Performance Engineering in the Assembly of Sparse Approximate Inverses

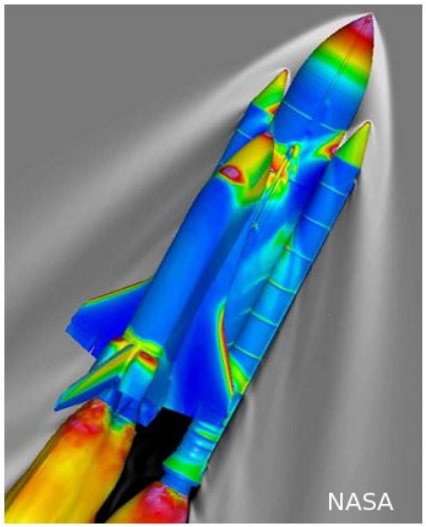
David Schneider

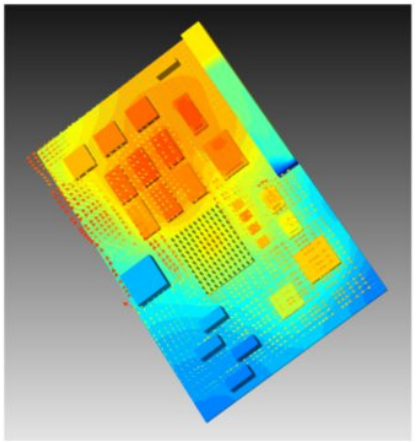
Institute for Applied Mathematics, TU Dortmund

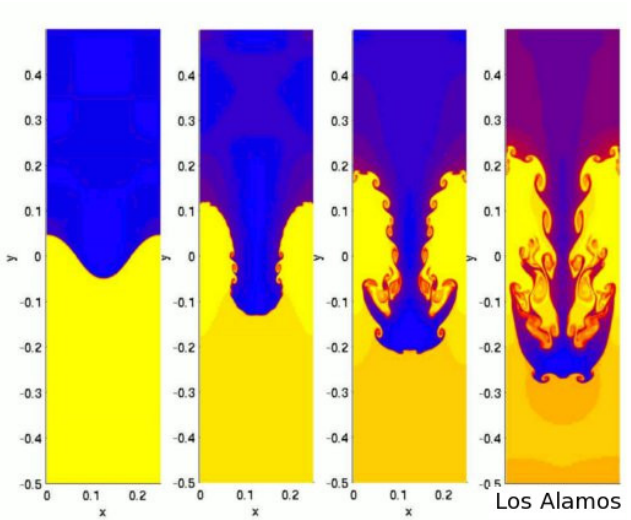
1 June 2016

Overview

Why Preconditioning?







Find $x \in \mathbb{R}^m$ such that $Ax = b$ where $A \in \mathbb{R}^{m \times m}$, $b \in \mathbb{R}^m$ and

- A sparse $\rightarrow$ iterative solvers
- A large $\rightarrow$ parallel, distributed hardware
- $\text{cond}(A) \gg 1$, many iterations required $\rightarrow$ preconditioning

Types of Preconditioners

Let $M \in \mathbb{R}^{m \times m}$ have full rank. Equivalent problems:

- $MAx = Mb$: **Left** Preconditioning
  Objective: $\text{cond}(MA) < \text{cond}(A)$
- $AMy = b$ and $x = My$: **Right** Preconditioning
  Objective: $\text{cond}(AM) < \text{cond}(A)$

AM or MA is never computed, perform $w \leftarrow Mv$ once per iteration

Two classes of preconditioners:

1. construct $M^{-1} \approx A$ and apply via $M^{-1}w = v$
2. construct M and apply via $w = Mv$

Types of Preconditioners

Select suitable preconditioner according to

1. Quality
   - A is very ill-conditioned $\rightarrow$ high quality necessary
   - many systems with similar matrix $A \rightarrow$ high quality pays off
2. Computational Complexity
3. Memory Requirements
4. Parallelizability / Scalability
   - Significance hardware-specific
   - Typical sequential elements:
     Solution of triangular systems, matrix factorizations

As usual, there is a tradeoff between 1 and 2 & 3.

Commonly Used Preconditioners

Class 1: Simplify A

	Quality	Construction	Application	Memory
Jacobi	Low	none	$2n$ FLOPs	none
SSOR	High	none	FWB substitution	none
ILU	High	simple, but sequential	FWB substitution	$\mathcal{O}(\text{mem}(A))$

Class 2: Approximate A^{-1}:

	Quality	Construction	Application	Memory
Jacobi	Low	n divs	n mults	1 vector
Polynomial	Low	none	several MV-mult	none
(M)SPAI	High	involved, but parallel	1 MV-mult	$\mathcal{O}(\text{mem}(A))$

Sparse Approximate Inverses – (M)SPAI

- high-quality, **parallel** preconditioner (Grote, Huckle 1997)
- construct right (or left) preconditioner M by

$$M := \arg\min_{M \in \mathcal{S}} \|AM - I\|_F^2$$

  with a **given sparsity** pattern $\mathcal{S}$
- Static SPAI: fix $\mathcal{S}$, for example to a power of A
- SPAI(ϵ): use heuristic iterations to augment the pattern
- main part: solve m **independent** minimization problems
- efficient implementations require one-sided **communication**, latency hiding and load balancing

Stochastic Matrix Inversion

- Idea: use **Monte Carlo** sampling to obtain a rough sparse approximation of A^{-1} (Dimov et al. 1998)
- **Jacobi** method for the m linear systems $Ax = I$:
  $u_{k+1} = Lu_k + f$ where $L = I - DA$, $f = Db$, and
  $D = \mathrm{diag}(A)^{-1}$
- Solution can be expressed as the Neumann series
  $u = \sum_{q=0}^{\infty} L^q f$
- Construct **Markov chain** for a given $v = (v_1, v_2, \dots)$:
  - state space $\{1, 2, \dots, m\}$
  - transition probabilities $p_{\alpha\beta} = |L_{\alpha\beta}| / \sum_{j=1}^{m} |L_{\alpha j}|$
  - initial probabilities $p_{\alpha} = |v_{\alpha}| / \sum_{j=1}^{m} |v_j|$

Stochastic Matrix Inversion

Stochastic Solution of a SLAE (Spanier, Gelbert 1969)

Let $s_0, s_1, \ldots$ be a Markov chain with the above properties. Then

$$(v, u) = E\left(\frac{v_{s_0}}{p_{s_0}} \sum_{q=0}^{\infty} W_q f_{s_q}\right) \quad \text{with} \quad W_q = \prod_{r=0}^{q-1} \frac{L_{s_r s_{r+1}}}{p_{s_r s_{r+1}}}.$$

As a corollary, a single element of M is given by

$$m_{ij} = \lim_{N\to\infty} \frac{1}{N} \sum_{s=1}^{N} \left(\sum_{q=0}^{\infty} \frac{W_q}{A_{ii}} \delta_{s_q j} \right).$$

Stochastic Matrix Inversion

$$m_{ij} = \lim_{N\to\infty} \frac{1}{N} \sum_{s=1}^{N} \left(\sum_{q=0}^{\infty} \frac{W_q}{A_{ii}} \delta_{s_q j} \right)$$

- One set of Markov chains per row $\rightarrow$ inherent **parallelism**
- **Truncation error** of the chain: Controlled by $|W_q| < \epsilon$
- **Stochastic error**: Controlled by the number of realizations N

- A irreducibly (but not strictly) **diagonally dominant**
  $\rightarrow$ arbitrarily high computational complexity
- A not diagonally dominant $\rightarrow$ divergence

Stochastic Matrix Inversion: General Matrices

Stochastic SPAI algorithm for general matrices:
(Branford, Weihrauch, Alexandrov 2005)

1. Make A diagonally dominant by $A' \leftarrow A + \alpha||A||I$ with $\alpha > 1$
2. Construct M' from A' as before
3. Reconstruct $M = M^{(0)}$ from $M' = M^{(m)}$ by a sequence of **Sherman-Morrison-Updates** of the form

$$M^{(k-1)} \leftarrow M^{(k)} + \frac{\alpha||A||}{1 - \alpha||A||M^{(k)}_{kk}} M^{(k)}_{:k} M^{(k)}_{k:}$$

This algorithm has been demonstrated to **outperform MSPAI** in many cases. (Alexandrov et al. 2014)

Phase 1: Markov Chains

$$m_{ij} = \lim_{N\to\infty} \frac{1}{N} \sum_{s=1}^{N} \left(\sum_{q=0}^{\infty} \frac{W_q}{A_{ii}} \delta_{s_q j} \right)$$

Runtime complexity:

- tuning parameters:
  - linear dimension m
  - truncation tolerance $|W_q| < \epsilon$
  - number of Markov chains N
- overall complexity $\mathcal{O}(mN\lambda)$ with $\lambda \approx \log(\epsilon)/\log(\mathrm{spr}\, L)$

Phase 1: Markov Chains

$$m_{ij} = \lim_{N\to\infty} \frac{1}{N} \sum_{s=1}^{N} \left(\sum_{q=0}^{\infty} \frac{W_q}{A_{ii}} \delta_{s_q j} \right)$$

Memory:

- during their construction, rows are stored in dense format
- afterwards, CSR/ELL compression is possible

Parallelisability / Scalability:

- initial broadcast of A, final broadcast of M
- no further communication required

Phase 1: Markov Chains

Aspects specific to GPUs / parallel hardware:

- map **threads to rows** of M
- Random walks induce scattered **access pattern** to global memory
- for optimal performance / latency hiding, several thousand threads (i.e. rows) need to be constructed in parallel
  $\rightarrow$ high **memory consumption** for large m
- use single (or half) precision during the construction to maximize storage capacity
- first numerical experiments on Tesla K20X for $m \approx 10k$ still show **5-8x speedup** when compared to sequential implementation on Xeon E5-2670

Phase 2: Sherman-Morrison-Updates

Extension to general matrices requires updates:

$$M^{(k-1)} \leftarrow M^{(k)} + \frac{\alpha||A||}{1 - \alpha||A||M^{(k)}_{kk}} M^{(k)}_{:k} M^{(k)}_{k:}$$

Runtime complexity:

- Single update has a general complexity $\mathcal{O}(n^2)$
  $\rightarrow$ original matrix M' needs to be very sparse
- use sparse updates / kernels insteads of BLAS *ger routine

Phase 2: Sherman-Morrison-Updates

Extension to general matrices requires updates:

$$M^{(k-1)} \leftarrow M^{(k)} + \frac{\alpha||A||}{1 - \alpha||A||M^{(k)}_{kk}} M^{(k)}_{:k} M^{(k)}_{k:}$$

Memory:

- Update phase very **inefficient** in a **sparse** matrix storage format
  $\rightarrow$ in principle, (distributed) memory is needed to hold the full dense inverse, scaling as $\mathcal{O}(m^3)$
- Problematic for large, non diagonally dominant FEM matrices

Phase 2: Sherman-Morrison-Updates

$$M^{(k-1)} \leftarrow M^{(k)} + \frac{\alpha||A||}{1 - \alpha||A||M^{(k)}_{kk}} M^{(k)}_{:k} M^{(k)}_{k:}$$

Scalability:

- Update phase does require communication
- Efficient and scalable implementation demonstrated by Alexandrov et al.

Additional aspects relevant to GPUs:

- parallel sparse updates lead to some overhead
- first numerical experiments for $m \approx 10k$ show **similar speedups** as for the Markov chain phase

Summary and Outlook

Stochastic SPAI preconditioning

- is a viable option
  - when a high quality preconditioner is required
  - for (strictly) diagonally dominant or not too sparse matrices
  - on highly distributed architectures
- benefits from the use of parallel hardware such as GPUs
- compares well to existing parallel (M)SPAI preconditioners

Challenges and Future Work:

- use **mixed-precision** approach to access larger m
- combine MPI-based distribution with GPU acceleration
- redesign the update phase to exploit **sparse** matrix formats
- MC-SPAI as multigrid **smoother** instead of preconditioner

Acknowledgements

This work has been supported in part by the German Research Foundation (DFG) through the Priority Program 1648 “Software for Exascale Computing” (grant TU 102/48).