Communication Avoiding and Hiding in preconditioned Krylov solvers

[1]Applied Mathematics, University of Antwerp, Belgium
[2] Future Technology Group, Berkeley Lab, USA
[3]Intel ExaScience Lab, Belgium
[4]USI, Lugano, CH
[5]KU Leuven, Belgium

B. Reps[1], P. Ghysels[2], O. Schenk[4], K. Meerbergen[3,5], W. Vanroose[1,3]

IHP 2015, Paris FRx

Overview

- Arithmetic Intensity in Krylov
- Arithmetic Intensity in Multigrid
- Pipelining communication and computation in Krylov

GMRES, classical Gram-Schmidt

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

Sparse Matrix-Vector product

- Only communication with neighbors
- Good scaling

Dot-product

- Global communication
- Scales as $\log(P)$

Scalar vector multiplication, vector-vector addition

- No communication

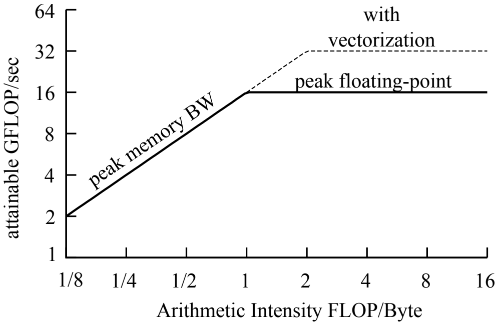
Part I

Arithmetic Intensity and Sparse Matrix vector product

Performance of Kernel of dependent SpMV

- SIMD: SSE → AVX → _mm512 on Xeon Phi.
- Similar with NEON on ARM.

Bandwidth is the bottleneck and will remain even with 3D stacked memory.



Arithmetic intensity (Flop/byte)

S. Williams, A. Waterman and D. Patterson (2008)

Arithmetic intensity of k dependent SpMVs

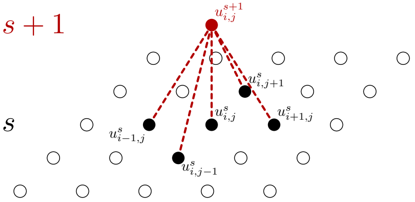
	1 SpMV	k SpMVs	k SpMVs in place
flops	$2n_{nz}$	$2k \cdot n_{nz}$	$2k \cdot n_{nz}$
words moved	$n_{nz} + 2n$	$kn_{nz} + 2kn$	$n_{nz} + 2n$
q	2	2	**2k**

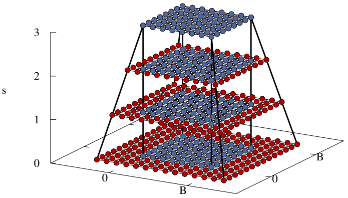
J. Demmel, CS 294-76 on Communication-Avoiding algorithms

M. Hoemmen, Communication-avoiding Krylov subspace methods. (2010).

Increasing the arithmetic intensity

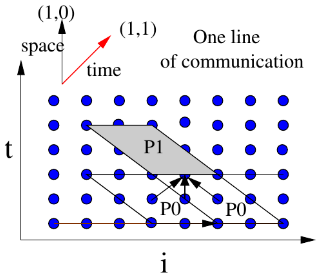
- Divide the domain in tiles which fit in the cache
- Ground surface is loaded in cache and reused k times
- Redundant work at the tile edges





P. Ghysels, P. Klosiewicz, and W. Vanroose. "Improving the arithmetic intensity of multigrid with the help of polynomial smoothers." Num. Lin. Alg. Appl. **19** (2012): 253-267.

Stencil Compilers. (polytope model)

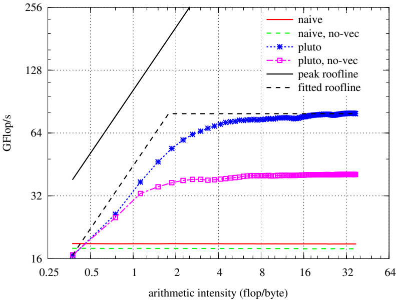


Pluto

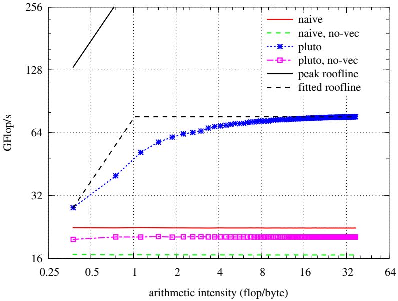
- Automatic loop parallelization,
- Locality optimizations based on the polyhedral model,
- add OpenMP pragma's around the outer loop,
- `ivdep` and `vector always`. compilers (guided) auto-vectorization.

U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, *Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model*, Int. Conf. Compiler Construction (ETAPS CC), Apr 2008, Budapest, Hungary.

Increasing the arithmetic intensity with a stencil compiler (Pluto)



Dual socket Intel Sandy Bridge, 16×2 threads



Intel Xeon Phi, 61×4 threads

Krylov methods

- Classical Krylov

$$K_k(A, v) = \text{span}\{v, Av, A^2v, \dots, A^{k-1}v\}$$

the residual and error can then be written as

$$r^{(k)} = P_k(A)r^{(0)} \tag{1}$$

Krylov methods

- Classical Krylov

$$K_k(A, v) = \mathrm{span}\{v, Av, A^2v, \dots, A^{k-1}v\}$$

the residual and error can then be written as

$$r^{(k)} = P_k(A)r^{(0)} \tag{1}$$

- Polynomial Preconditioning

$$P_m(A)x = q(A)Ax = q(A)b$$

$$K_k(P_m(A), v) = \mathrm{span}\{v, P_m(A)v, P_m(A)^2v, \dots, P_m(A)^{k-1}v\}$$

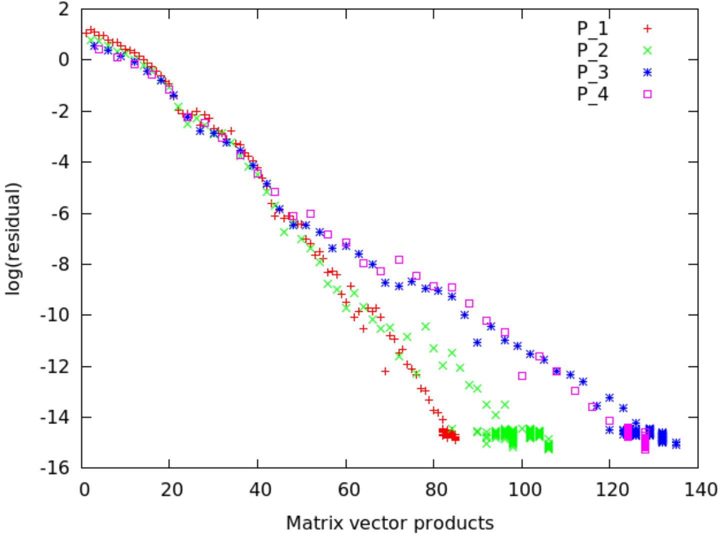
where $P_m(A) = (A - \sigma_1)(A - \sigma_2) \cdot \ldots \cdot (A - \sigma_m)$ is a fixed low-order polynomial

$$r^{(k)} = Q_k(P_m(A))r^{(0)} \tag{2}$$

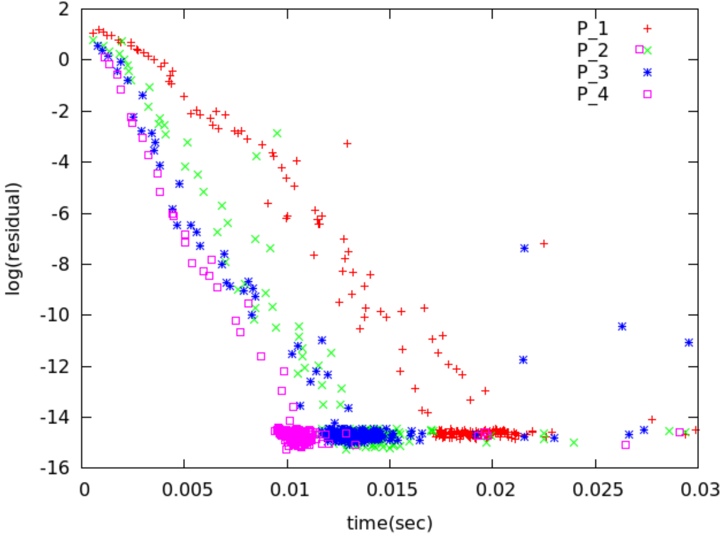
Incomplete list of the literature on polynomial preconditioning

- Y. Saad, *Iterative methods for sparse linear systems* Chapter 12, SIAM (2003).
- D. O'Leary, *Yet another polynomials preconditioner for the Conjugate Gradient Algorithm*, Lin. Alg. Appl. (1991) p377
- A. van der Ploeg, *Preconditioning for sparse matrices with applications*, (1994)
- M. Van Gijzen, *A polynomial preconditioner for the GMRES algorithm* J. Comp. Appl. Math **59** (1995): 91-107.
- A. Basermann, B. Reichel, C. Schelthoff, *Preconditioned CG methods for sparse matrices on massively parallel machines*, **23** (1997), 381398
- ...

Convergence of CG with different short Polynomials



Time to solution is reduced

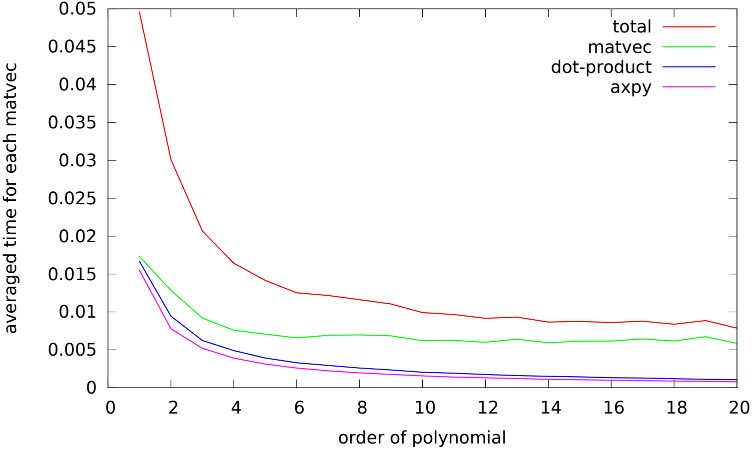


Recursive calculation of $w_{k+1} = p_{k+1}(A)v$

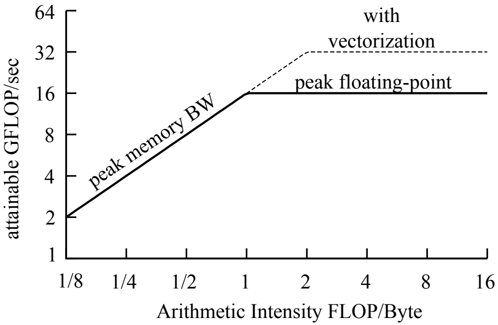
$\sigma_1 = \theta/\delta$
$\rho_0 = 1/\sigma_1$
$w_0 = 0,\ w_1 = \frac{1}{\theta}Av$
$\Delta w_1 = w_1 - w_0$
for k=1,... **do**
    $\rho_k = 1/(2\sigma_1 - \rho_{k-1})$
    $\Delta w_{k+1} = \rho_k \left[\frac{2}{\delta}A(v - w_k) + \rho_{k-1}\Delta w_k\right]$
    $w_{k+1} = w_k + \Delta w_{k+1}$
end for

Mangled with Pluto to raise arithmetic intensity.

Average cost for each matvec



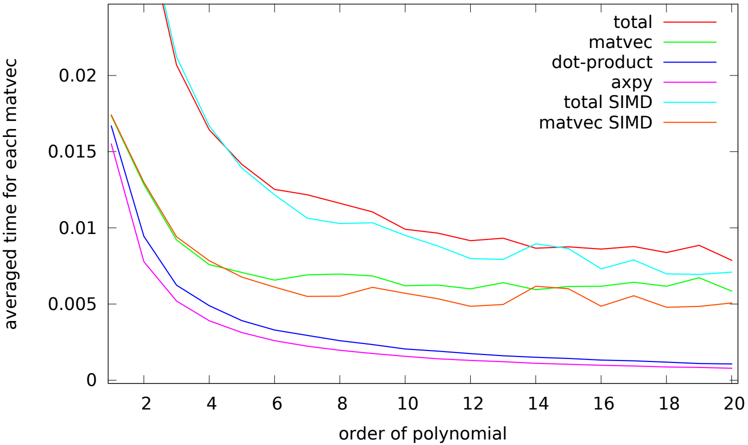
2D 2024×2048 finite difference poisson problem on i7-2860QM



Arithmetic intensity (Flop/byte)

S. Williams, A. Waterman and D. Patterson (2008)

Relying on compiler autovectorization



Intel i7-2860QM

Part II

Arithmetic Intensity in Multigrid

Arithmetic Intensity Multigrid

$$MG = (I - I_{2h}^{h}(\ldots)I_{h}^{2h}A)S^{\nu_1}, \tag{3}$$

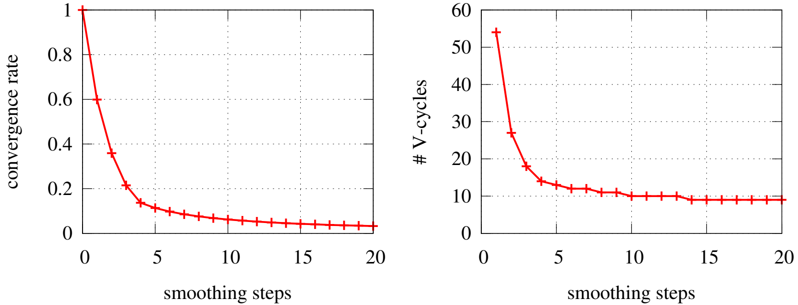
where

- S is the smoother, for example ω-Jacobi where $S = I - \omega D^{-1}A$,
- The interpolation I_{2h}^{h} and restriction I_{h}^{2h}.

These are all low arithmetic intensity.

Multigrid Cost Model

Increasing the number of smoothing steps per level



$$\text{MG iterations} = \log(10^{-8}) / \log(\rho(\text{V-cycle}))$$

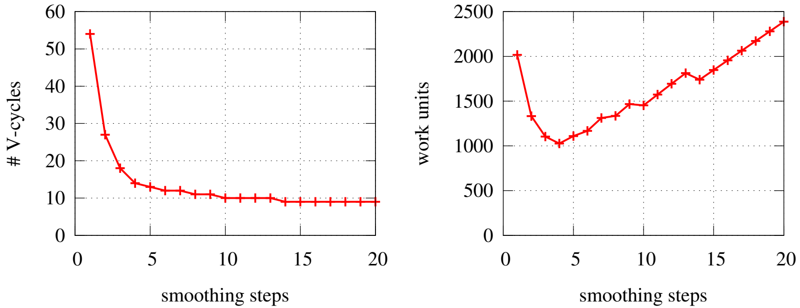
where ρ(V-cycle) is the spectral radius of the V-cycle, can be rounded to higher integer

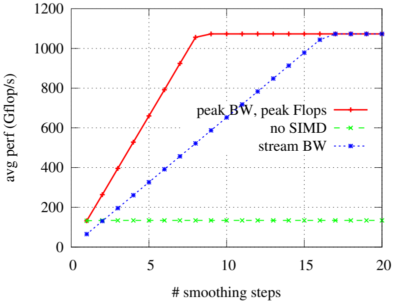
Work Unit Cost model

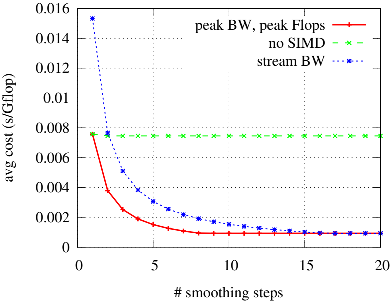
- Work Unit cost model ignores memory bandwidth

$$1\,\mathsf{WU} = \text{smoother cost} = \mathcal{O}(n)$$

- $$(9\nu+19)(1+\frac{1}{4}+\dots)\,\frac{\log(\texttt{tol})}{\log(\rho)}\mathsf{WU} \le (9\nu+19)\,\frac{4}{3}\,\frac{\log(\texttt{tol})}{\log(\rho)}\mathsf{WU}$$





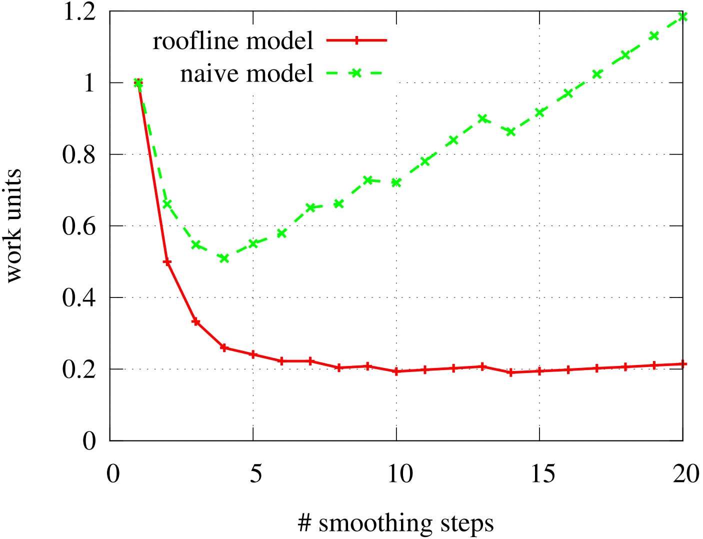


Attainable GFlop/sec

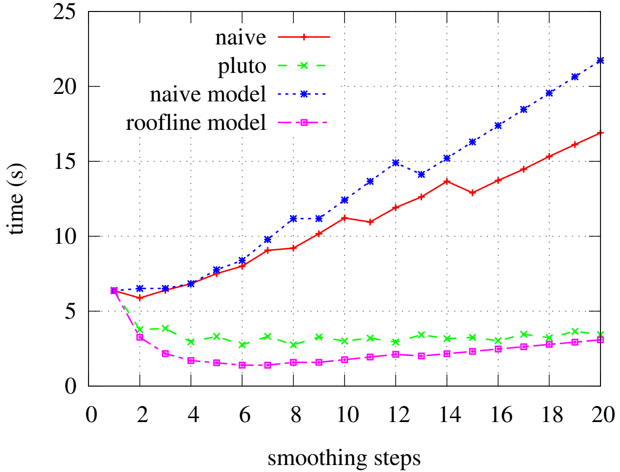
Average cost in sec/GFlop

$$\langle \nu_1 \times \omega\text{-Jac} \rangle = \frac{\text{flops}\,(\nu_1 \times \omega\text{-Jac})}{\text{roof}\,(q(\nu_1 \times \omega\text{-Jac}))} \approx \frac{\nu_1 \text{flops}\,(\omega\text{-Jac})}{\text{roof}\,(\nu_1 q_1(\omega\text{-Jac}))} \tag{4}$$

Roofline-based vs naive cost model

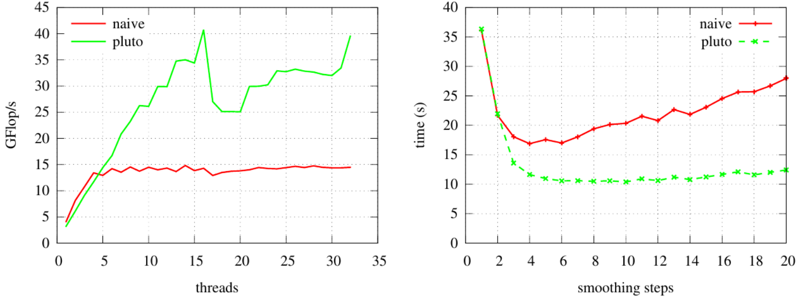


Timings for 2D 8190^2 (Sandy Bridge)



P. Ghysels and W. Vanroose, *Modelling the performance of geometric multigrid on many-core computer architectures*, Exascience.com techreport 09.2013.1 Sept, 2013.

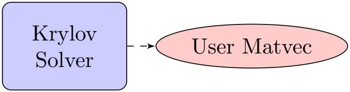
3d results 511^3 (Sandy Bridge)



(a) smoother scaling

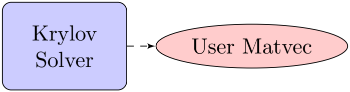
(b) V-cycle timing

Current Krylov Solvers



- User provides a matvec routine and selects Krylov method at command line.
- No opportunities to increase arithmetic intensity.

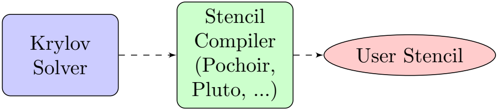
Current Krylov Solvers



- User provides a matvec routine and selects Krylov method at command line.
- No opportunities to increase arithmetic intensity.

Future Krylov Solvers

- User provides a stencil routine.
- Stencil compiler increases arithmetic intensity.



W .Vanroose, P. Ghysels, D. Roose and K. Meerbergen, Position paper at DOE Exascale Mathematics workshop 2013.

Part III

Pipelining communication and computations

GMRES, classical Gram-Schmidt

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

Sparse Matrix-Vector product

- Only communication with neighbors
- Good scaling but BW limited.

Dot-product

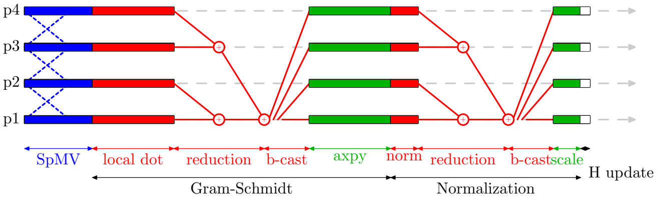
- Global communication
- Scales as $\log(P)$

Scalar vector multiplication, vector-vector addition

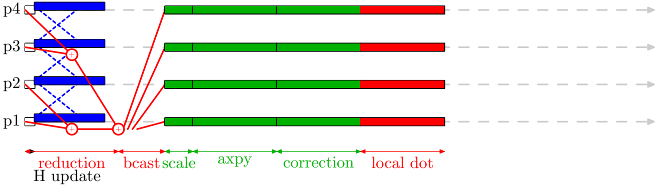
- No communication

GMRES vs Pipelined GMRES iteration on 4 nodes

Classical GMRES

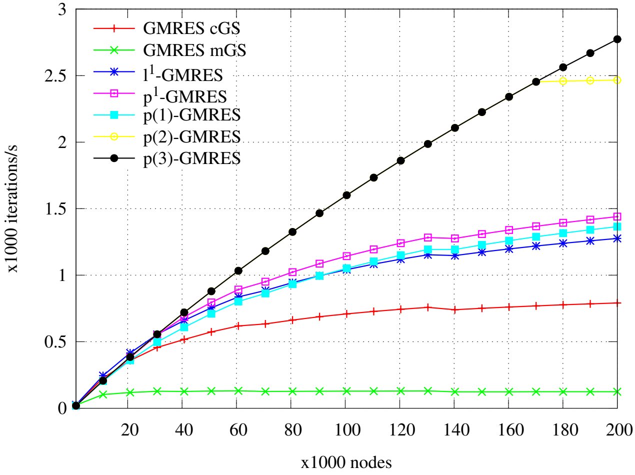


Pipelined GMRES



P. Ghysels, T. Ashby, K. Meerbergen and W. Vanroose *Hiding global communication latency in the GMRES algorithm on massively parallel machines.* SIAM J. Scientific Computing, 35(1):C48C71, (2013).

Better Scaling



Prediction of a strong scaling experiment for GMRES on XT4 part of Cray Jaguar.

Are there similar opportunities in preconditioned Conjugate Gradients?

Preconditioned Conjugate Gradient

$r_0 := b - Ax_0;\ u_0 := M^{-1} r_0;\ p_0 := u_0$
for $i = 0, \ldots, m-1$ **do**
  $s := Ap_i$
  $\alpha := \langle r_i, u_i \rangle \,/\, \langle s, p_i \rangle$
  $x_{i+1} := x_i + \alpha p_i$
  $r_{i+1} := r_i - \alpha s$
  $u_{i+1} := M^{-1} r_{i+1}$
  $\beta := \langle r_{i+1}, u_{i+1} \rangle \,/\, \langle r_i, u_i \rangle$
  $p_{i+1} := u_{i+1} + \beta p_i$
end for

Chronopoulos/Gear CG

Only one global reduction each iteration.

$r_0 := b - Ax_0;\ u_0 := M^{-1}r_0;\ w_0 := Au_0$
$\alpha_0 := \langle r_0, u_0 \rangle \,/\, \langle w_0, u_0 \rangle;\ \beta := 0;\ \gamma_0 := \langle r_0, u_0 \rangle$
for $i = 0, \ldots, m-1$ **do**
    $p_i := u_i + \beta_i p_{i-1}$
    $s_i := w_i + \beta_i s_{i-1}$
    $x_{i+1} := x_i + \alpha p_i$
    $r_{i+1} := r_i - \alpha s_i$
    $u_{i+1} := M^{-1} r_{i+1}$
    $w_{i+1} := Au_{i+1}$
    $\gamma_{i+1} := \langle r_{i+1}, u_{i+1} \rangle$
    $\delta := \langle w_{i+1}, u_{i+1} \rangle$
    $\beta_{i+1} := \gamma_{i+1}/\gamma_i$
    $\alpha_{i+1} := \gamma_{i+1}/(\delta - \beta_{i+1}\gamma_{i+1}/\alpha_i)$
end for

pipelined Chronopoulos/Gear CG

Global reduction overlaps with matrix vector product.

$r_0 := b - Ax_0$; $w_0 := Au_0$
for $i = 0, \ldots, m-1$ **do**
  $\gamma_i := \langle r_i, r_i \rangle$
  $\delta := \langle w_i, r_i \rangle$
  $q_i := Aw_i$
  if $i > 0$ **then**
    $\beta_i := \gamma_i / \gamma_{i-1}$; $\alpha_i := \gamma_i / (\delta - \beta_i \gamma_i / \alpha_{i-1})$
  else
    $\beta_i := 0$; $\alpha_i := \gamma_i / \delta$
  end if
  $z_i := q_i + \beta_i z_{i-1}$
  $s_i := w_i + \beta_i s_{i-1}$
  $p_i := r_i + \beta_i p_{i-1}$
  $x_{i+1} := x_i + \alpha_i p_i$
  $r_{i+1} := r_i - \alpha_i s_i$
  $w_{i+1} := w_i - \alpha_i z_i$
end for

Preconditioned pipelined CG

$r_0 := b - Ax_0;\ u_0 := M^{-1} r_0;\ w_0 := Au_0$
for $i = 0, \ldots$ **do**
  $\gamma_i := \langle r_i, u_i \rangle$
  $\delta := \langle w_i, u_i \rangle$
  $m_i := M^{-1} w_i$
  $n_i := Am_i$
  if $i > 0$ **then**
    $\beta_i := \gamma_i / \gamma_{i-1};\ \alpha_i := \gamma_i / \left(\delta - \beta_i \gamma_i / \alpha_{i-1}\right)$
  else
    $\beta_i := 0;\ \alpha_i := \gamma_i / \delta$
  end if
  $z_i := n_i + \beta_i z_{i-1}$
  $q_i := m_i + \beta_i q_{i-1}$
  $s_i := w_i + \beta_i s_{i-1}$
  $p_i := u_i + \beta_i p_{i-1}$
  $x_{i+1} := x_i + \alpha_i p_i$
  $r_{i+1} := r_i - \alpha_i s_i$
  $u_{i+1} := u_i - \alpha_i q_i$
  $w_{i+1} := w_i - \alpha_i z_i$
end for

Preconditioned pipelined CR

$r_0 := b - Ax_0$; $u_0 := M^{-1}r_0$; $w_0 := Au_0$
for $i = 0, \ldots$ **do**
  $m_i := M^{-1}w_i$
  $\gamma_i := \langle w_i, u_i \rangle$
  $\delta := \langle m_i, w_i \rangle$
  $n_i := Am_i$
  if $i > 0$ **then**
    $\beta_i := \gamma_i/\gamma_{i-1}$; $\alpha_i := \gamma_i/\left(\delta - \beta_i\gamma_i/\alpha_{i-1}\right)$
  else
    $\beta_i := 0$; $\alpha_i := \gamma_i/\delta$
  end if
  $z_i := n_i + \beta_i z_{i-1}$
  $q_i := m_i + \beta_i q_{i-1}$
  $p_i := u_i + \beta_i p_{i-1}$
  $x_{i+1} := x_i + \alpha_i p_i$
  $u_{i+1} := u_i - \alpha_i q_i$
  $w_{i+1} := w_i - \alpha_i z_i$
end for

Cost Model

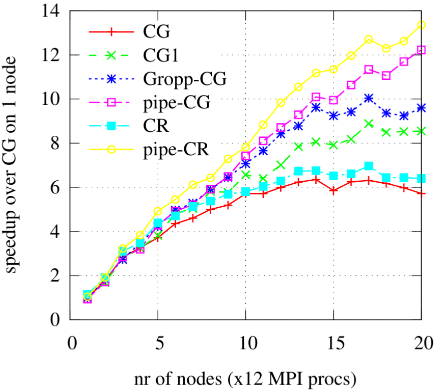
- G := time for a global reduction
- SpMV := time for a sparse-matrix vector product
- PC := time for preconditioner application
- local work such as AXPY is neglected

	flops	time (excl, AXPYs, DOTs)	#glob syncs	memory
CG	10	$2G + SpMV + PC$	2	4
Chro/Gea	12	$G + SpMV + PC$	1	5
CR	12	$2G + SpMV + PC$	2	5
pipe-CG	20	$\max(G, SpMV + PC)$	1	9
pipe-CR	16	$\max(G, SpMV) + PC$	1	7
Gropp-CG	14	$\max(G, SpMV) + \max(G, PC)$	2	6

P. Ghysels and W. Vanroose, *Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm*, Parallel Computing, **40**, (2014), Pages 224238

Better Scaling

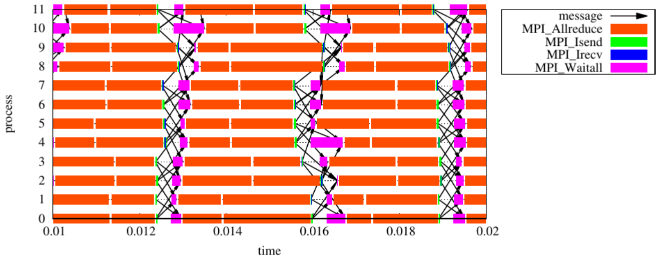
- Hydrostatic ice sheet flow, $100 \times 100 \times 50$ $Q1$ finite elements
- line search Newton method (rtol $= 10^{-8}$, atol $= 10^{-15}$)
- CG with block Jacobi ICC(0) precond (rtol $= 10^{-5}$, atol $= 10^{-50}$)



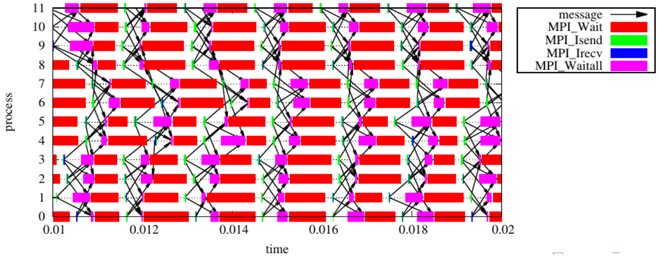
Measured speedup over standard CG for different variations of pipelined CG.

MPI trace

Conjugate Gradients for 2D 5-point stencil



Pipelined Conjugate Gradients



Pipelined methods in PETSc (from 3.4.2)

- Krylov methods: KSPPIPECR, KSPPIPECG, KSPGROPPCG, KSPPGMRES
- Uses MPI-3 non-blocking collectives
- export MPICH_ASYNC_PROGRES=1

```
...
KSP_PCApply(ksp,W,M); /* m ← Bw */
if (i > 0 && ksp→normtype == KSP_NORM_PRECONDITIONED)
      VecNormBegin(U,NORM_2,&dp);
VecDotBegin(W,U,&gamma);
VecDotBegin(M,W,&delta);
PetscCommSplitReductionBegin(PetscObjectComm((PetscObject)U));
KSP_MatMult(ksp,Amat,M,N); /* n ← Am */
if (i > 0 && ksp→normtype == KSP_NORM_PRECONDITIONED)
      VecNormEnd(U,NORM_2,&dp);
VecDotEnd(W,U,&gamma);
VecDotEnd(M,W,&delta);
...
```

Krylov subspace methods with additional global reductions

- Deflation: Remove a few known *annoying* eigenvectors.
  - Helmholtz.
  - FETI methods.
  - ...
- Augmenting: adds a subspace to the Krylov subspace, e.g. recycling.
  - Newton-Krylov methods.
  - Numerical Continuation.
  - Coarse Solver in multigrid.
  - ...

$$\mathcal{S}_n := \mathcal{K}_n(A, v) + \mathcal{U} \tag{5}$$

$$x_n = x_0 + V_n y_n + U u_n$$

where U forms a basis for $\mathcal{U}$.

Deflation with eigenvectors with smallest eigenvalues

Smallest eigenvalues and vectors:

$$A\,[w_1, w_2, \ldots, w_m] = [\lambda_1 w_1, \lambda_2 w_2, \ldots, \lambda_m w_m] + \epsilon[\Theta]$$

where $\lambda_1 < \lambda_2 < \lambda_3 < \ldots < \lambda_m < \ldots$ and $\|\Theta\| \approx 1$.

$$W := [w_1, w_2, \ldots, w_m]$$

Correction step

$$e = W(W^T A W)^{-1} W^T r$$

Deflated CG

$r_{-1} := b - Ax_{-1}$
$x_0 := x_{-1} + W(W^TAW)^{-1}W^Tr_{-1}$
$r_0 := b - Ax_0$
$p_0 := r_0 - W(W^TAW)^{-1}W^TAr_0$
for $i = 0, \dots$ **do**
  $s := Ap_i$
  $\alpha_i := \langle r_i, r_i \rangle \,/\, \langle s, p_i \rangle$
  $x_{i+1} := x_i + \alpha_i p_i$
  $r_{i+1} := r_i - \alpha_i s$
  $\beta_i := \langle r_{i+1}, r_{i+1} \rangle \,/\, \langle r_i, r_i \rangle$
  $w := Ar_{i+1}$
  $\sigma := \langle W, w \rangle$
  $p_{i+1} := r_{i+1} + \beta_i p_i - W(W^TAW)^{-1}\sigma$
end for

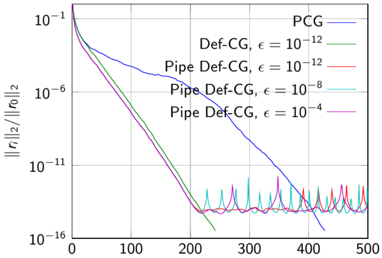
- x_0 such that $W^Tr_0 = 0$ with $r_0 = b - Ax_0$ (cfr init-CG)
- $\sigma = \langle W, w \rangle = \langle W, Ar_{i+1} \rangle = \langle AW, r_{i+1} \rangle$: store AW?
- CG on $H^TAH\tilde{x} = H^Tb$ with $H = I - W(W^TAW)^{-1}(AW)^T$

Pipelined Deflation CG

Pipe-Def-CG(A, M^{-1}, b, x_{-1}, W)

$r_0 = b - Ax_0$
$x_0 = x_0 + W(W^TAW)^{-1}W^Tr_0$
$r_0 = b - Ax_0$
$p_0 = r_0 - W(W^TAW)^{-1}W^TAr_0$
$w_0 = Ar_0$
for $i = 0, \ldots$ **do**
  $\gamma_i = (r_i, r_i), \quad \delta = (w_i, r_i)$
  $\sigma = (W, w_i)$
  $q_i = Aw_i$
  if $i > 0$ **then**
    $\beta_i = \gamma_i/\gamma_{i-1}, \quad \alpha_i = \gamma_i/\left(\delta - \beta_i\gamma_i/\alpha_{i-1}\right)$
  else
    $\beta_i = 0, \quad \alpha_i = \gamma_i/\delta$
  end if
  $z_i = q_i + \beta_i z_{i-1} - A^2W(W^TAW)^{-1}\sigma$
  $s_i = w_i + \beta_i s_{i-1} - AW(W^TAW)^{-1}\sigma$
  $p_i = r_i + \beta_i p_{i-1} - W(W^TAW)^{-1}\sigma$
  $x_{i+1} = x_i + \alpha_i p_i$
  $r_{i+1} = r_i - \alpha_i s_i$
  $w_{i+1} = w_i - \alpha_i z_i$
end for

- 2D 100^2 Poisson, $d = 20$

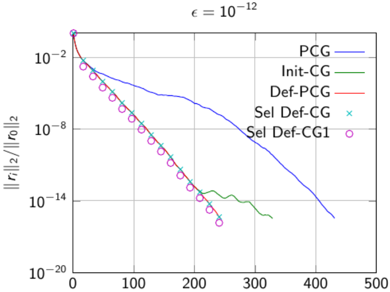


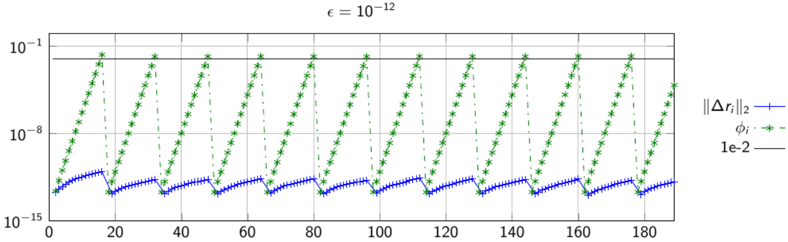
- True vs update residual

Selective Deflation

- $2D$ 100^2 Poisson equation
- $d = 20$

ϵ	10^{-12}	10^{-8}	10^{-4}
cg	431	431	431
init-cg	330	387	423
dcg	243	243	243
sdcg	242/16	242/22	242/42
sdcg1	244/16	244/23	244/41





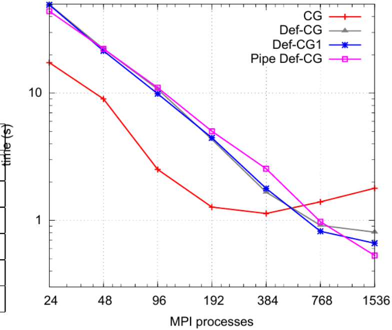
Sel-DCG1(A, M^{-1}, b, x_{-1}, W, λ_1, ϵ)

$x_0 = x_{-1} + W(W^TAW)^{-1}W^T(b - Ax_{-1})$
$r_0 = b - Ax_0$
$p_0 = r_0 - W(W^TAW)^{-1}W^TAr_0$
$\phi_0 = 0, \quad \psi_0 = 0$
for $i = 0, \ldots$ **do**
  $w = Ar_i$
  $\gamma_i = (r_i, r_i), \quad \delta = (w, r_i)$
  if $\psi_i/\|r_i\| > \tau$ **then**
    $\zeta = (W, r_i), \quad \eta = (W, w)$
  end if
  if $i > 0$ **then**
    $\beta_i = \gamma_i/\gamma_{i-1}, \quad \alpha_{i+1} = \gamma_i/(\delta - \beta_i\gamma_i/\alpha_i)$
  else
    $\beta_i = 0, \quad \alpha_{i+1} = \gamma_i/\delta$
  end if
  if $\psi_i/\|r_i\| > \tau$ **then**
    $r_i = r_i - W\left(W^TW\right)^{-1}\zeta$
    $p_i = r_i + \beta_i p_{i-1} - W\left(W^TAW\right)^{-1}\eta$
    $s_i = w_i + \beta_i s_{i-1} - AW\left(W^TAW\right)^{-1}\eta$
    $\phi_{i+1} = 0, \quad \psi_{i+1} = 0$
  else
    $p_i = r_i + \beta_i p_{i-1}$
  end if
  $x_{i+1} = x_i + \alpha_{i+1}p_i$
  $r_{i+1} = r_i - \alpha_{i+1}s$
  $\psi_{i+1} = \psi_i + \phi_i|\alpha_i|\|M^{-1}A\|$
  $\phi_{i+1} = \psi_{i+1} + \phi_i|\beta_i| + \epsilon\lambda_1^{-1}\|r_i\|\|W\|$
end for

Hopper – Cray XE6 at NERSC

- Gemini 3D-torus network

	time (s)			
	$P=24$	$P=1536$	it	#def
cg	17.4	1.791	2275	-
dcg	49.7	0.808	1084	1084
dcg1	49.4	0.662	1084	1084
pdcg	44.0	0.530	1084	1084
sdcg	11.8	0.665	1076	100
sdcg1	12.4	0.381	1084	100

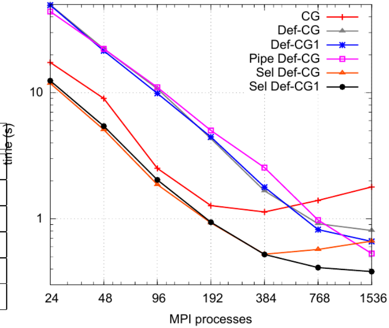


- Def-CG needs more flops but less iterations (less communication)
  - improved scalability
- DCG1 and pipe-DCG improve scalability further

Hopper – Cray XE6 at NERSC

- Gemini 3D-torus network

	time (s)			
	$P=24$	$P=1536$	it	#def
cg	17.4	1.791	2275	-
dcg	49.7	0.808	1084	1084
dcg1	49.4	0.662	1084	1084
pdcg	44.0	0.530	1084	1084
sdcg	11.8	0.665	1076	100
sdcg1	12.4	0.381	1084	100



- Def-CG needs more flops but less iterations (less communication)
  - improved scalability
- DCG1 and pipe-DCG improve scalability further
- Selective deflation reduces number of flops

Conclusions

- Two communication bottlenecks:
  - limited BW in Av. No benefit from SIMD.
  - Latencies in (u, v). Poor strong scaling.

Conclusions

- Two communication bottlenecks:
  - limited BW in Av. No benefit from SIMD.
  - Latencies in (u, v). Poor strong scaling.
- Replace Av with a $P_m(A)v$ with a fixed coefficients.
  - Execute with a stencil compiler.
  - Combine with Multigrid.
  - Many open question for unstructured matrices.

Conclusions

- Two communication bottlenecks:
  - limited BW in Av. No benefit from SIMD.
  - Latencies in (u, v). Poor strong scaling.
- Replace Av with a $P_m(A)v$ with a fixed coefficients.
  - Execute with a stencil compiler.
  - Combine with Multigrid.
  - Many open question for unstructured matrices.
- Pipelining of Krylov algorithms reduce the number of global reductions and hide their latencies.
  - They can scale as the Sparse Matrix-Vector product.
  - The pipelined GMRES and CG code is in PETSc-current.
  - Extension to Deflated and Augmented Krylov methods. Algorithms with additional global reductions.

Conclusions

- Two communication bottlenecks:
  - limited BW in Av. No benefit from SIMD.
  - Latencies in (u, v). Poor strong scaling.
- Replace Av with a $P_m(A)v$ with a fixed coefficients.
  - Execute with a stencil compiler.
  - Combine with Multigrid.
  - Many open question for unstructured matrices.
- Pipelining of Krylov algorithms reduce the number of global reductions and hide their latencies.
  - They can scale as the Sparse Matrix-Vector product.
  - The pipelined GMRES and CG code is in PETSc-current.
  - Extension to Deflated and Augmented Krylov methods. Algorithms with additional global reductions.

Acknowledgements:

- EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement n 610741.
- Intel and the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT).