Toward a Memory-Efficient Linear Solver

Allison Baker
Department of Applied Mathematics

E.R. Jessup, J.M. Dennis
Department of Computer Science

University of Colorado at Boulder

Linear systems are ubiquitous

model of physical phenomenon

$\Downarrow$

systems of partial differential equations

$\Downarrow$

(discrete) approximations

$\Downarrow$

large, sparse systems of linear equations $*$

$*$ often time-consuming to solve!

Can we solve large, sparse systems of linear equations more efficiently?

- Linear system: $\mathbf{Ax} = \mathbf{b}$

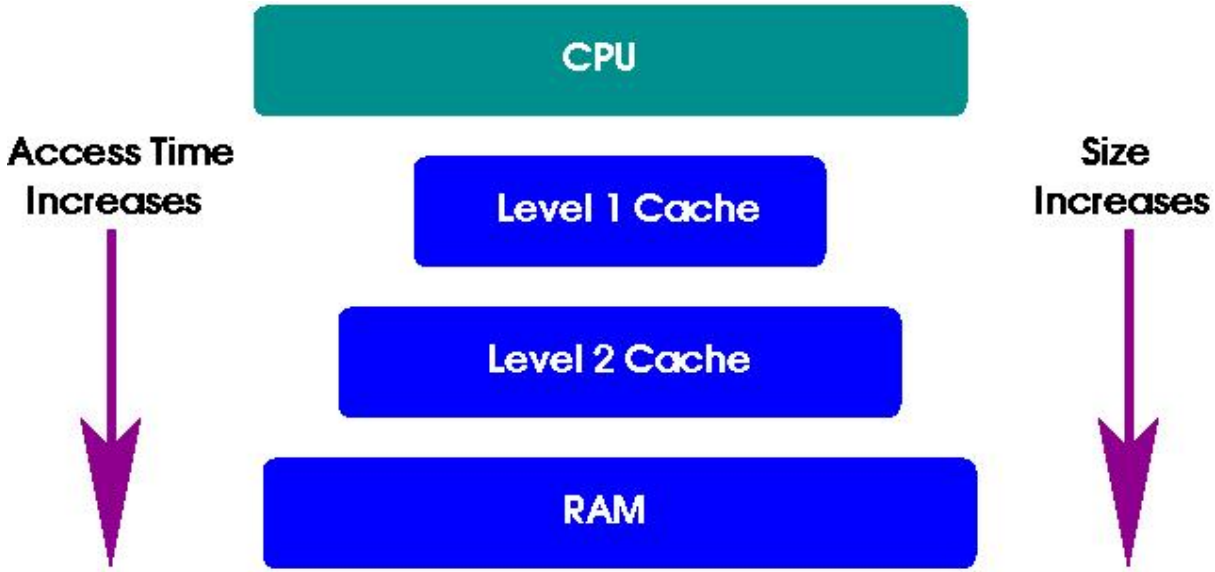
  ($\mathbf{A}$ is large, sparse, and nonsymmetric)

- Costs: (1) floating-point operations

  (2) movement of data through memory

- Our goal: reduce memory costs (2)

Memory Hierarchy



component	Typical CPU latency value
L1 cache	1 - 3 cycles
L2 cache	6 - 20 cycles
RAM (main memory)	60 - 140 cycles
disk	10,000 cycles !!!

Memory costs impact performance

- moving data is expensive

- number of floating-point operations required by algorithms has decreased

- getting worse:

  microprocessor performance $\sim$ 60% per year vs. memory performance: $<$ 10% per year

$\Rightarrow$ time the CPU spends waiting for data is increasing!

Iterative linear solvers

- repeatedly revise estimate for solution $\mathbf{x}$
- based on a sparse matrix-vector multiply
- access $\mathbf{A}$ once per iteration loop for $\mathbf{A} * \mathbf{x}$

$\Rightarrow$ number of floating point operations is not an accurate predictor of algorithm performance

$\Rightarrow$ need to minimize data movement

Approach to improving an iterative linear solver

Algorithmic developments:

- Solve $\mathbf{Ax} = \mathbf{b}$ via $\mathbf{AX} = \mathbf{B}$

$$\begin{bmatrix} \\ \mathbf{A} \\ \\ \end{bmatrix} \begin{bmatrix} \\ \mathbf{x} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \mathbf{b} \\ \\ \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} \\ \mathbf{A} \\ \\ \end{bmatrix} \begin{bmatrix} \\ \mathbf{X} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \mathbf{B} \\ \\ \end{bmatrix}$$

- choose $\mathbf{X}$ and $\mathbf{B}$ to promote convergence

Performance programming techniques:

- recover costs of $\mathbf{A} * \mathbf{X}$ vs. $\mathbf{A} * \mathbf{x}$

A common linear solver: GMRES(m)

Why?: popular for nonsymmetric problems!

Solve: $\mathbf{Ax = b}$

initial guess $\rightarrow \mathbf{x_0}$

residual $\rightarrow \mathbf{r_0 = b - Ax_0}$

Iterate:

- $\mathbf{x_{i+1} \in x_i + \mathcal{K}_m(A, r_i)}$

  $\mathbf{\mathcal{K}_m(A, r_i) \equiv span\{r_i, Ar_i, \ldots, A^{m-1} r_i\}}$

- minimize the 2-norm of the residual: $\|\mathbf{b - Ax_{i+1}}\|_2$

- discard approximation space and restart ($i = i + 1$)

What should we choose for B for GMRES(m)? ($\mathbf{Ax = b \Rightarrow AX = B}$)

- goal:
  balance memory-usage and favorable numerical properties

- observation:
  restarted GMRES(m) throws away useful information with every restart, which slows convergence

- our approach:
  compensate for this loss $\rightarrow$ put the lost information in B!

$\mathbf{Ax = b \Rightarrow AX = B}$

GMRES(m):

- restart cycle $\mathbf{i}$: $\mathbf{x_i = x_{i-1} + c}, \quad \mathbf{c \in \mathcal{K}_m(A, r_{i-1})}$
- ideally: $\mathbf{c = x_{exact} - x_{i-1} \longrightarrow x_i = x_{exact}}$
- error approximation: $\mathbf{z_i \equiv x_i - x_{i-1}}$

idea:

- put the error approximation vector in $\mathbf{B}$: $\mathbf{B = [\, b\,,\, z_i\,]}$

A new algorithm: B-LGMRES

B-LGMRES(m, 1):

- restart cycle $\mathbf{i+1}$: $\mathbf{Ax = b \Rightarrow AX = B}$

$$\mathbf{X = [\, x_i \,,\ 0\,]}$$

$$\mathbf{B = [\, b \,,\ z_i\,]}$$

  i.e., augment $\mathbf{b}$ with an error approximation

- $\underbrace{\mathbf{x_{i+1} \in x_i + \mathcal{K}_m(A, r_i)}}_{\|\ \text{GMRES(m)}} + \mathcal{K}_\mathbf{m}(\mathbf{A}, \mathbf{z_i})$

(resembles a full conjugate gradient method with polynomial preconditioning)

B-LGMRES: an efficient implementation

Matrix-multivector multiply (Kaushik '99):

multiply $\mathbf{A}$*4 vectors in $\approx$ 1.5 the time for $\mathbf{A}$*1 vector

- interleave the vector components in memory:
  $\mathbf{v_1(j)}$, $\mathbf{v_2(j)}$, $\mathbf{v_3(j)}$, $\mathbf{v_4(j)}$ are adjacent

- unroll the inner matrix multiplication loop
  (across multivector elements)

Other misc. multivector routines:

multivector dot, saxpy, maxpy, and forward and back solves

Evaluating the new method:

B-LGMRES(15, 1) vs. GMRES(30):

- equal approximation space size (30)
- variety of problems from several test collections
- problem size: $n = 7,740$ to $1,709,982$
  ($nnz = 79,566$ to $1,717,792$)
- either ILU preconditioner or no preconditioner
- Argonne's PETSc with modifications

Some results...

$\Rightarrow$ the new method is generally faster

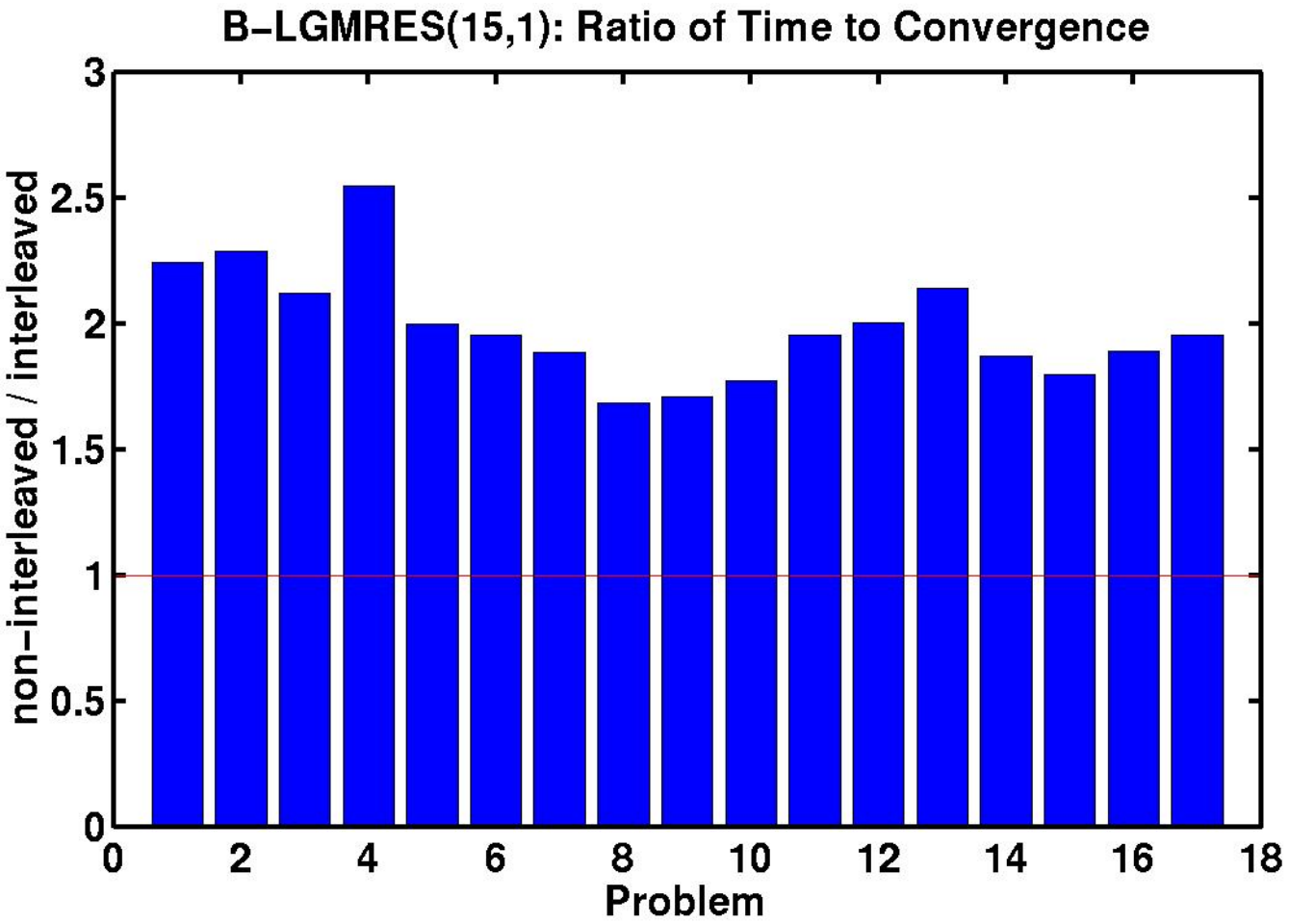
Problem 12: 2-D fluid flow

size: n=13,535, nnz=390,607

method	execution time	accesses of $\mathbf{A}$	matrix-vector multiplies
GMRES(30)	$165.9 \pm .6$ s	1960	1960
B-LGMRES(15,1)	$78.5 \pm .2$ s	1004	2008

$\Rightarrow$ execution time correlates to number of accesses of $\mathbf{A}$
(related to memory usage, not floating-point operations)

Importance of an efficient implementation



interleaved: $\mathbf{v_1(j)}$, $\mathbf{v_2(j)}$ are adjacent

non-interleaved: $\mathbf{v_1(j)}$, $\mathbf{v_1(j+1)}$ are adjacent

In summary...

Simple changes have large impacts on performance

- algorithmic changes: changing b to B
- implementation: basic programming tricks

$\Rightarrow$ modifying a linear solver to reduce data movement is a viable approach to gaining efficiency!

For more information...

- **A.H. Baker**, J.M. Dennis, and E.R. Jessup. An efficient block variant of GMRES. Technical Report #CU-CS-957-03, University of Colorado, Department of Computer Science, July, 2003. *Submitted for publication.*

- **A.H. Baker**, E.R. Jessup, and T. Manteuffel. A technique for accelerating the convergence of restarted GMRES. Technical Report #CU-CS-945-03, University of Colorado, Department of Computer Science, January, 2003. *Submitted for publication.*

Download from:
http://amath.colorado.edu/student/allisonb/research.html