Dynamic Non-Uniform Randomization in Asynchronous Linear Solvers

Evan Coleman[1] Masha Sosonkina[2]

[1]United States Department of Defense

[2]Old Dominion University

17^{th} Copper Mountain Conference on Iterative Methods
April 4–8, 2022

Outline

Outline

Introduction

- Problem: solve the linear system $Ax = b$
- Some systems don't need to be solved with high accuracy
  - e.g., in AI applications arriving quickly at a sufficiently good answer is preferable to waiting longer for a highly accurate answer
- Asynchronous solvers gain prominence at the exascale and heterogeneous systems
- There are a number of papers that explore the feasibility of using randomized linear solvers to achieve this goal:
  - Leventhal and Lewis;[1] • Griebel and Oswald;[2] • Avron, Druinsky, and Gupta[3]

[1] Dennis Leventhal and Adrian S Lewis. "Randomized methods for linear constraints: convergence rates and conditioning". In: *Mathematics of Operations Research* 35.3 (2010), pp. 641–654.

[2] Michael Griebel and Peter Oswald. "Greedy and randomized versions of the multiplicative Schwarz method". In: *Linear Algebra and its Applications* 437.7 (2012), pp. 1596–1610.

[3] Haim Avron, Alex Druinsky, and Anshul Gupta. "Revisiting asynchronous linear solvers: Provable convergence rate through randomization". In: *Journal of the ACM (JACM)* 62.6 (2015), p. 51.

Motivation

- Wolfson-Pou and Chow[4] investigated a Southwell-like approach for solving linear systems

Question:

Is there a way to combine the natural greediness of the Southwell algorithm with the randomized asynchronous nature of the solvers as proposed in [1–3]?

[4] Jordi Wolfson-Pou and Edmond Chow. "Distributed Southwell: an iterative method with low communication costs". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM. 2017, p. 48.

Outline

Setting the stage

- Everything considered here is some variant of (block) asynchronous Jacobi
- Although these solvers have applications inside other solvers, we focus on their ability to solve systems directly
- First we will describe our approach and motivation then we will go over some results and discuss paths forward

Randomized Gauss Seidel (from Avron et al)

Let $A \in \mathcal{R}^{n\times n}$ be SPD, $b, x_0 \in \mathcal{R}^n$, then perform iterative updates based on:

- $r_0 = b - Ax_j$
- $\gamma_j = d_j^T r_j / d_j^T A d_j$
- $x_{j+1} = x_j + \gamma_j d_j$

for some direction vectors $d_0, d_1, \ldots, d_n$. If the d_j are selected using the distribution,

$$Pr(d_j = e_i) = a_{ii}/Tr(A) \tag{1}$$

then,

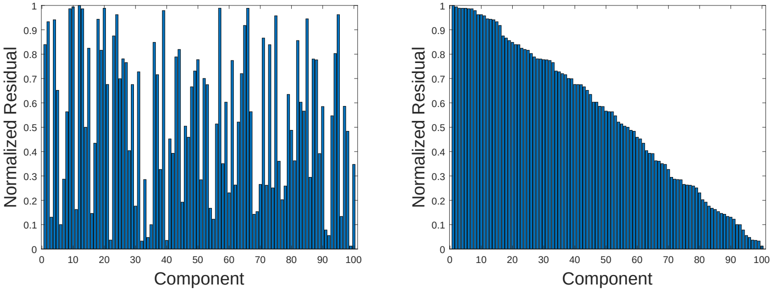
$$\mathbb{E}[\|x_j - x\|_A^2] \leq \left(1 - \frac{\lambda_{min}}{Tr(A)}\right)^m \|x_0 - x^*\|_A^2 \tag{2}$$

Generic algorithm

for *each processing element* P_l **do**
    for $i = 1, 2, \dots$ *until convergence* **do**
        Pick a component $j \in \{1, 2, \dots, m\}$ *somehow*
        Read the corresponding entries of A, x, b
        Perform the relaxation for equation x_j
        Update the data for x_j

- We want to make the component selection random and dynamic

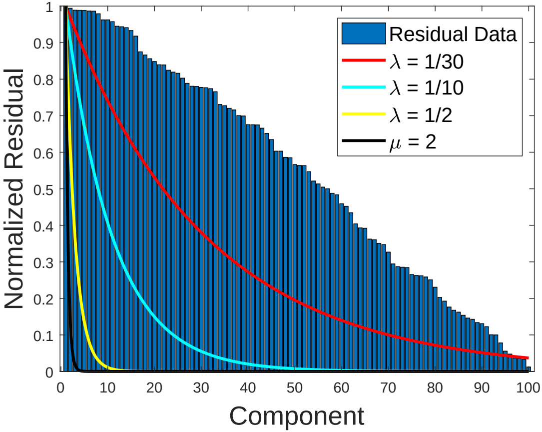
Residual data for finite-difference of 2D Laplacian



(a) Unsorted residuals (b) Sorted residuals

Figure: Initial component residuals $(r_i / \max(r_i))$.

Ranked residual data for finite-difference of 2D Laplacian



(a) Sorted Residuals, exponential distributions

Our approach

- Idea: Make the random selection change dynamically
- Goal: Select the "right" residual components (similar to classical Southwell) without the large computational overhead, incurred in the Southwell by sorting and ranking after each update
- Relies on monitoring which (blocks of) residuals contribute most to the residual: $r = b - Ax$
- Finds and ranks periodically the local/component residuals (for the contribution of block i): $r_i = b_i - Ax_i$
    - Can select a component using a *non-uniform* distribution that favors components with higher local residual

Approaches towards making the component selection

- Uniform distribution
- Discrete (non-updated) distribution defined by the ratio of the diagonal element to the trace

$$\mathbb{P}(i = k) = \frac{a_{kk}}{tr(A)} \tag{3}$$

- "Greedy" selection[5] picks an element within a parameter defined threshold of optimal in the Southwell sense

[5] Griebel and Oswald, "Greedy and randomized versions of the multiplicative Schwarz method".

Approaches towards making the component selection (cont'd)

- Discrete distribution defined by the ratio of the local residual to the sum of all residuals

$$\mathbb{P}(i = k) = \frac{r_k}{\sum_j r_j} \tag{4}$$

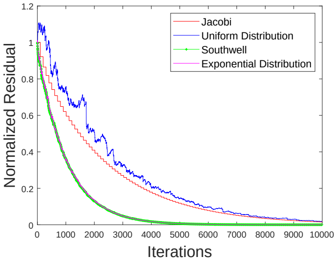
- Periodically fitting a continuous distribution to the (sorted) local residuals and drawing random numbers from this distribution
    - Exponential
    - Triangular

Outline

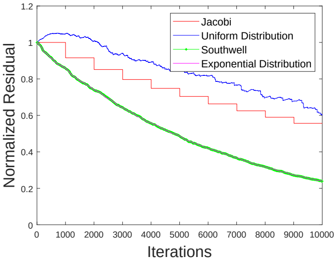
More motivation

- The real question is: how much can this type of component selection improve performance?
- With important subquestion: how much overhead do you introduce to make "better" selections?
- Notes about results:
    - "Uniform" here refers to a true uniform distribution
    - Only a few results are shown in an attempt to just give the flavor of results

Solver data (Laplacian)



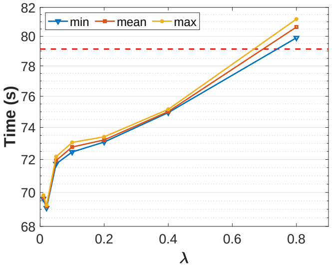
(b) 2D problem (5-pt stencil, 10×10 grid)



(c) 3D problem (27-pt stencil, $10 \times 10 \times 10$ grid)

Figure: Residual (r/r_0) progression for the first 10,000 iterations of four stationary methods solving the 2D (a) and 3D (b) Laplacian.

Solver data (cont'd)

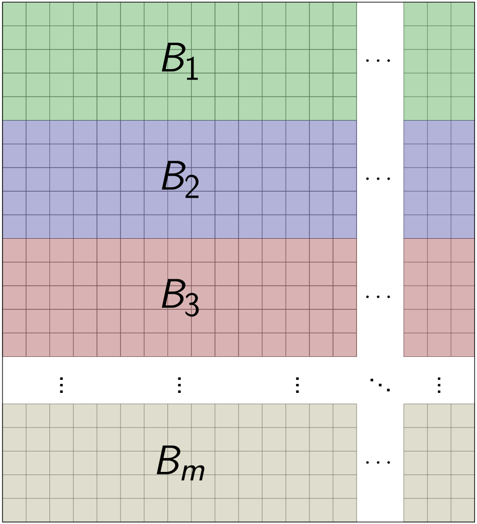


- Shared memory experiments on a node at Old Dominion University
- 64 core Intel Xeon Phi
- 2D discretization of the Laplacian over an 800×800 grid
- The distribution was updated every 5 iterations
- Dashed red line represents the performance of uniform selection

Approach: block methods

- One of the next logical steps would be to look into the extension of these ideas to block methods
- Divide the domain into m subdomains, $\mathbb{R}^n = \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \cdots \mathbb{R}^{n_m}$, where $n = \sum_i n_i$
- Each time a block is selected it performs one or more internal iterations of a stationary solver

Notional block picture



Generic block algorithm

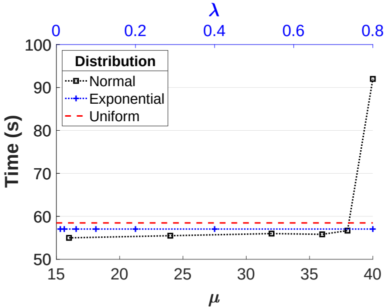
for *each processing element* P_l **do**
  for $i = 1, 2, \ldots$ *until convergence* **do**
    Pick a block $j \in \{1, 2, \ldots, m\}$ *somehow*
    Read the corresponding entries of A, x, b
    Perform Jacobi or Gauss-Seidel relaxations for all equations in block j
    Update the data for block j

- Key: we're performing the dynamic selection on the blocks themselves, and performing traditional iteration inside the blocks

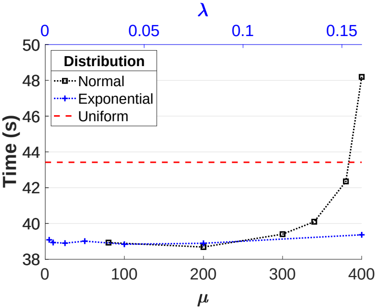
Dynamic block algorithm

- Instead of dividing the domain into blocks as in the previous image, we create blocks dynamically:
    - Each thread selects a single row using our proposed selection methodology to initialize
    - Each update causes each thread to select a new single row using the same methodology
    - These two rows create a block inside which traditional updates (e.g., Jacobi or Gauss-Seidel) are performed on all the components of the two rows
- Need to add locks to avoid multiple threads trying to write to the same component

Solver data from Wahab



(a) Traditional block implementation



(b) Dynamic block implementation

- Still single node, shared memory experiments (different architecture: two Intel Xeon E5-2695 v3 14 core Haswell-EP processors with 32 GB of DRAM)

Outline

Summary

- Dynamic non-uniform randomization provides a potential way to improve the performance of asynchronous linear solvers
- Moving forward, many questions need to be answered to establish that it's an area worth pursuing
- Initial results do suggest that there is potential for the method to provide a modest improvement over existing techniques in certain circumstances

Moving forward

- Questions:
  - How does this extend to a distributed setting?
  - How can we optimize some of these parameters based on intrinsic properties of the matrix?
  - Does the gain in performance overcome the extra computational overhead?
- Further investigation:
  - Try incorporating new solvers into more complex existing routines
  - Keep experimenting with different distributions (still chosen beforehand) and ranking methods and periodicities
  - Try using an evolving probability distribution where the parameters of distribution shift over time

Questions?