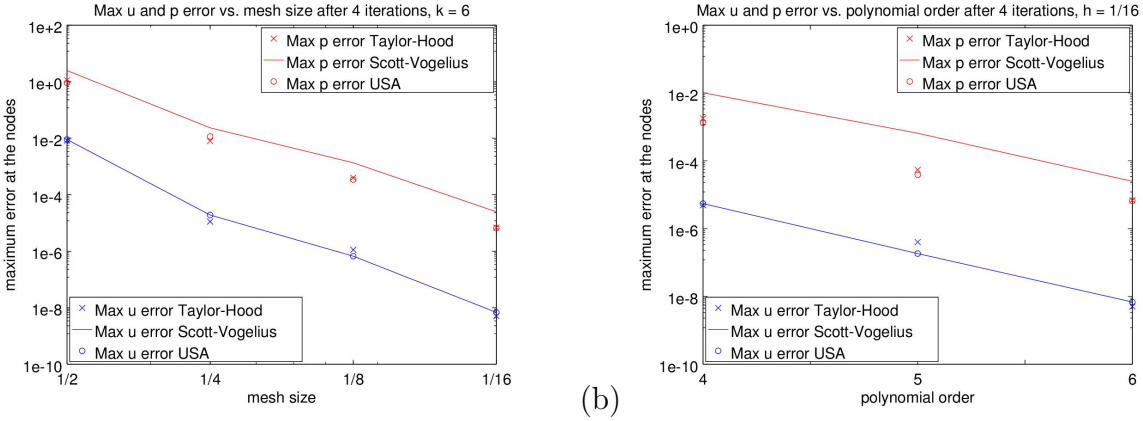


Figure 2.4: Computed maximum error as a function of (a) mesh size with fixed $k = 6$ and (b) polynomial order with fixed $h = 1/16$.

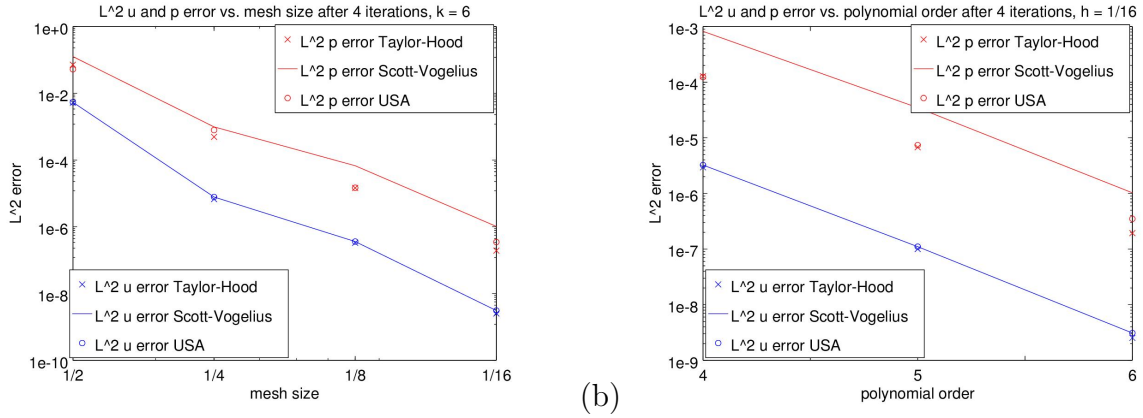


Figure 2.5: Computed L^2 error as a function of (a) mesh size with fixed $k = 6$ and (b) polynomial order with fixed $h = 1/16$.

		Taylor-Hood	Scott-Vogelius, unified Stokes	
$\frac{1}{h}$	k	$\|\nabla \cdot \mathbf{u}\|_{L^2(\Omega)}$	$\|\nabla \cdot \mathbf{u}^1\|_{L^2(\Omega)}$	$\|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)}$
2	4	7.05e-01	2.22e-03	8.51e-11
2	5	9.09e-02	1.63e-03	7.25e-11
2	6	5.49e-02	1.57e-03	3.44e-11
4	4	8.28e-03	1.57e-03	3.47e-11
4	5	1.03e-02	1.57e-03	3.33e-11
4	6	1.93e-04	1.57e-03	3.24e-11
8	4	3.63e-03	1.57e-03	3.26e-11
8	5	2.75e-04	1.57e-03	3.18e-11
8	6	1.69e-05	1.57e-03	3.14e-11
16	4	2.25e-04	1.57e-03	3.14e-11
16	5	9.45e-06	1.57e-03	3.10e-11
16	6	2.61e-07	1.57e-03	3.10e-11

Table 2.2: $\|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)}$ for varying h and k after one iteration and after termination, $n = 4$ iterations.

FEniCS				
$\frac{1}{h}$	k	Taylor-Hood	Scott-Vogelius	Unified Stokes
2	4	2.55e-02	2.82e-02	2.89e-02
2	5	2.96e-02	3.41e-02	3.68e-02
2	6	3.84e-02	4.55e-02	4.72e-02
4	4	4.66e-02	5.15e-02	5.27e-02
4	5	7.43e-02	8.40e-02	8.59e-02
4	6	1.22e-01	1.31e-01	1.31e-01
8	4	1.48e-01	1.55e-01	1.54e-01
8	5	6.62e-01	3.29e-01	3.29e-01
8	6	1.26e+00	5.83e-01	5.84e-01
16	4	9.72e-01	6.63e-01	6.58e-01
16	5	2.17e+00	1.70e+00	1.69e+00
16	6	4.33e+00	3.14e+00	3.12e+00

Table 2.3: Runtimes in seconds for varying h and k (one solve for Taylor-Hood, four iterations of (2.6) for Scott-Vogelius and unified Stokes).

We also want to show that unified Stokes is indeed a useful approach by comparing our results to both Taylor-Hood and Scott-Vogelius. We use Taylor-Hood, Scott-Vogelius, and unified Stokes to solve the Stokes system and compare each velocity and pressure to (2.22) and (2.23), respectively. The maximum and L^2 errors with varying mesh size and polynomial order can be found in Figures 2.4 and 2.5. The maximum errors for velocity and pressure were calculated by interpolating the exact solution onto the appropriate function space for each method and taking the maximum difference across the nodes. We used the Dolfin function `errornorm` with a degree rise of 3 to calculate the L^2 error. We see that all velocity solutions are approximately equivalent, but Taylor-Hood and unified Stokes compute a more accurate pressure than Scott-Vogelius in all cases that were tested. Table 2.2 details $\|\nabla \cdot \mathbf{u}\|_{L^2(\Omega)}$ for each method. For Scott-Vogelius and unified Stokes we show the result after one iteration and after each algorithm terminated (four iterations for all values of h and k for both) and $\sqrt{\int_\Omega (\nabla \cdot \mathbf{u})^2 \, dx}$ was calculated directly as described above. Scott-Vogelius and unified Stokes compute the same velocity, so we report them together.

In Table 2.3, we see that, surprisingly, unified Stokes and Scott-Vogelius are more efficient than Taylor-Hood, even though the former methods require multiple linear solves. The reason is that the Taylor-Hood system is much larger, since $\mathbf{u}$ and p are found simultaneously. For example, with $h = \frac{1}{16}$ and $k = 6$ the Taylor-Hood system has 50,211 unknowns compared to 37,250 for unified Stokes and Scott-Vogelius.

We choose a direct linear solver with full pivoting (the default solver in PETSc) to do our comparisons since that is a solver that is fully justified for the indefinite system in Taylor-Hood. We could have used a more efficient solver for unified Stokes and Scott-Vogelius, such as Cholesky, which would make the comparisons with Taylor-Hood even more favorable. A more extensive comparison of the linear algebra for Taylor-Hood on the one hand and unified Stokes and Scott-Vogelius on the other would require an in-depth study of possible solution algorithms.

We have limited our computational experiments for Scott-Vogelius and the unified Stokes

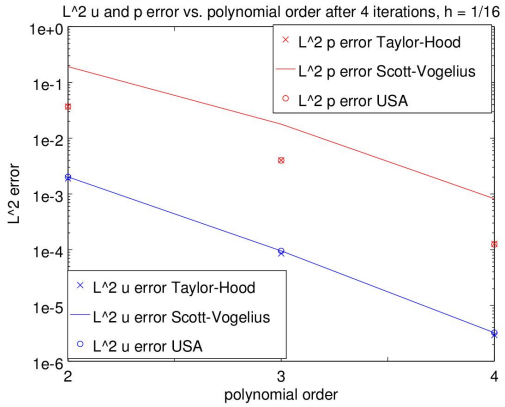


Figure 2.6: Computed L^2 error on a crossed mesh of size $h = 1/16$ and polynomial order $k \leq 4$.

algorithm to $k \geq 4$ since this is the case that does not have significant mesh restrictions (in two dimensions). However, on crossed meshes, it is known that this approach is optimally convergent for all $k \geq 1$ [35], and on other specialized meshes for low degree as well [26]. Moreover, convergence is assured even if only a fraction of the triangles come from crossed quadrilaterals, provided only that the fraction does not degenerate to zero as the mesh size is reduced [35]. Thus the unified Stokes approach can be used on such specialized meshes for $k \geq 2$ as demonstrated in Figure 2.6 for the problem in Section 2.6.1. In the case $k = 1$, the pressure is only piecewise constant, so there is no simple analog of a continuous pressure, nor is there a corresponding Taylor-Hood method.

2.6.2 Lid-driven cavity problem

Now we will test the methods using the lid-driven cavity problem with lower regularity than before. We will take the domain Ω again to be the two dimensional unit square and use the unstructured mesh generator Gmsh [19] to discretize the domain. Let $\mathbf{u} = (1, 0)$ on the top of the square and $\mathbf{u} = (0, 0)$ elsewhere, with $\mathbf{f} = (0, 0)$.

Table 2.4 shows $\|\nabla \cdot \mathbf{u}\|_{L^2}$ for each method. Like before, we terminate the iterated penalty method after $\|\nabla \cdot \mathbf{u}\|_{L^2} <$ 10e-10. The last column of Table 2.4 is the number of iterations needed. More iterations of the penalty method are used by Scott-Vogelius and unified Stokes

		Taylor-Hood	Scott-Vogelius, unified Stokes			
$\frac{1}{h}$	k	$\|\nabla \cdot \mathbf{u}\|_{L^2}$	$\|\nabla \cdot \mathbf{u}^1\|_{L^2}$	$\|\nabla \cdot \mathbf{u}^4\|_{L^2}$	$\|\nabla \cdot \mathbf{u}^n\|_{L^2}$	n
2	4	1.16e-01	3.52e-03	4.78e-08	8.57e-11	6
2	5	8.23e-02	3.88e-03	1.44e-08	1.84e-11	6
2	6	6.01e-02	4.20e-03	8.10e-09	1.02e-11	6
4	4	1.17e-01	4.58e-03	3.56e-07	1.01e-11	8
4	5	8.46e-02	4.89e-03	1.73e-07	5.78e-11	7
4	6	6.16e-02	5.18e-03	9.70e-08	3.00e-11	7
8	4	1.35e-01	5.48e-03	3.59e-09	7.43e-11	5
8	5	9.85e-02	5.79e-03	2.57e-09	6.96e-11	5
8	6	7.32e-02	6.06e-03	1.49e-09	2.66e-11	5
16	4	2.15e-01	6.76e-03	1.33e-08	2.38e-11	6
16	5	1.63e-01	6.89e-03	1.28e-08	9.36e-12	6
16	6	1.27e-01	7.05e-03	6.64e-09	2.43e-12	6

Table 2.4: $\|\nabla \cdot \mathbf{u}^n\|_{L^2(\Omega)}$ for varying h and k after one iteration, four iterations, and after termination in the lid-driven cavity problem.

for this problem, resulting in longer runtimes than Taylor-Hood. If the tolerance is relaxed to, say, 10e-6 the method converges in just a few iterations with competitive runtimes. Table 2.4 also shows that Taylor-Hood does not deal well with the divergence constraint in this problem compared to the one in Section 2.6.1. That is, $\|\nabla \cdot \mathbf{u}\|_{L^2}$ is about the same for all values of h, rather than going to zero as the mesh is refined or the polynomial degree is increased.

2.6.3 Collision in a square domain

Consider domain $\Omega = [-1, 1]^2$. Define vector functions

$$\mathbf{g}^x(x, y) = (1 - y^2, 0) \quad \text{and} \quad \mathbf{g}^y(x, y) = (0, 1 - x^2).$$

We have that $\mathbf{g}^x$, $\mathbf{g}^y \in H^1(\Omega)$ and $\nabla \cdot \mathbf{g}^x = \nabla \cdot \mathbf{g}^y = 0$ in Ω. Similarly, define scalar functions

$$p^x(x, y) = -2x \quad \text{and} \quad p^y(x, y) = -2y.$$

These are both in $L^2(\Omega)$ and have mean zero. In Ω, $-\Delta \mathbf{g}^x = (2,0) = -\nabla p^x$, and $-\Delta \mathbf{g}^y = (0,2) = -\nabla p^y$. Then $(\mathbf{g}^x, p^x)$ and $(\mathbf{g}^y, p^y)$ are strong solutions of Stokes in Ω. Since the Stokes equations are linear, any linear combination of these solutions is also an exact solution of the Stokes equations.

We will try to recover the exact solution $(\mathbf{g}^x + \mathbf{g}^y, p^x + p^y)$, representing colliding flow in a square domain, by solving the Stokes equations in the domain $\Omega = [-1,1]$ with boundary conditions

$$\mathbf{u}(x,y) = \begin{cases} (0, 1-x^2) & x = -1 \quad \text{or} \quad x = 1 \\ (1-y^2, 0) & y = -1 \quad \text{or} \quad y = 1 \end{cases}$$

and forcing term $\mathbf{f}(x,y) = (0,0)$. Here, we stop the iterated penalty method after 10 iterations. Unified Stokes results are shown in Table 2.5 where $\mathbf{u}_e$ and p_e are the exact solutions and runtime is in seconds. It is likely that the inconsistencies in the pressure approximation is due to the presence of triangles with two sides in the corners of the domain. The Taylor-Hood method had similar results in this experiment.

$1/h$	k	$\|\nabla \cdot \mathbf{u}\|_{L^2}$	$\|\mathbf{u} - \mathbf{u}_e\|_{L^2}$	$\|p - p_e\|_{L^2}$	Runtime
2	4	1.96e-13	7.81e-10	3.88e-01	1.35e-01
2	5	1.50e-13	1.24e-09	2.62e-01	1.50e-01
2	6	3.77e-13	2.76e-09	1.88e-01	1.84e-01
4	4	2.95e-13	7.34e-10	1.56e-06	2.07e-01
4	5	3.85e-13	1.77e-09	2.45e-07	3.34e-01
4	6	7.99e-13	2.88e-09	5.71e-07	4.25e-01
8	4	5.39e-13	1.51e-09	1.51e-06	5.17e-01
8	5	9.24e-13	1.62e-09	4.22e-07	8.55e-01
8	6	1.84e-12	3.50e-09	6.09e-07	1.42
16	4	1.47e-12	1.55e-09	3.43e-02	1.71
16	5	3.55e-12	5.06e-09	2.31e-02	3.15
16	6	3.68e-12	5.92e-09	1.66e-02	5.64

Table 2.5: Unified Stokes results for collision in a square domain after 10 iterations of the penalty method.

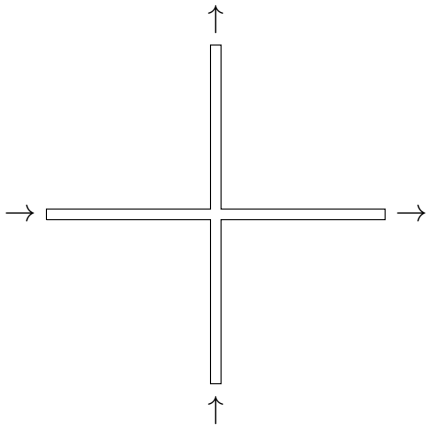


Figure 2.7: Cross shaped domain with inflow on the bottom and left sides and outflow on the top and right.

2.7 Complex domains

It has been shown that mass conservation is important for numerical stability [29]. In this section, we study mass conservation in problems with complex domains or irregular flows. We compute solutions to the Stokes equations in a cross shaped domain similar to [29] in 2.7.1 and in a bow tie domain with parabolic inflow and outflow in 2.7.2. Again, we use FEniCS to compute solutions using the Taylor-Hood and unified Stokes methods and compare $\|\nabla \cdot \mathbf{u}\|_{L^2}$. For the iterated penalty method used by unified Stokes, we let the penalty parameter $r = 10^4$ and stop the iterations when either $\|\nabla \cdot \mathbf{u}\|_{L^2} < 10^{-12}$ or 100 iterations are reached.

2.7.1 Cross shaped domain

Consider the cross domain shown in Figure 2.7, defined by

$$\Omega^x = \{(x, y) \ : \ |x| < 32, \ |y| < 1\} \quad \text{and} \quad \Omega^y = \{(x, y) \ : \ |y| < 32, \ |x| < 1\},$$

so that $\Omega = \Omega^x \cup \Omega^y$, the domain used in [29]. We take boundary conditions so that there is inflow on the bottom and left and outflow top and right. For this, we let $\mathbf{u}$ on the bottom

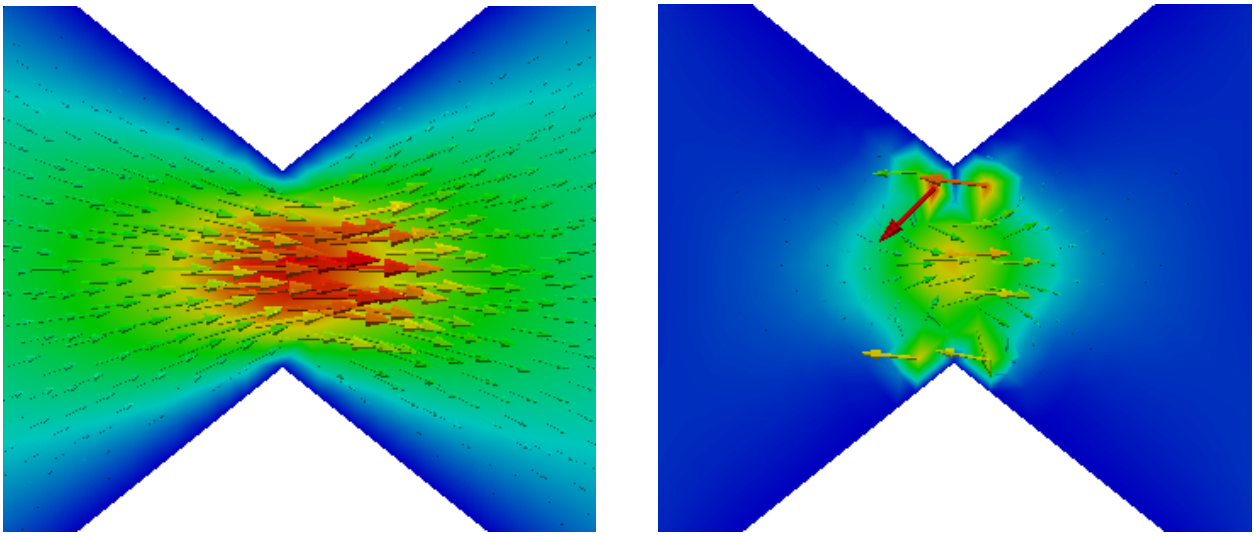
and top of the cross be $\mathbf{u}(x, y) = (0, 1 - x^2)$ and on left and right be $\mathbf{u}(x, y) = (1 - y^2, 0)$, similar to what was described in 2.6.3. We impose no-slip boundary conditions $\mathbf{u} = (0, 0)$ elsewhere and let $\mathbf{f} = (0, 0)$.

Table 2.6 shows the mass conservation of Taylor-Hood and unified Stokes for various discretizations and polynomials. In this complex domain, it takes many iterations of the penalty method to drive $\|\nabla \cdot \mathbf{u}\|_{L^2}$ to zero. Furthermore, $\left\|\nabla \cdot \mathbf{u}^4\right\|_{L^2}$ is quite small and in some cases more iterations of the penalty method is not very helpful. Many iterations also makes the unified Stokes method less efficient than Taylor-Hood without a much different velocity estimate. However, we do not see improvement in $\|\nabla \cdot \mathbf{u}\|_{L^2}$ using the Taylor-Hood method as the mesh is refined and polynomial order increased, as we did in the previous section. We also do not see that a better velocity approximation is correlated with better mass conservation, as was suggested in [29].

2.7.2 Bow tie domain

A "bow tie" is a unit square mesh with triangles cut out of top and bottom so that the domain resembles a bow tie. We use parabolic inflow on the left edge and outflow on the right edge with $\mathbf{u}(x, y) = (-y^2 + y, 0)$. We take the no-slip conditions $\mathbf{u}(x, y) = (0, 0)$ elsewhere and let $\mathbf{f}(x, y) = (0, 0)$. Figure 2.8 shows a close-up of the computed unified Stokes solution and a comparison between unified Stokes and Taylor-Hood with $h = \frac{1}{32}$ and $k = 4$.

Mass conservation is shown in Table 2.7 and we see results similar to 2.7.1. The unified Stokes method takes many more iterations to push $\|\nabla \cdot \mathbf{u}\|_{L^2}$ below the threshold so that it is less efficient, although it converged in fewer iterations than the cross shaped domain. Taylor-Hood does not improve much as the mesh is refined and polynomial order increased. Using a very fine mesh and high-order polynomials, a standard laptop did not have enough memory to compute the Taylor-Hood solution.



(a) Unified Stokes computed velocity. (b) Difference between solutions.

Figure 2.8: Comparison of computed solutions with $h = \frac{1}{32}$ and $k = 4$ in a bow tie shaped domain.

2.8 Conclusion

We have described an algorithm for efficiently solving the Stokes system based on the Scott-Vogelius technique and the iterated penalty method. We used the iterated penalty method to compute a divergent-free velocity and a continuous function space to compute an accurate pressure. Additionally, this method is more efficient than a Taylor-Hood approach in some cases, which similarly computes a continuous, accurate pressure. We have given a full description of our implementation in FEniCS as well as numerical results that show that our algorithm is competitive. Furthermore, we study our method in complex domains where mass conservation is known to be important for numerical stability.

		Taylor-Hood	Unified Stokes			
$1/h$	k	$\|\nabla \cdot \mathbf{u}\|_{L^2}$	$\|\nabla \cdot \mathbf{u}^1\|_{L^2}$	$\|\nabla \cdot \mathbf{u}^4\|_{L^2}$	$\|\nabla \cdot \mathbf{u}^n\|_{L^2}$	n
16	2	1.33e+00	4.46e-02	2.00e-05	5.40e-11	9
16	3	1.00e+00	5.26e-02	7.09e-05	2.27e-11	12
16	4	5.93e-01	5.21e-02	6.73e-05	1.43e-11	11
16	5	5.00e-01	5.21e-02	6.70e-05	1.63e-11	11
16	6	4.18e-01	5.20e-02	6.63e-05	2.66e-11	11
32	2	1.15e+00	6.26e-02	1.43e-02	2.61e-03	100
32	3	7.94e-01	5.24e-02	7.22e-05	6.28e-11	15
32	4	5.93e-01	5.21e-02	6.74e-05	1.16e-11	11
32	5	4.73e-01	5.20e-02	6.66e-05	1.78e-11	11
32	6	3.92e-01	5.19e-02	6.61e-05	9.97e-12	11
64	2	7.99e-01	5.23e-02	7.53e-05	8.64e-11	22
64	3	5.33e-01	5.19e-02	6.61e-05	2.20e-11	11
64	4	4.02e-01	5.19e-02	6.57e-05	1.36e-11	11
64	5	3.21e-01	5.18e-02	6.56e-05	2.52e-11	11
64	6	2.67e-01	5.18e-02	6.55e-05	1.44e-11	11
128	2	5.57e-01	5.20e-02	1.67e-04	1.14e-05	100
128	3	3.69e-01	5.19e-02	6.57e-05	6.53e-08	100
128	4	2.77e-01	5.18e-02	6.55e-05	6.54e-08	100
128	5	2.21e-01	5.18e-02	6.54e-05	8.70e-09	100
128	6	1.84e-01	5.18e-02	6.54e-05	7.96e-09	100
256	2	3.70e-01	5.19e-02	6.77e-05	7.93e-06	100
256	3	2.48e-01	5.18e-02	6.55e-05	4.08e-08	100
256	4	1.87e-01	5.18e-02	6.54e-05	6.07e-10	100
256	5	1.49e-01	5.18e-02	6.54e-05	2.91e-10	100
256	6	1.23e-01	5.18e-02	6.54e-05	9.09e-11	35

Table 2.6: $\|\nabla \cdot \mathbf{u}\|_{L^2}$ for different test cases in the cross domain for Taylor-Hood and unified Stokes after one iteration, four iterations, and after termination and number of iterations used to satisfy the stopping condition.

		Taylor-Hood	Unified Stokes			
$1/h$	k	$\|\nabla \cdot \mathbf{u}\|_{L^2}$	$\|\nabla \cdot \mathbf{u}^1\|_{L^2}$	$\|\nabla \cdot \mathbf{u}^4\|_{L^2}$	$\|\nabla \cdot \mathbf{u}^n\|_{L^2}$	n
16	2	5.21e-01	2.38e-03	4.16e-05	4.19e-11	17
16	3	3.63e-01	1.71e-03	9.33e-09	5.62e-11	6
16	4	2.67e-01	1.68e-03	4.76e-09	2.52e-11	6
16	5	2.15e-01	1.66e-03	2.22e-09	1.09e-11	6
16	6	1.79e-01	1.65e-03	1.41e-09	9.67e-11	5
32	2	3.57e-01	2.01e-03	6.86e-05	5.47e-08	100
32	3	2.47e-01	1.81e-03	6.36e-06	6.98e-11	34
32	4	1.82e-01	1.79e-03	1.71e-07	7.21e-11	24
32	5	1.45e-01	1.79e-03	1.56e-07	9.74e-11	23
32	6	1.21e-01	1.78e-03	1.33e-08	7.33e-11	14
64	2	2.43e-01	1.95e-03	4.75e-05	3.57e-08	100
64	3	1.62e-01	1.84e-03	3.53e-07	3.79e-10	100
64	4	1.21e-01	1.83e-03	4.85e-07	9.06e-11	19
64	5	9.58e-02	1.83e-03	1.64e-07	7.96e-11	17
64	6	7.92e-02	1.82e-03	6.42e-08	9.15e-11	15
128	2	1.66e-01	1.86e-03	2.18e-05	1.43e-07	100
128	3	1.13e-01	1.84e-03	1.76e-07	7.02e-10	100
128	4	8.25e-02	1.84e-03	6.21e-08	9.04e-11	41
128	5		1.83e-03	2.90e-08	9.88e-11	39
128	6		1.83e-03	6.15e-08	8.12e-11	40

Table 2.7: $\|\nabla \cdot \mathbf{u}\|_{L^2}$ for different test cases in the bow tie domain for Taylor-Hood and unified Stokes after one iteration, four iterations, and after termination and number of iterations used to satisfy the stopping condition.

CHAPTER 3

A STOCHASTIC PERFORMANCE MODEL FOR PIPELINED KRYLOV METHODS

3.1 Introduction

Krylov methods [36] iteratively solve systems of equations $Ax = b$ by building a basis and looking for the solution x in the Krylov subspace

$$\mathcal{K}_m = \{b, Ab, A^2b, \ldots, A^{m-1}b\}$$

in iteration m. These subspaces are natural places to look for solutions to matrix equations since the inverse of a nonsingular matrix A^{-1} can be written as a linear combination of the powers of A through its minimum polynomial. Krylov methods have become an indispensable tool for the scalable solutions of large, sparse linear systems, which in turn have enabled an explosion in massively parallel scientific simulation [28]. Krylov methods use global operations such as vector norms and orthogonalizations in each iteration so that processors must perform in lockstep. As we move to larger massively parallel architectures, the latency cost for reduction operations, central to Krylov methods, has ballooned [17]. In an effort to control these costs, "pipelined" versions of many Krylov algorithms have been developed, which allow some of the latency cost to be hidden by computational work. One cannot arbitrarily remove synchronizations from algorithms and expect comparable behavior, so pipelined variants of Krylov methods employ rearrangements which give arithmetically equivalent methods with looser data dependencies and with possibility to overlap computation and global communication, at the cost of additional intermediate storage and local computation, increased latency as a pipeline is filled, and degraded numerical stability.

A pipelined version of the classical conjugate gradient (CG) method was already developed in [11] for vector machines, revisited for large scale parallelism in [21], and imple-

mented for field-programmable gate arrays in [44]. Similarly, pipelined versions of CG [15], GMRES [20] and BiCGStab [25] have also been put forward. We have included a GMRES algorithm (Algorithm 1) and a pipelined version by [20] (Algorithm 2) for completeness.

Algorithm 1 GMRES

$r_0 \leftarrow b - Ax_0$; $v_0 \leftarrow r_0/\|r_0\|_2$
for $i = 0, 1, \ldots, m-1$ **do**
  $z \leftarrow Av_i$
  $h_{j,i} \leftarrow \langle z, v_j \rangle, j = 0, 1, \ldots, i$
  $\tilde{v}_{i+1} \leftarrow z - \sum_{j=1}^{i} h_{j,i} v_j$
  $h_{i+1,i} \leftarrow \|\tilde{v}_{i+1}\|_2$
  $v_{i+1} \leftarrow \tilde{v}_{i+1}/h_{i+1,i}$
  # apply Givens rotations to $H_{:,i}$
end for
$y_m \leftarrow \text{argmin}\|(H_{m+1,m}y_m - \|r_0\|_2 e_1)\|_2$
$x \leftarrow x_0 + V_m y_m$

Algorithm 2 PGMRES

$r_0 \leftarrow b - Ax_0$; $v_0 \leftarrow r_0/\|r_0\|_2$; $z_0 \leftarrow v_0$
for $i = 0, 1, \ldots, m+1$ **do**
  $w \leftarrow Az_i$
  if i > 1 **then**
    $v_{i-1} \leftarrow v_{i-1}/h_{i-1,i-2}$
    $z_i \leftarrow z_i/h_{i-1,i-2}$
    $w \leftarrow w/h_{i-1,i-2}$
    for $j = 0, 1, \ldots, i$ **do**
      $h_{j,i-1} \leftarrow h_{j,i-1}/h_{i-1,i-2}$
    end for
    $h_{i-1,i-1} \leftarrow h_{i-1,i-1}/h^2_{i-1,i-2}$
  end if
  $z_{i+1} \leftarrow w - \sum_{j=0}^{i-1} h_{j,i-1} z_{j+1}$
  if i > 0 **then**
    $v_i \leftarrow z_i - \sum_{j=0}^{i-1} h_{j,i-1} v_j$
    $h_{i,i-1} \leftarrow \|v_i\|_2$
  end if
  $h_{j,i} \leftarrow \langle z_{i+1}, v_j \rangle, j = 0, 1, \ldots, i$
end for
$y_m \leftarrow \text{argmin}\|(H_{m+1,m}y_m - \|r_0\|_2 e_1)\|_2$
$x \leftarrow x_0 + V_m y_m$

It has been difficult to understand the performance of these solvers on existing machines, judge the impact of algorithmic tradeoffs, and predict performance on future architectures

due to the lack of a coherent performance model [23].

Some runs in [21] exhibit a speedup of slightly more than a factor of 2, which is difficult to explain in a deterministic setting. Operating system interrupts cause detours in computations, degrading performance in highly synchronous applications [24] and making them difficult to characterize. While some detours (e.g., timer updates) are periodic and predictable, others are not easy to predict, making it difficult to develop performance models for highly synchronous methods [6, 32]. Some work has been done using stochastic models to predict performance in HPC environments. For example, [1] studied the impact of different noise distributions on a single collective operation and [42] modeled operating system "jitter" as random variables in order to compute computational slowdown in the presence of noise.

In this work, we present a stochastic performance model for traditional and pipelined Krylov solvers, examine the implications of different waiting time distributions on algorithm performance, and clarify a "folk theorem" which says that speedup gained from pipelined methods is bound by $2\times$. In Section 3.2, we derive performance models and study them under different analytical distributions. In Sections 3.3, 3.4, and 3.5, we describe an experimental environment and use PETSc runs to verify the models and study the noise distribution for various simulations. A conclusion is given in Section 3.6.

3.2 Mathematical Model

We model a Krylov iterative method as a set of P communicating processes who must perform a calculation consisting of local computations, separated by periodic global synchronizations, and interrupted by waiting, perhaps due to unsatisfied requests to memory, actions of the operating system, etc. We will label the set of computations and waiting by the index k, which corresponds to the iteration number for the Krylov method. The removal of the global synchronizations will correspond to the introduction of split-phase collectives [23] for the norm calculation and orthogonalization step. A split-phase, or non-blocking, collective

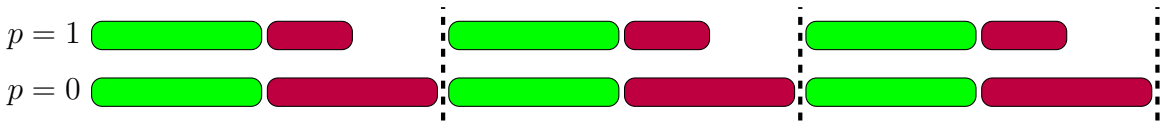


Figure 3.1: Computation with $P = 2$ processes for $K = 3$ steps. The green rectangles represent computation, the purple waiting, and the dotted lines are synchronization.

is a collective operation, such as a broadcast, which has been split into two parts so that it no longer requires a global synchronization at the point of the call. Rather, the collective operation is first initiated, say with `MPI_Ibcast()`, and then later finalized with `MPI_Wait()`. This strategy removes the global synchronization and allows the operation to be overlapped with useful work between the initialization and finalization. The ratio between times with and without synchronization will give us a bound on the speedup of the pipelined algorithm over the classical variant. We illustrate this model using $P = 2$ processes.

3.2.1 Deterministic Computation and Waiting Times

In the simplest scenario, each process takes a certain time c_p for local computation, w_p for waiting, which is independent of the step k. Fig. 3.1 represents computation with three steps. The total running time T (or makespan) of the computation is then given by the expression

$$T = \sum_k \max_p (c_p + w_p) = K \max_p T_p, \tag{3.1}$$

where K is the total number of steps, and $T_p = c_p + w_p$ is the time on process p for one step excluding waiting for the global barrier. Clearly, without loss of generality, we can replace the separate computation and waiting timing with a single process time T_p.

If we remove the synchronizations, as shown in Fig. 3.2, then the makespan is given by

$$T' = \max_p \sum_k (c_p + w_p) = K \max_p T_p, \tag{3.2}$$

so that no speedup is achievable. The removal of synchronizations can in general be modeled by the interchange of the sum over steps and the maximum over process times.

3.2.2 Stochastic Process Times

If we allow the amount of local computation and waiting to fluctuate over the steps, as shown in Fig. 3.3, we will see that speedup is achievable, as shown in Fig. 3.4. Most simulation codes, including most linear solvers, statically partition the data so that computation times do not show large fluctuations. Waiting times, however, are more volatile and fluctuate across processes and steps [43], which can arise from interactions with the OS [18].

In the very simple scenario that one process waits for a long time W on the first step, the other on the second, and on other steps the processes both take time T_0, as shown in Fig. 3.3, then the makespan with and without synchronizations is given by

$$T = \sum_k \max_p T_p = 2W + KT_0, \tag{3.3}$$

$$T' = \max_p \sum_k T_p = W + KT_0. \tag{3.4}$$

Thus the possible speedup is

$$\frac{T}{T'} = \frac{2W + KT_0}{W + KT_0} = \frac{2 + \alpha}{1 + \alpha} \tag{3.5}$$

where $\alpha = KT_0/W$, which is bounded above by 2. Extended to P processes this gives an upper bound of P on the speedup, since all the waiting time is collapsed to the first interval when synchronization is removed and computation time becomes small. This is related to

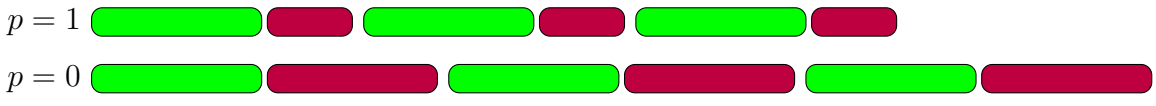


Figure 3.2: Computation with $P = 2$ processes for $K = 3$ steps. The green rectangles represent computation, the purple waiting, and there is no synchronization.

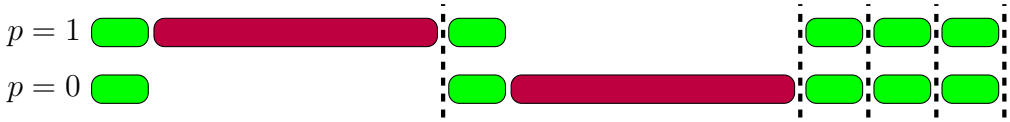


Figure 3.3: Computation with $P = 2$ processes for $K = 5$ steps. The green rectangles represent computation, the purple waiting, and the dotted lines are synchronization.

Figure 3.4: Computation with $P = 2$ processes for $K = 5$ steps. The green rectangles represent computation, the purple waiting, and there is no synchronization.

the folklore result that speedup from covering communication with computation is limited to a factor of 2, where we have two players, computing and communicating. This is because if we cover all communication by computation, then one is larger than the other and is more than half of the computational time.

We will employ a stochastic description of the waiting time variation making it amenable to analysis by letting the time of iteration k on process p be a random variable. We begin with a stochastic process time $\mathcal{T}_p^k$ at each step that is drawn from a distribution independent of process and stationary in step number. Let T be the total Krylov time (computation and waiting) of the classical algorithm with synchronization, and T' the total time for the pipelined version without synchronization where $T = \sum_k \max_p \mathcal{T}_p^k$ and $T' = \max_p \sum_k \mathcal{T}_p^k$. We can ask for the expected total time with and without synchronizations,

$$E[T] = E[\sum_k \max_p \mathcal{T}_p^k] = \sum_k E[\max_p \mathcal{T}_p^k], \tag{3.6}$$

and

$$E[T'] = E[\max_p \sum_k \mathcal{T}_p^k]. \tag{3.7}$$

These stage times will be modeled as random variables, drawn from some underlying dis-

tribution. Since the time spent computing should not fluctuate, within our measurement accuracy, it only affects the mean of the distribution. The variability in the distribution models the waiting times which can arise from interactions with the OS [43, 18]. We assume that waiting times across processes are independent because we have no expectation that OS operations will be correlated across processes. In this section, we compute the ratio of these quantities for a range of representative distributions for waiting times, in order to estimate the potential speedup.

3.2.3 Stochastic performance models

We would like to calculate the speedup after k steps, $\frac{E[T]}{E[T']}$, where expected total time $E[T]$ is defined by (3.6) and likewise $E[T']$ is defined by (3.7). First, we will derive an expression for the expected value of the maximum of a set of random variables in order to find an expression for $E[T]$. Then we will find one for $E[T']$.

A general expression for the expected value of the maximum of a set of random variables can be found in [34] and is described here for completeness. Let $X_1, X_2, \ldots, X_n$ be independent, identically distributed (iid) random variables with probability distribution function (pdf) $f(x)$ and cumulative distribution function (cdf) $F(x)$. Let $X_{\max}$ be another random variable such that $X_{\max} = \max\{X_1, X_2, \ldots, X_n\}$. By noticing that $X_{\max} \leq x$ if and only if $X_i \leq x$ for all i's,

$$\begin{aligned} F_{\max}(x) &= P(X_{\max} \leq x) \\ &= P(X_1 \leq x, X_2 \leq x, \ldots, X_n \leq x) \\ &= P(X_1 \leq x)P(X_2 \leq x)\ldots P(X_n \leq x) \\ &= F(x)F(x)\ldots F(x) \\ &= F(x)^n \end{aligned}$$

using the fact that the X_i's are independent and identically distributed. Furthermore,

$$f_{\max}(x) = \frac{d}{dx}F_{\max}(x) = \frac{d}{dx}F(x)^n = nF(x)^{n-1}f(x)$$

since by definition $\frac{d}{dx}F(x) = f(x)$. The expected value of $X_{\max}$, integrating over the support of x, is

$$E[X_{\max}] = n\int_{-\infty}^{\infty} xF(x)^{n-1}f(x)dx. \tag{3.8}$$

At step $k = 1$, we have P iid random variables $\mathcal{T}_0^1, \mathcal{T}_1^1, \ldots, \mathcal{T}_{P-1}^1$ from an underlying distribution with pdf $f_1(x)$, cdf $F_1(x)$, and joint cdf $F(x_0, x_1, \ldots, x_{P-1})$. Similarly, for a given step k, $\mathcal{T}_0^k, \mathcal{T}_1^k, \ldots, \mathcal{T}_{P-1}^k$ are iid random variables with pdf $f_k(x)$, cdf $F_k(x)$, and joint cdf $F(x_0, x_1, \ldots, x_{P-1})$. Since the random variables are stationary in k, the joint cdf and thus the individual pdfs and cdfs remain the same at step k so that $f_1(x) = f_k(x)$ and $F_1(x) = F_k(x)$. Then using (3.8), we have

$$E[\max_p \mathcal{T}_p^1] = P\int_{-\infty}^{\infty} xF_1(x)^{P-1}f_1(x)dx = E[\max_p \mathcal{T}_p^k].$$

Then

$$E[T] = \sum_k E[\max_p \mathcal{T}_p^k] = KE[\max_p \mathcal{T}_p^1]. \tag{3.9}$$

Also because the random variables $\mathcal{T}_p^k$ are stationary in k, $E[\mathcal{T}_p^k] = \mu$ for all k. The sum $\sum_k \mathcal{T}_p^k$ will approach $K\mu$ in the limit of large K so that we also have

$$E[T'] = E\big[\max_p \sum_k \mathcal{T}_p^k\big] \to K\mu. \tag{3.10}$$

In this case, speedup is given by $\frac{E[T]}{E[T']} \to \frac{E[\max_p \mathcal{T}_p^1]}{\mu}$.

We will examine a of range of common analytical distributions. As the tails of $f_k(x)$ become heavier, the potential speedup increases and can eventually exceed $2\times$.

3.2.4 Uniform Distribution

Let the $\mathcal{T}_p$'s from P processes be independent random variables from a uniform distribution on $[a, b]$ with pdf $f(x) = \frac{1}{b-a}$, cdf $F(x) = \frac{x-a}{b-a}$, and mean $\mu = \frac{a+b}{2}$. Using (3.8), we calculate the expected value of the maximum

$$\begin{aligned} E[\max_p \mathcal{T}_p] &= P \int_a^b x \left(\frac{x-a}{b-a}\right)^{P-1} \frac{1}{b-a} dx \\ &= \frac{a+Pb}{P+1} \end{aligned}$$

to find speedup on P processes

$$\frac{E[T]}{E[T']} = \frac{\frac{a+Pb}{P+1}}{\frac{a+b}{2}} = \frac{2(a+Pb)}{(P+1)(a+b)}.$$

On the interval $[0, b]$, we find that the asynchronous speedup, $\frac{2P}{P+1}$, is bounded from above by 2.

3.2.5 Exponential Distribution

Let $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$, the times for four processes, be independent random variables from an exponential distribution with pdf $f(x) = \lambda e^{-\lambda x}$, cdf $F(x) = 1 - e^{-\lambda x}$, and mean $\mu = \frac{1}{\lambda}$.

Again, using (3.8), we calculate the expected value of the maximum

$$\begin{aligned} E[\max_p \mathcal{T}_p] &= 4\lambda \int_0^\infty x\big(1-e^{-\lambda x}\big)^3 e^{-\lambda x} dx \\ &= \frac{25}{12\lambda}. \end{aligned}$$

We find that the speedup on four processes is

$$\frac{E[T]}{E[T']} = \frac{25/12\lambda}{1/\lambda} = \frac{25}{12} > 2.$$

When the $\mathcal{T}_p$'s are from an exponential distribution, asynchronous speedup is greater than 2 on four or more processes. Furthermore, the speedup on P processes

$$\frac{E[T]}{E[T']} = \frac{E[\max_p \mathcal{T}_p]}{\mu} = \frac{\lambda P \int_0^\infty x\big(1-e^{-\lambda x}\big)^{P-1} e^{-\lambda x} dx}{1/\lambda} = H_P.$$

Here, $H_P = \log P + \gamma + O(1/P)$ is the Pth harmonic number and γ is Euler's constant [39].

3.2.6 Log-normal Distribution

Let the $\mathcal{T}_p$'s from P processes be independent random variables from a log-normal distribution with pdf $f(x) = \frac{1}{x\sqrt{2\pi}\sigma} e^{\frac{-(\ln(x)-\mu)^2}{2\sigma^2}}$, cdf $F(x) = \frac{1}{2} + \frac{1}{2}\operatorname{erf}\big(\frac{\ln(x)-\mu}{\sqrt{2}\sigma}\big)$, and mean $\mu' = e^{\mu+\frac{\sigma^2}{2}}$. Note that if a random variable X is normally distributed with mean μ and variance σ, then $Y = e^X$ is a random variable from a log-normal distribution with mean μ'. After fixing P, μ, and σ, we can calculate the speedup from P processes numerically using equation (3.8) and Octave's `quad` function. First, let $P = 2$, $\mu = 0$, and $\sigma = 1$. Equation (3.8) becomes

$$E[\max_p \mathcal{T}_p] = 2\int_0^\infty x\left(\frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{\ln(x)}{\sqrt{2}}\right)\right)\left(\frac{1}{x\sqrt{2\pi}} e^{\frac{-(\ln(x))^2}{2}}\right) dx \approx 2.5069,$$

so that the speedup on two processes is

$$\frac{E[T]}{E[T']} = \frac{E[\max_p \mathcal{T}_p]}{\mu'} \approx \frac{2.5069}{\sqrt{e}} \approx 1.5205.$$

Now let $P = 4$, $\mu = 0$, and $\sigma = 1$. Equation (3.8) becomes

$$E[\max_p \mathcal{T}_p] = 4 \int_0^\infty x \left(\frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{\ln(x)}{\sqrt{2}} \right) \right)^3 \left(\frac{1}{x\sqrt{2\pi}} e^{\frac{-(\ln(x))^2}{2}} \right) dx \approx 3.6406.$$

We again find that on four processors, the potential speedup is greater than 2:

$$\frac{E[T]}{E[T']} \approx \frac{3.6406}{\sqrt{e}} \approx 2.2081 > 2.$$

3.3 Experimental Results

The speedups reported in [21] for PIPECG and PIPECR on an Intel Xeon cluster using Infiniband seem limited by 2× speedup, but at the limit of 20 processes exceed this slightly (2.09 and 2.14 respectively). However, understanding the origin of speedup exceeding 2× requires the examination of many identical runs and collection of per iteration per processor runtimes in order to amass statistics. The PIPECG and PGMRES algorithms are similar in that work involving sparse matrix-vector products (SpMV) is juxtaposed with work involving vector dot products. The dot products are reductions which require some sort of global synchronization. Thus the computational portion of our model is associated with the SpMV and orthogonalization with the BLAS AXPY calls, whereas the synchronizations apply to the dot product portion. These algorithms decouple these two operations, so that as long as the SpMV is more expensive than a global synchronization, the dot product operation involves negligible waiting.

In our experiments, we will use PETSc tutorials ex23 and ex48. Tutorial ex23 uses a simple tridiagonal system which is a one-dimensional discretization of the Laplacian. PETSc

ex48 solves the hydrostatic, that is Blatter-Pattyn, equations for ice sheet flow, where the ice uses a power-law rheology with Glen exponent 3. This generates a much denser system of equations with about 10x more nonzeros per row than ex23.

In Section 3.4 we look at the statistics for repeated runs of ex23, further stressing the need for a nondeterministic performance model and in Section 3.5 we collect per iteration per processor data to verify our model and study the noise present in our simulations.

3.4 Repeated runs

We have generated repeated runs for PETSc KSP tutorial ex23 using CG, PIPECG, GMRES, and PGMRES on 8192 processors of the Piz Daint Cray XC30 supercomputer at CSCS [14], storing the data in an open repository [37]. We force 5000 iterates of the Krylov method on a tridiagonal system of size 2,097,152. The pipelined methods produce almost identical residuals to the original methods for this problem. Most of the runtime for CG and PIPECG is concentrated in dot products, the `VecTDot()` operation in PETSc, rather than in SpMV. This means there is no computation to cover the communication cost, and very often we see no speedup, or even slowdown, for these runs. In [21], PIPECG achieves 2× speedup for SNES tutorial ex48 because the much denser matrices in ex48 provide enough computation to cover the communication costs. However, we see some outliers in the data where speedup exceeds 2×. This could potentially be explained by assuming a noise distribution and using the prior results from Sec. 3.2. Summary statistics for the GMRES, PGMRES, CG, and PIPECG runs are given in Table 3.1. It appears that some level of system noise is present in these simulations.

3.5 Fine-grained analysis

We also run PETSc tutorials ex23 and ex48 using GMRES and PGMRES on the Cray XC40 supercomputer Theta at the Argonne Leadership Computing Facility and use the data to

	GMRES	PGMRES	CG	PIPECG
$\bar{x}$	0.9465	0.5902	0.9349	0.7521
median	0.9932	0.5856	0.8632	0.6792
s	0.1303	0.0962	0.2385	0.2429
s^2	0.0170	0.0092	0.0569	0.0590
λ	1.0565	1.6942	1.0696	1.3295
$X_{\min}$	0.6617	0.4644	0.6051	0.5545
$X_{\max}$	1.0740	0.7697	1.6060	1.6950

Table 3.1: PGMRES and PIPECG runtime statistics

verify the Krylov and pipelined Krylov methods in Section 3.2.

3.5.1 PETSc KSP tutorial ex23

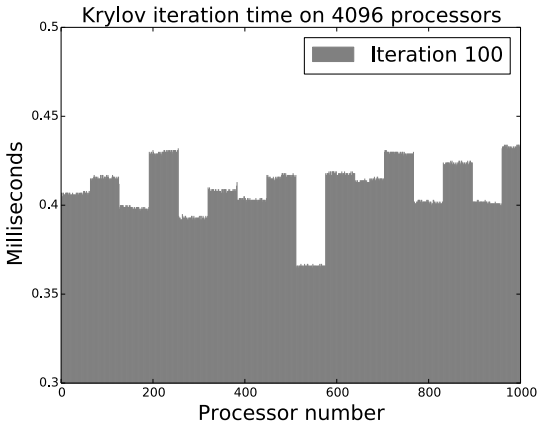
We analyze one run of PETSc tutorial ex23 using 5000 iterations of Krylov solver GMRES on 4096 processors on Theta. This run was submitted Friday September 15, 2017 at 2:26PM CST and began less than one minute later.

To collect per iteration per processor runtimes, we add calls to `MPI_Wtime()` at the beginning and end points of the while loop in `KSPGMRESCycle()` in the PETSc GMRES solver and stored iteration times in an array. IO was done after the solve was complete. We show that the model (3.1) for the total runtime T and our data collection is consistent. We calculate

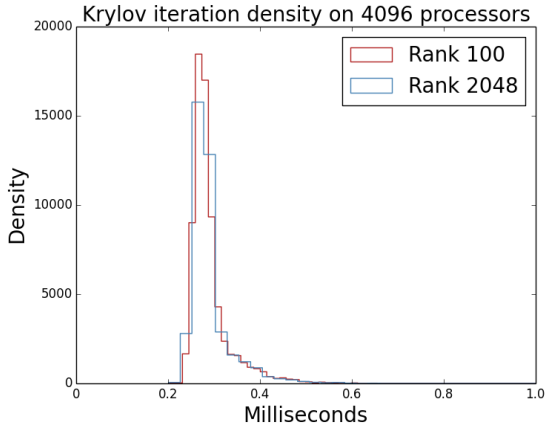
$$T = \sum_k \max_p T_p^k = 1.643$$

from the collected data compared to $T = 1.514$, the KSPSolve given in the PETSc `-log_view` file.

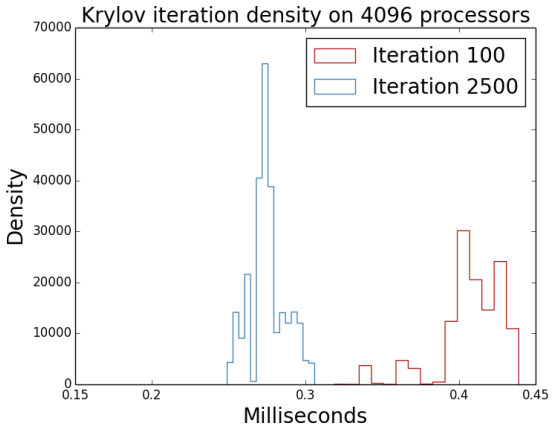
The stochastic model (3.9) assumes that the runtimes $\mathcal{T}_p^k$ are independent and identical with respect to process p and identical with respect to iteration k. Figure 3.5 shows a visualization of these assumptions. Although we do not expect OS noise to be correlated across processes, Figure 3.5a suggests that the runtimes $\mathcal{T}_p^k$ are highly dependent on the node to which processor p belongs. In particular, we notice that the runtimes for process p on node n are approximately equal so that $\mathcal{T}_{p_1}^k \approx \mathcal{T}_{p_2}^k$ if p_1 and p_2 are on the same node. Since



(a) Independent with respect to processor.

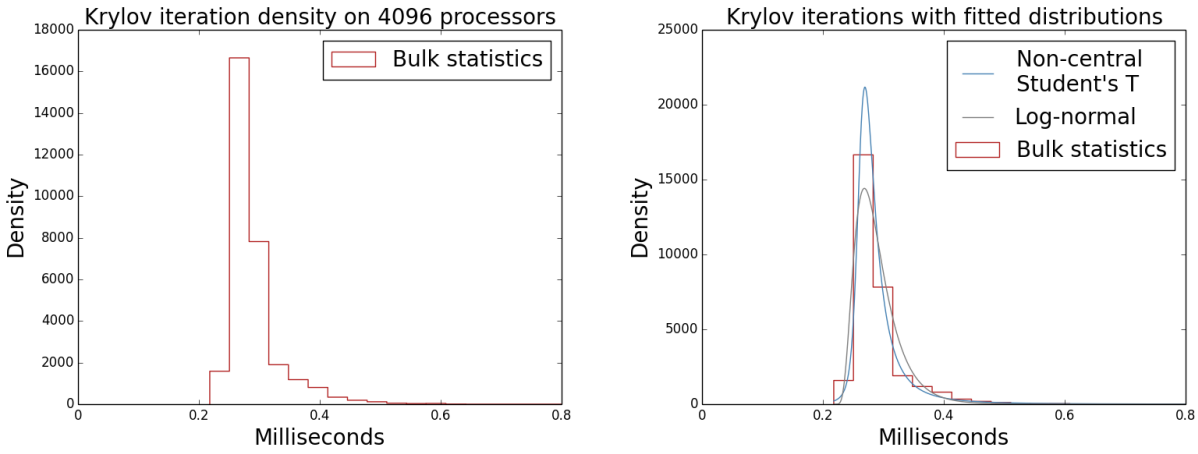


(b) Identical with respect to processor.



(c) Stationary in time.

Figure 3.5: Assumptions on $\mathcal{T}_p^k$ for stochastic Krylov model.



(a) Histogram of aggregate runtimes.

(b) Distribution fitting using `scipy`.

Figure 3.6: Aggregate runtimes $\mathcal{T}_p^k$ for stochastic Krylov model.

each node on Theta has 64 cores, we will model P dependent processes as $\frac{P}{64}$ independent processes. Figure 3.5b shows that it is reasonable to assume that the runtimes are identical with respect to process. That is, $f_{\mathcal{T}_{p_1}^k} \approx f_{\mathcal{T}_{p_2}^k}$ for processors p_1 and p_2. For a more rigorous argument, future work will include results from a Two-sample KolmogorovSmirnov test which can be used to test whether or not the underlying distribution of two data sets differs. From Figure 3.5c, it is clear that the iterations are not stationary in time. This is more problematic since we would like to drop the sum in (3.9) in favor of multiplication. To circumvent this issue, we could take a "representative" iteration k as a stand in, but at this time we have not identified a pattern for the shifts in iteration length.

Instead, we will take bulk statistics over all iterations and use the aggregate runtimes to find a suitable distribution. Figure 3.6a shows a normalized histogram of $\mathcal{T}_p^k$ for all k and p. We use Python's `scipy` package for distribution fitting and fit available distributions to our data. We calculate the sum of squared errors between our data and a fitted distribution and pick the "best" distribution that minimizes the error. The bulk data can be reasonably represented by different distributions, shown in Figure 3.6b. It's worth noting that a handful of distributions were good fits, including an inverse Gaussian, right-skewed Gumbel, Johnson SB, and others.

Analytical bounds were used in [42] that bound the maximum of a set of random variables $X = \{x_1, x_2, \ldots, x_N\}$ using the sample mean μ_X and sample standard deviation σ_X. Cramer [12] bound the maximum of N identical and independent random variables by

$$E[X_{\max(N)}] \leq \mu_X + \frac{\sigma_X(N-1)}{\sqrt{2N-1}}$$

and Bertsimas [7] bound the maximum of N identical, but not independent random variables by

$$E[X_{\max(N)}] \leq \mu_X + \sigma_X\sqrt{N-1},$$

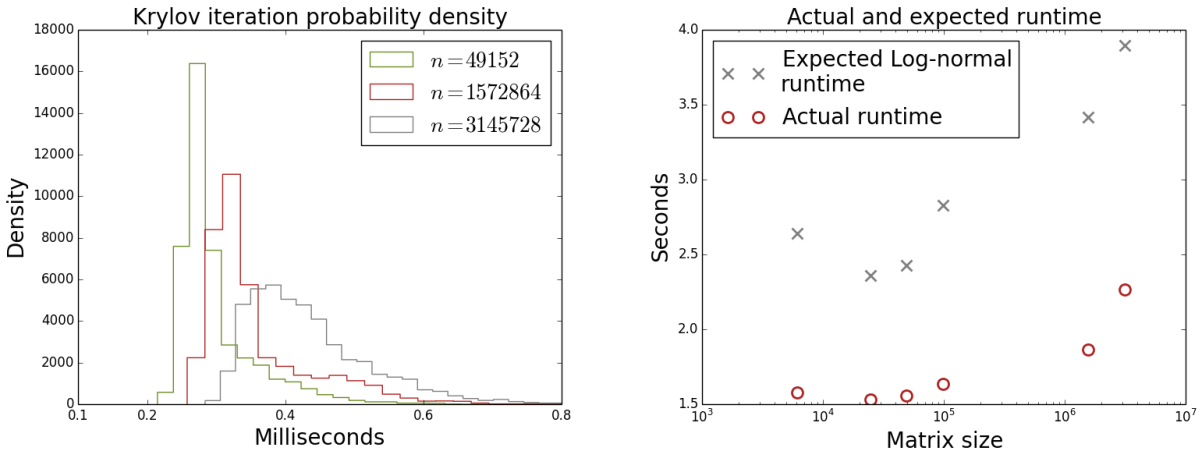
which corresponds to our situation.

We compute $E[T]$ using our integral formulation (3.8) and model (3.9) with fitted distributions, using Python's `quad` function and integrating over the bounds of the data. Using the bulk statistics to calculate μ_X and σ_X with $N = 64$, we also compute the Cramer and Bertsimas bounds. Results are shown in Table 3.2. The actual runtime was $T = 1.514$ seconds.

We want to see how features of a computation will influence the underlying runtime distribution and use our model to calculate the expected runtime. To do this, we collect data from runs of PETSc tutorial ex23 varying the matrix size and number of processors used, still with 5000 iterations of GMRES on Theta at Argonne's ALCF. We see how the bulk iteration time distribution changes on a fixed number of nodes as we change the problem size (Figure 3.7) and on a varying number of nodes with a fixed amount of work per node

		$E[T]$
distribution	NCT	2.956
	Log-normal	2.097
bound	Cramer	2.935
	Bertsimas	3.558

Table 3.2: Expected runtimes using stochastic Krylov model and analytical bounds.



(a) Histograms of aggregate runtimes.

(b) Expected and actual runtimes.

Figure 3.7: Problem size scaling results.

(weak scaling, Figure 3.8). Increasing the problem size and number of nodes increases the variability of the runtimes and flattens out the distribution. There was no distribution that was the best fit in each case, so we use Log-normal to calculate $E[T]$ for all our experiments. The model consistently overestimates the runtime for our experiments and seems to degrade as the number of processors increases.

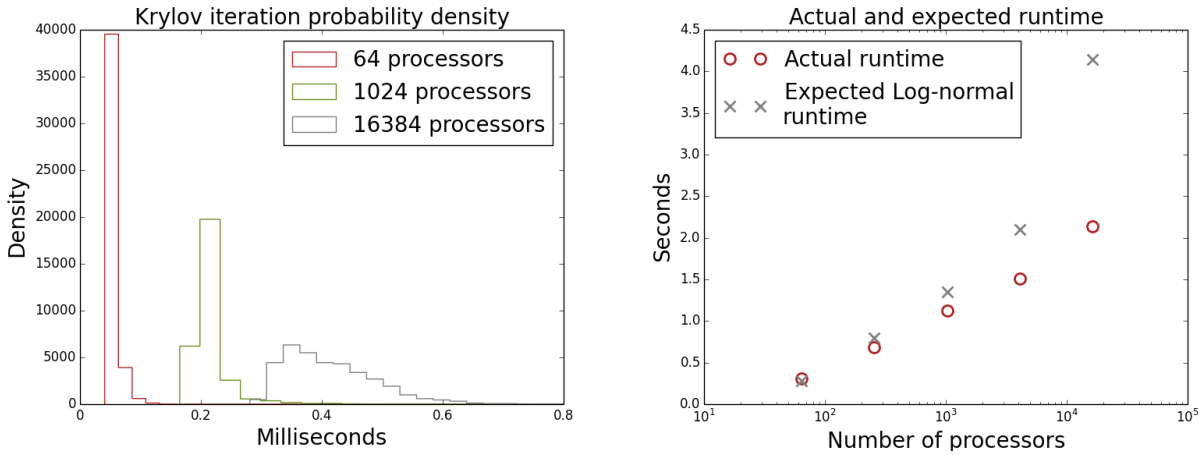
We repeat this analysis with data from one run of PETSc tutorial ex23 using 5000 iterations of Krylov solver PGMRES on 4096 processors on Theta at Argonne's ALCF. Data was collected in a similar way, using calls to `MPI_Wtime()` in the for loop of PETSc's `KSPPGMRESCycle()`. The run was submitted Wednesday February 28, 2018 at 2:46PM CST and began about 21 minutes later.

We check that the model (3.2) for the total runtime T' is of a pipelined method reasonable. From out data, we have

$$T' = \max_p \sum_k T_p^k = 1.843$$

compared to $T' = 1.910$, the time given for KSPSolve in the PETSc `-log_view` file.

The stochastic model (3.10) again assumes that the iteration times $\mathcal{T}_p^k$ are independence in p, identicalness in p, and stationary in k, as before, shown in Figure 3.9. Runtimes $\mathcal{T}_p^k$ for a fixed k are not dependent on node than we saw in the traditional Krylov case. As we



(a) Histograms of aggregate runtimes.

(b) Expected and actual runtimes.

Figure 3.8: Weak scaling results.

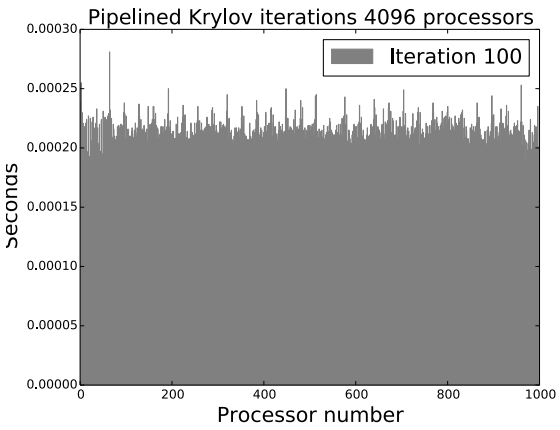
saw before, the runtimes appear to be identical with respect to processor (Figure 3.9b), but not stationary in time (Figure 3.9c). We use the bulk statistics again to find the underlying distribution of process times, excluding the iterations that fill the PGMRES pipeline (two for every restart). Like before, a number of distributions are good fits, including exponential and Log-normal, shown in Figure 3.9d.

Using our stochastic model, we calculate the expected runtime $E[T']$ using the good fitting Log-normal and exponential distributions, shown in Table 3.3. The total time for this run was $T' = 1.910$. This model is better approximation than the traditional Krylov stochastic model.

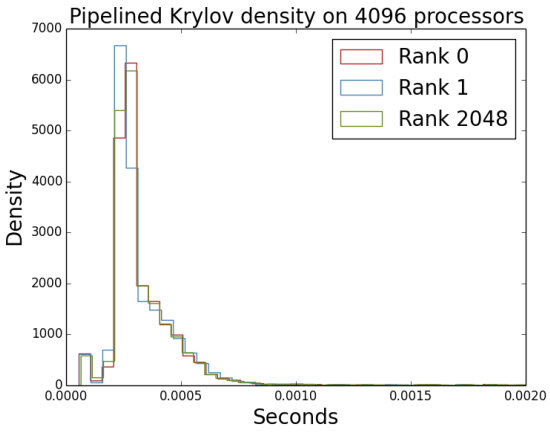
We perform problem scaling and weak-scaling experiments like before and use our model again to predict $E[T']$. Figure 3.10 shows the results of the stochastic model using a Log-normal distribution compared to actual runtimes. Below are some results from varying the amount of work done by process and a weak scaling experiment using a Log-normal

distribution	$E[T']$
Exponential	1.784
Log-normal	1.709

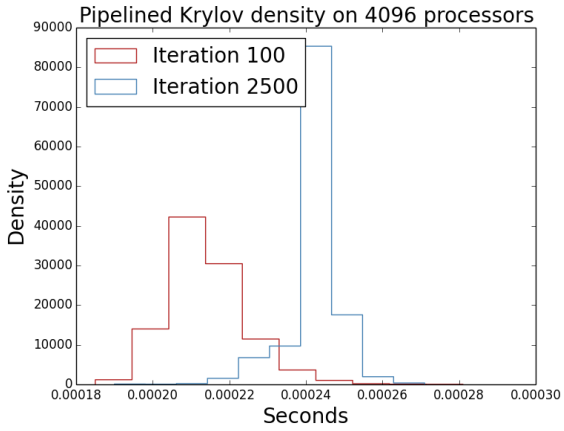
Table 3.3: Expected runtimes using stochastic pipelined Krylov model.



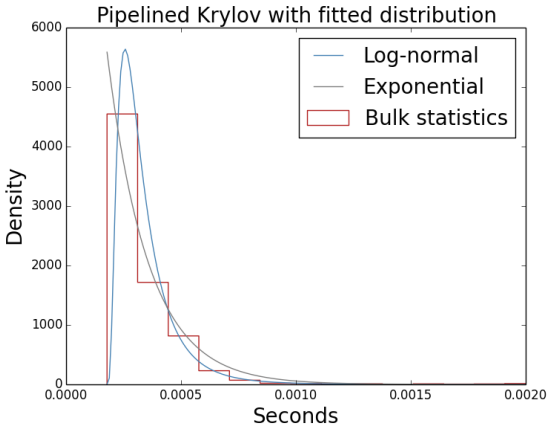
(a) Independent with respect to processor.



(b) Identical with respect to processor.

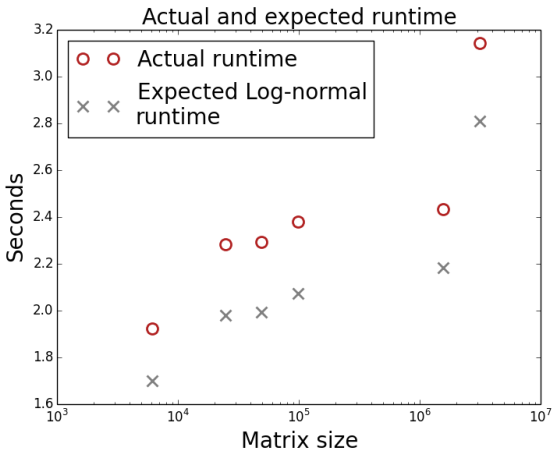


(c) Independent with respect to processor.

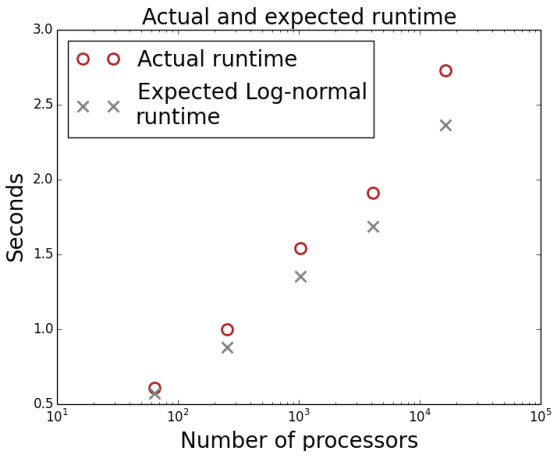


(d) Fitted distributions.

Figure 3.9: Assumptions on $\mathcal{T}_p^k$ for stochastic Krylov model.



(a) Problem size experiments.



(b) Weak-scaling experiments.

Figure 3.10: Actual and calculated expected runtimes for experiments.

distribution to model the runtimes. The model does a good job of predicting the execution time.

3.5.2 PETSc SNES tutorial ex48

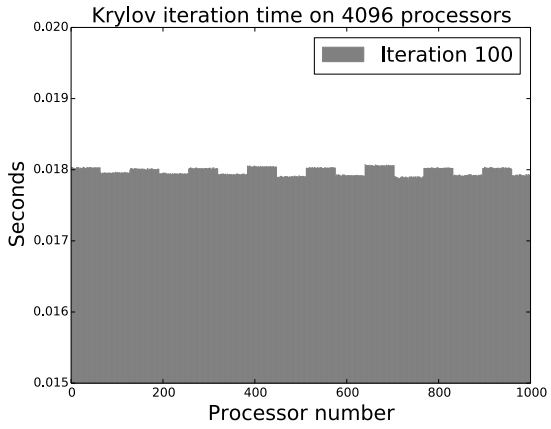
Now we will look at data from one run of PETSc SNES tutorial ex48 using 5000 iterations of Krylov solver GMRES on 4096 processors on the Cray XC40 Theta at Argonne's ALCF. This run was submitted Thursday September 7, 2017 at 10:17PM CST and began about 2 minutes later. We expect the results to be different from ex23 given that the sparsity pattern of the matrix in ex48 is complex, complicating the communication between processors during a sparse matrix vector multiplications, for example.

Using our data collection and our model, we have

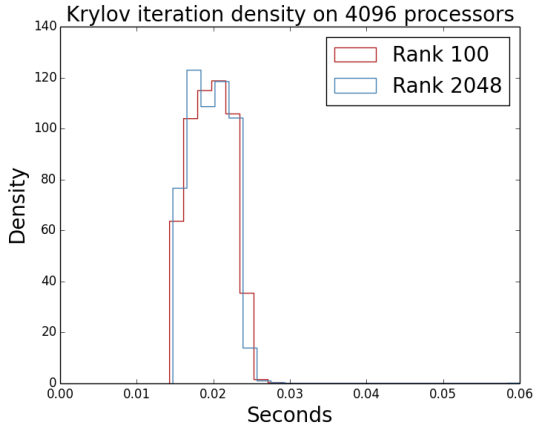
$$T = \sum_k \max_p T_p^k = 99.178$$

compared to the actual $T = 102.00$ seconds, the time given for KSPSolve in the PETSc `-log_view` file.

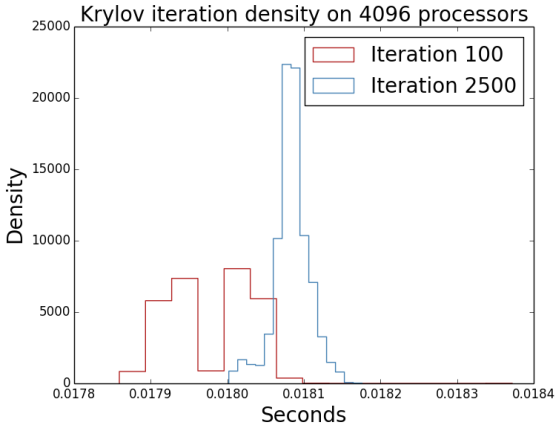
Figure 3.11 shows visual representations the assumptions on our data as before. Runtimes



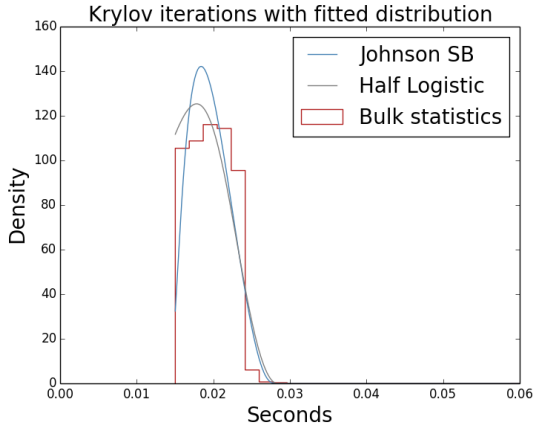
(a) Independent with respect to processor.



(b) Identical with respect to processor.



(c) Stationary in time.



(d) Fitted distributions.

Figure 3.11: Assumptions for stochastic Krylov model and fitted distributions.

$\mathcal{T}_p^k$ for a fixed k are still dependent on node, but there is much less variation between the nodes. Still, we will use $P = 64$ independent variables to model 4096 dependent processors as before for consistency. The runtimes are not stationary in time.

We use the bulk statistics to find the underlying distribution of process times, shown in Figure 3.11d. The runtime distribution is very different than what we found for ex23. The runtimes are grouped and there are fewer outliers. Not surprisingly, we find different distributions when fitting this data. We calculate the expected runtime $E[T]$ again using the good fitting distributions and analytical bounds, shown in Table 3.4. The total time for this run was $T = 102.00$ seconds. Like before, our model consistently overestimates the actual

		$E[T]$
distribution	Half Logistic	130.099
	Johnson SB	128.390
bound	Cramer	210.632
	Bertisimas	257.719

Table 3.4: Expected runtimes using stochastic Krylov model and analytical bounds.

distribution	$E[T']$
Semicircular	106.235
Johnson SB	104.894

Table 3.5: Expected runtimes using stochastic pipelined Krylov model.

execution time, possibly because of the unrealistic assumptions made to simplify the model.

Now we will look at data from one run of PETSc tutorial ex48 using 5000 iterations of Krylov solver PGMRES on 4096 processors on Theta at Argonne's ALCF. This run was submitted Thursday March 1, 2018 at 3:30PM CST and began about 23 minutes later.

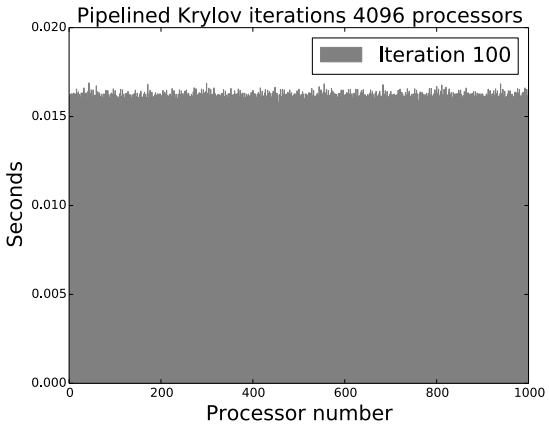
Again, we check the actual execution time T' with our data and model. We have

$$T' = \max_p \sum_k T_p^k = 107.851$$

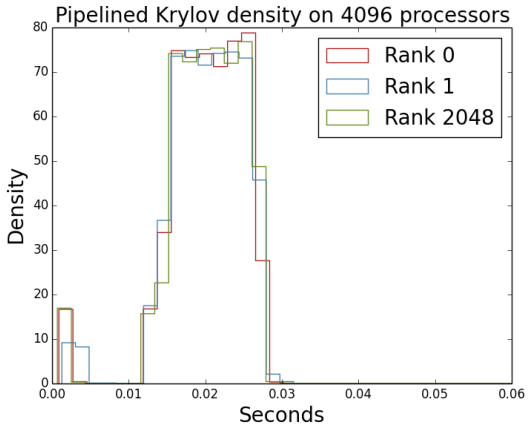
calculated from the collected data compared to $T' = 111.09$ seconds, the actual KSPSolve time.

Figure 3.12 shows the assumptions on the data, which do not appear to be dependent on which node a process belongs to, and which are again not stationary in time. In Figure 3.12b, the group of very fast runtimes belong to iterations which fill the pipeline at the beginning of each restart.

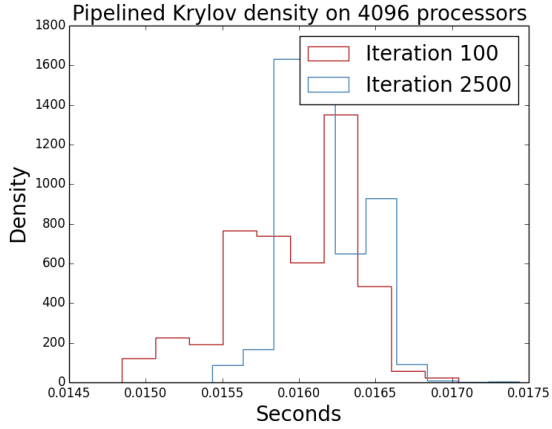
Figure 3.12d shows two distributions fit to the aggregate data that we will use to calculate the expected runtime $E[T']$. The results are shown in Table 3.5. The total time for this run was $T' = 111.09$ seconds.



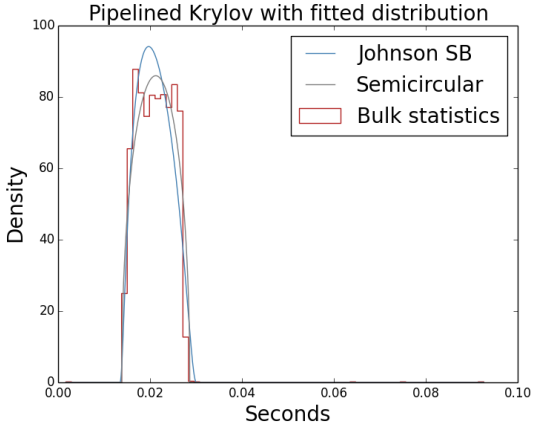
(a) Independent with respect to processor.



(b) Identical with respect to processor.

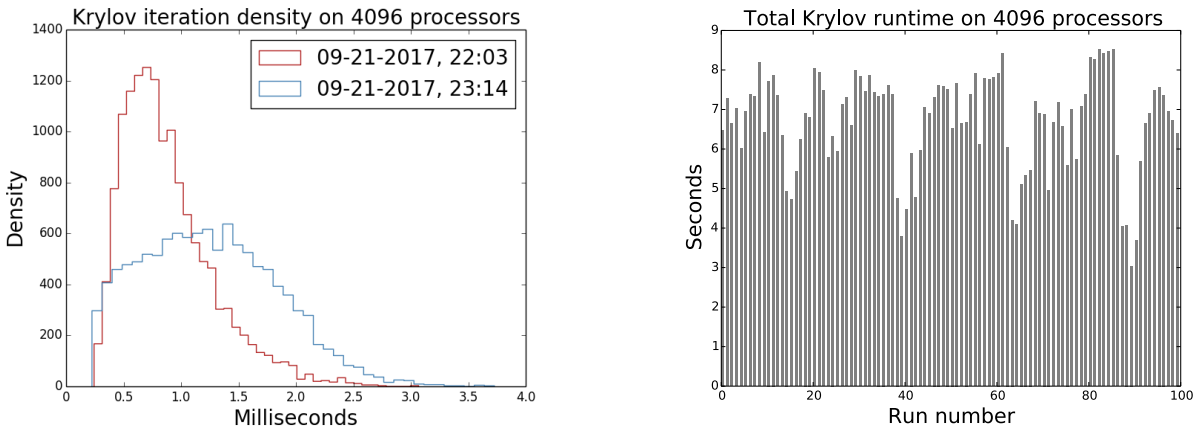


(c) Stationary in time.



(d) Fitted distributions.

Figure 3.12: Assumptions for stochastic pipelined Krylov model and fitted distributions.



(a) Independent with respect to processor.

(b) Identical with respect to processor.

Figure 3.13: Assumptions for stochastic pipelined Krylov model and fitted distributions.

3.5.3 Complications

Ultimately, we would like to be able to calculate the expected runtime for a Krylov or pipelined Krylov computation under some specific conditions (e.g. problem size and sparsity pattern, number of processors used). However, there is evidence that there are features of a computation that are difficult to capture with a stochastic description (e.g. machine state, physical nodes used). In this subsection, we will demonstrate some of these complications.

Figure 3.13a shows aggregate data from two identical runs of ex23 using GMRES on 4096 processors submitted with the same script on the same day about an hour apart. Despite their similarities, the results are very different. One possible explanation could be the "busyness" of the machine during the separate runs, since they sat the in queue for different lengths of time before execution (the run at 22:03 waited for approximately 50 minutes and 23:14 waited for about 30 minutes).

Figure 3.13b shows 100 runs of ex23 using GMRES on 4096 processors, where each bar is the total KSPSolve time for one run. These runs were submitted on a single script and executed in a loop so that they were allocated the same nodes. However, the runtimes show periodic behavior on a time scale larger than the simulation. This behavior could impact, for example, repeated Krylov solves during Newton's method.

3.6 Conclusion

According to the performance data in publication dealing with asynchronous Krylov methods [21], overall speedup did not exceed 2×, confirming the folk theorem for the case that we do no more than overlap communication with computation. However, looking at a large number of repeated runs, we find that operating system noise makes a measurable contribution, and can raise the speedup bound necessitating a new analysis. In this work, introduced a stochastic model showed that uncommon delays, which could quite possibly have been rejected as outliers in other work, are well-modeled by this description. Combined with our new analysis, this demonstrates that speedup greater than 2× is possible for isolated runs, and in a statistical sense. By collecting fine-grained data, we verified our models and studied the noise distribution in different computations.

In future work, we will apply these algorithms to high latency situations where we expect unpredictable delays, such as heavily loaded machines, loosely coupled networks such as those employed for cloud computing or wireless computing clusters, and heterogeneous machines such as those using GPUs. These workloads must be seen in a statistical sense since most users will see computations contaminated by unpredictable, stochastic delays. Our analysis shows the effectiveness of asynchronous methods in these situations, the necessity of characterizing the distribution of noise, and can perhaps guide performance tuning of current algorithms and the development of new projection methods.

BIBLIOGRAPHY

[1] Saurabh Agarwal, Rahul Garg, and Nisheeth K Vishnoi. The impact of noise on the scaling of collectives: A theoretical approach. In *International Conference on High-Performance Computing*, pages 280–289. Springer, 2005.

[2] Douglas N Arnold, Franco Brezzi, and Michel Fortin. A stable finite element for the Stokes equations. *Calcolo*, 21(4):337–344, 1984.

[3] Satish Balay, Shrirang Abhyankar, Mark F Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D Gropp, Dinesh Kaushik, Matthew G Knepley, Dave A May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.8, Argonne National Laboratory, 2017.

[4] Satish Balay, Kris Buschelman, William D Gropp, Dinesh Kaushik, Matthew G Knepley, Lois Curfman McInnes, Barry F Smith, and Hong Zhang. Petsc web page, 2001, 2004.

[5] Satish Balay, William D Gropp, Lois Curfman McInnes, and Barry F Smith. Efficient management of parallelism in object oriented numerical software libraries. In E Arge, A M Bruaset, and H P Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[6] Pete Beckman, Kamil Iskra, Kazutomo Yoshii, and Susan Coghlan. The influence of operating systems on the performance of collective operations at extreme scale. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1–12. IEEE, 2006.

[7] Dimitris Bertsimas, Karthik Natarajan, and Chung-Piaw Teo. Tight bounds on expected order statistics. *Probability in the Engineering and Informational Sciences*, 20(4):667–686, 2006.

[8] Daniele Boffi. Three-dimensional finite element methods for the Stokes problem. *SIAM Journal on Numerical Analysis*, 34(2):664–670, 1997.

[9] S C Brenner and L R Scott. *The Mathematical Theory of Finite Element Methods*. Spinger-Verlag, third edition, 2008.

[10] Franco Brezzi and Michel Fortin. Mixed and hybrid finite element methods, no. 15 in Springer Series in Computational Mathematics, 1991.

[11] A T Chronopoulos and C W Gear. s-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics*, 25:153–168, 1989.

[12] Harald Cramér. *Mathematical methods of statistics (PMS-9)*, volume 9. Princeton University Press, 2016.

[13] Michel Crouzeix and P-A Raviart. Conforming and nonconforming finite element methods for solving the stationary Stokes equations I. *Revue française d'automatique informatique recherche opérationnelle. Mathématique*, 7(R3):33–75, 1973.

[14] CSCS Swiss National Supercomputing Center. The Piz Daint supercomputer, 2015.

[15] E de Sturler and H A van der Vorst. Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. *Applied Numerical Mathematics*, 18(4):441–459, 1995.

[16] Jean Donea and Antonio Huerta. *Finite element methods for flow problems*. John Wiley & Sons, 2003.

[17] Jack Dongarra, Piotr Luszczek, et al. HPC challenge, 2015.

[18] Kurt B Ferreira, Patrick Bridges, and Ron Brightwell. Characterizing application sensitivity to OS interference using kernel-level noise injection. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 19:1–19:12, Piscataway, NJ, 2008. IEEE Press.

[19] Christophe Geuzaine and Jean-François Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 7(11):1309–1331, 2009.

[20] P Ghysels, T J Ashby, K Meerbergen, and W Vanroose. Hiding global communication latency in the GMRES algorithm on massively parallel machines. *SIAM Journal on Scientific Computing*, 35(1):C48–C71, 2013.

[21] P Ghysels and W Vanroose. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm. *Parallel Computing*, 40(7):224–238, 2014. 7th Workshop on Parallel Matrix Algorithms and Applications.

[22] Michael A Heroux, Roscoe A Bartlett, Vicki E Howle, Robert J Hoekstra, Jonathan J Hu, Tamara G Kolda, Richard B Lehoucq, Kevin R Long, Roger P Pawlowski, Eric T Phipps, et al. An overview of the Trilinos project. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):397–423, 2005.

[23] T Hoefler, A Lumsdaine, and W Rehm. Implementation and performance analysis of non-blocking collective operations for MPI. In *Proceedings of the 2007 International Conference on High Performance Computing, Networking, Storage and Analysis, SC07*. IEEE Computer Society/ACM, Nov. 2007.

[24] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Characterizing the influence of system noise on large-scale applications by simulation. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE Computer Society, 2010.

[25] T Jacques, L Nicolas, and C Vollaire. Electromagnetic scattering with the boundary integral method on MIMD systems. In *High-Performance Computing and Networking*, volume 1593 of *Lecture Notes in Computer Science*, pages 1025–1031. Springer, 1999.

[26] V John, A Linke, C Merdon, M Neilan, and L G Rebholz. On the divergence constraint in mixed finite element methods for incompressible flows. *SIAM Review*, to appear, 2017.

[27] Héctor Juárex, Ridgway Scott, Ralph Metcalfe, and Babak Bagheri. Direct simulation of freely rotating cylinders in viscous flows by high-order finite element methods. *Computers & Fluids*, 29:547–582, 2000.

[28] D E Keyes, editor. *A Science-based Case for Large-scale Simulation*. U.S. Department of Energy, 2004.

[29] Alexander Linke. Collision in a cross-shaped domain – A steady 2d Navier—Stokes example demonstrating the importance of mass conservation in CFD. *Computer Methods in Applied Mechanics and Engineering*, 198:3278–3286, 2009.

[30] Alexander Linke, Leo G Rebholz, and Nicholas E Wilson. On the convergence rate of grad-div stabilized Taylor-Hood to Scott-Vogelius solutions for incompressible flow problems. *Journal of Mathematical Analysis and Applications*, 381:612–626, March 2011.

[31] Kent-Antre Mardal and Ragnar Winther. Preconditioning discretizations of systems of partial differential equations. *Numerical Linear Algebra with Applications*, 18(1):1–40, 2011.

[32] Alessandro Morari, Roberto Gioiosa, Robert W Wisniewski, Francisco J Cazorla, and Mateo Valero. A quantitative analysis of OS noise. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 852–863. IEEE, 2011.

[33] Maxim Olshanskii and Arnold Reusken. Grad-div stablilization for Stokes equations. *Mathematics of Computation*, 73(248):1699–1718, 2004.

[34] J Pitman. *Probability*. Springer Texts in Statistics. Springer, New York, Berlin, and Heidelberg, 1993.

[35] Jinshui Qin. *On the convergence of some low order mixed finite elements for incompressible fluids*. PhD thesis, Penn State, 1994.

[36] Yousef Saad. *Iterative Methods for Sparse Linear Systems, 2nd edition*. SIAM, Philadelpha, PA, 2003.

[37] Patrick Sanan. PGMRES and PIPECG test data, 2015.

[38] Francisco-Javier Sayas. A gentle introduction to the finite element method. *Lecture notes, University of Delaware*, 2008.

[39] L R Scott. *Numerical Analysis*. Princeton University Press, 2011.

[40] L R Scott and M Vogelius. Conforming finite element methods for incompressible and nearly incompressible continua. In B. E. Engquist and et al., editors, *Large Scale Computations in Fluid Mechanics*, volume 22 (Part 2), pages 221–244. Providence: AMS, 1985.

[41] L R Scott and M Vogelius. Norm estimates for a maximal right inverse of the divergence operator in spaces of piecewise polynomials. *Mathematical Modelling and Numerical Analysis*, 19(1):111–143, 1985.

[42] Seetharami Seelam, Liana Fong, Asser Tantawi, John Lewars, John Divirgilio, and Kevin Gildea. Extreme scale computing: Modeling the impact of system noise in multicore clustered systems. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.

[43] David Skinner and William Kramer. Understanding the causes of performance variability in HPC workloads. In *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, pages 137–149. IEEE, 2005.

[44] R Strzodka and D Göddeke. Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components. In *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 259–270. IEEE Computer Society, 2006. FCCM '06.

[45] Cedric Taylor and Paul Hood. A numerical solution of the Navier-Stokes equations using the finite element technique. *Computers & Fluids*, 1(1):73–100, 1973.

[46] Andy R Terrel, L Ridgway Scott, Matthew G Knepley, Robert C Kirby, and Garth N Wells. Finite elements for incompressible fluids. In A Logg, K A Mardal, and G Wells, editors, *Automated solutions of differential equations by the finite element method: The FEniCS Book*, pages 381–394. Springer-Verlag New York Inc, 2012.

[47] Shangyou Zhang. Divergence-free finite elements on tetrahedral grids for $k \geq 6$. *Mathematics of Computation*, 80(274):669–695, April 2011.