Sparse Approximate Inverse Preconditioners Revisited

Salvatore Filippone   Daniele Bertaccini

salvatore.filippone@uniroma2.it
Università di Roma Tor Vergata

Due Giorni Algebra Lineare Numerica
Dip. Matematica, Uni. Genova, Sep. 16 2012

Outline

- Background: why sparse inverses;
- Approximate inverse algorithmic variants;
- Applications and tests;
- Recipes;
- Directions;

Background

Preconditioning for Krylov Methods

Krylov methods: search for the solution to $Ax = b$ in

$$x_m \in \{x_0 + \mathcal{K}_m\}, \quad b - Ax_m \perp \mathcal{L}_m$$

where

$$\mathcal{K}_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2 r_0, \ldots, A^{m-1} r_0\}$$

Now, choose a transformation M

$$Ax = b \Leftrightarrow M^{-1}Ax = M^{-1}b,$$

such that:

1. M can be computed easily;
2. M^{-1} can be applied easily;
3. $M^{-1}A \approx I$.

Time to solution:

$$T_{tot} = T_{prec} + N_{it} \times T_{it}$$

If you can find a good compromise, you'll get convergence in $N_{it} \ll n$ iterations at an affordable cost T_{it} Often: no convergence without preconditioner.

Approximate inverses

Seek a matrix $G = M^{-1}$ to minimize

$$\|I - GA\|_F^2$$

This approach is theoretically attractive but exceedingly expensive. Alternative strategies:

- Biconjugation (Benzi, Cullum, Tuma around 2000)
- Inversion of incomplete factors (Van Duin 1999)

Recent work by Rafiei, Bollhöfer (2011) on biconjugation. All of them compute

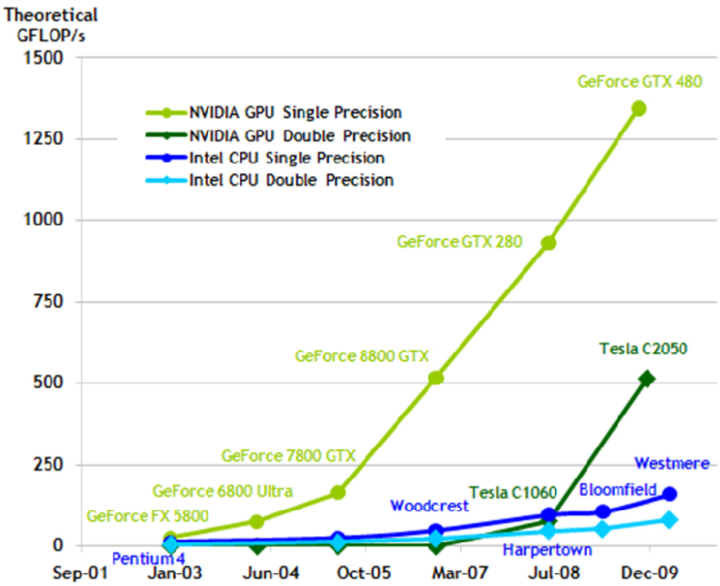
$$A^{-1} \approx ZD^{-1}W^T$$

Development directions: efficient and robust implementation, embed in multilevel and preconditioner update frameworks.
Joint work with D. Bertaccini, starting from (Benzi, Bertaccini: 2003), (Sgallari, Bertaccini 2010)

Approximate inverses: why?

Interest is renewed by the appearance on the computing scene of GPGPUs: fast becoming a key tool in scientific computing: price–performance ratio is extremely appealing.

Motivations



Motivations

Lots of work (including Barbieri, Cardellini and Filippone, 2009) on matrix multiply and other dense algorithms.

- Key algorithm (enabler for LAPACK);
- "Simple":
  - Can be studied thoroughly;
  - allows many variations;
- Reusable algorithmic patterns;

Situation MUCH more complicated for sparse kernels. In particular: Sparse triangular system solves are intractable (See Barbieri, Cardellini, Filippone and Rouson, 2011).
Machine hours grant PSBLAS-GPU at CASPUR, for a hybrid GPU-MPI version (under active development).

Approximate Inverses: Algorithmic variants

Approximate inverses: Biconjugation

Given two A-biconjugate sets of vectors W and Z we have

$$W^T AZ = D = \begin{pmatrix} p_1 & 0 & \dots & 0 \\ 0 & p_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & p_n \end{pmatrix};$$

it follows easily that

$$A^{-1} = \sum (1/p_i) z_i w_i^T.$$

An obvious idea is then:

> *Apply a biconjugation process with W and Z triangular, and sparsify while building.*

(Benzi & Tuma)
Critical issue: how many dot products do we actually perform.

Approximate inverses: Biconjugation

Right looking variant, stabilized (Benzi, Cullum, Tuma)

$w_i^{(0)} \leftarrow z_i^{(0)} \leftarrow e_i \qquad 1 \leq i \leq n$
for $i = 1, \ldots, n$ **do**
    $v_i \leftarrow A z_i^{(i-1)}$
    for $j = i, i+1, \ldots, n$ **do**
        $p_j^{(i-1)} \leftarrow v_i^T z_j^{(i-1)};$
    end for
    for $j = i+1, \ldots, n$ **do**
        $z_j^{(i)} \leftarrow z_j^{(i-1)} - \left(\frac{p_j^{(i-1)}}{p_i^{(i-1)}} \right) z_i^{(i-1)};$
    end for
end for
$z_i \leftarrow z_i^{(i-1)},\ p_i \leftarrow p_i^{(i-1)},\ 1 \leq i \leq n$

Better behaviour in difficult cases.

Approximate inverses: Biconjugation

Left looking variant

```
Let $z_1^{(0)} \leftarrow e_1$, $p_1^{(0)} \leftarrow a_{11}$
for $i = 2, \ldots, n$ do
    $z_i^{(0)} \leftarrow e_i$
    for $j = 1, \ldots, i-1$ do
        $p_i^{(j-1)} \leftarrow a_j^T z_i^{(j-1)}$
        $z_i^{(j)} \leftarrow z_i^{(j-1)} - \left(\frac{p_i^{(j-1)}}{p_j^{(j-1)}}\right) z_j^{(j-1)}$
    end for
    $p_i^{(i-1)} \leftarrow a_i^T z_i^{(i-1)}$
end for
```

Note: the drop strategy is applied just once per column, to the end result of all updates.

Approximate inverses: sparse inversion of sparse factors

Given one factor of an LDU decomposition

$$U = I + \sum e_i u_i^T = \prod_{i=n-1}^{1} (I + e_i u_i^T)$$

its inverse is also triangular; thus

$$U^{-1} = \prod_{i=1}^{n-1} (I - e_i u_i^T) = I + \sum e_i \hat{u}_i^T$$

and therefore

$$\hat{u}_i^T = -u_i^T \prod_{j=i+1}^{n-1} (I - e_j u_j^T)$$

It is natural to

1. Start from a sparse factor
2. Apply a drop strategy to avoid a dense inverse

Numerical drop strategy:

```
for j = 1 to n − 1 do
    û_i^T ← −u_i^T
    j location of first nonzero in û_i^T
    while j < n do
        α ← −û_i^T e_j
        if |α| > ε then
            û_i^T ← û_i^T + α u_j^T
        else
            û_i^T(j) ← 0
        end if
        j location of next nonzero in û_i^T
    end while
end for
```

Can be implemented reusing all the building blocks of ILUT.

Approximate inverses: sparse inversion of sparse factors

Fill level drop strategy:

for $j = 1$ to $n - 1$ **do**
  $\hat{u}_i^T \leftarrow -u_i^T$
  j location of first nonzero in $\hat{u}_i^T$
  while $j < n$ **do**
    if $level_{ij} \leq p$ **then**
      $\alpha \leftarrow -\hat{u}_i^T e_j$
      $\hat{u}_i^T \leftarrow \hat{u}_i^T + \alpha u_j^T$
      update fill levels $level_{ik} = \min(level_{ik}, level_{ij} + 1)$
    else
      $\hat{u}_i^T(j) \leftarrow 0$
    end if
    j location of next nonzero in $\hat{u}_i^T$
  end while
end for

Again, easy to implement from the building blocks of ILU.

Approximate inverses: sparse inversion of sparse factors

Need to adjust the input matrix: we apply a scaling factor α given by the maximum absolute value of the coefficients in A to compute

$$\alpha^{-1}A = LDU,$$

and therefore the input to the inversion step is given by

$$A = L(\alpha D)U.$$

For numerical drop tolerance this improves behaviour on nonsymmetric matrices; level-fill strategies are a bit more robust.
To be investigated: scale each row independently.

Approximate inverses: computational load

From Van Duin:

$$\begin{aligned} C_{\text{fact}} &= O\left(nnz_L \frac{nnz_U}{n}\right) \\ C_{\text{invrt}} &= O\left(nnz_{\hat{U}} \frac{nnz_U}{n}\right) \\ C_{\text{subst}} &= O\left(nnz_U\right) \end{aligned}$$

This is true of the sparse inversion of sparse factos; biconjugation is much more expensive.

Problem: this is "just" a FLOP count. Does not take into account many other issues, such as:

- Efficiency of search through the nonzeros;
- Size of resulting data structure;
- Memory access patterns.

In any case:

Approximate inverses are VERY expensive to compute;

Hence the interest of update formulations.

Approximate inverses: Algorithmic Variants on GPU

2D convection-diffusion PDE, centered finite differences, Dirichlet unit square, AMD processor, NVIDIA GTX 285.

Case	size	NOPREC			AINVK 0,1			AINVT 0,1,.01,.001		
		tpr	it	tslv	tpr	it	tslv	tpr	it	tslv
pde0100	10000	0.0	214	0.11	0.05	74	0.07	0.40	110	0.09
pde0200	40000	0.0	423	0.52	0.19	154	0.29	0.81	215	0.38
pde0300	90000	0.0	629	0.88	0.43	242	0.58	1.64	326	0.74
pde0400	160000	0.0	832	1.36	0.77	325	0.92	2.85	430	1.11
pde0500	250000	0.0	1043	1.99	1.19	394	1.34	4.40	543	1.64
pde0600	360000	0.0	1246	2.88	1.71	481	1.96	6.08	639	2.29
pde0700	490000	0.0	1476	4.22	2.33	556	2.74	8.28	755	3.21
pde0800	640000	0.0	1749	5.49	3.12	654	3.84	10.59	867	4.36
pde0900	810000	0.0	1965	7.47	3.93	724	5.10	13.54	1042	6.17
pde1000	1000000	0.0	† 2000	8.78	4.90	810	6.64	16.64	1106	7.64
pde1100	1210000	0.0	† 2000	10.19	5.95	883	8.49	19.89	1280	10.34

Note: preconditioner computed on CPU, solve on GPU. Different number of nonzeros among the two variants.

Approximate inverses: Algorithmic Variants on GPU

Case	size	AINVT 0,1,.01,.001			ORTH LL1,5,1e-5			ORTH LL2,5,1e-5		
		tpr	it	tslv	tpr	it	tslv			
pde0100	10000	0.4	110	0.09	0.2	66	0.06	0.3	66	0.06
pde0200	40000	0.8	215	0.38	1.0	146	0.27	1.4	146	0.27
pde0300	90000	1.6	326	0.74	2.7	233	0.56	3.4	233	0.56
pde0400	160000	2.8	430	1.11	5.6	318	0.88	6.0	318	0.88
pde0500	250000	4.4	543	1.64	10.1	467	1.54	9.4	467	1.54
pde0600	360000	6.1	639	2.29	16.8	553	2.21	14.2	553	2.21
pde0700	490000	8.3	755	3.21	25.3	844	3.99	19.6	844	4.00
pde0800	640000	10.6	867	4.36	36.8	1024	5.78	25.8	1024	5.76
pde0900	810000	13.5	1042	6.17	50.4	841	5.68	36.2	841	5.69
pde1000	1000000	16.6	1106	7.64	67.9	1195	9.39	43.1	1195	9.41
pde1100	1210000	19.9	1280	10.34	87.8	1344	12.58	53.2	1344	12.57

Note: preconditioner computed on CPU, solve on GPU

Approximate inverses: Algorithmic Variants on GPU

3D convection-diffusion PDE, similar structure to the 2D

Case	size	NOPREC			INVK 0,1			AORTH LL2,5,1e-4		
					tpr	it	tslv	tpr	it	tslv
pde010	1000	0.0	69	0.034	0.014	59	0.042	0.247	22	0.019
pde020	8000	0.0	80	0.043	0.066	21	0.020	1.474	25	0.023
pde030	27000	0.0	85	0.114	0.231	30	0.063	4.559	38	0.072
pde040	64000	0.0	113	0.169	0.563	41	0.105	13.287	51	0.120
pde050	125000	0.0	141	0.239	1.126	53	0.171	27.117	65	0.193
pde060	216000	0.0	173	0.349	1.927	64	0.252	47.647	77	0.263
pde070	343000	0.0	197	0.483	3.115	76	0.376	77.108	89	0.376
pde080	512000	0.0	226	0.690	4.840	88	0.554	146.303	102	0.542
pde090	729000	0.0	254	0.981	6.937	99	0.818	209.359	114	0.786
pde100	1000000	0.0	282	1.365	9.573	110	1.164	276.180	127	1.108

Note: at these sizes the problem is biased to favour NOPREC, but the solve time is still better for the preconditioned versions.

Approximate inverses: Algorithmic Variants

Right-looking orthogonalization variant quickly becomes impractical

Case	size	AORTH LL1, 5, 1e-5			AORTH LL2, 5, 1e-5			AORTH RL, 5, 1e-5		
		tpr	it	tslv	tpr	it	tslv	tpr	it	tslv
pde010	1000	0.0	24	0.015	0.1	24	0.015	0.04	22	0.019
pde020	8000	0.3	22	0.015	0.8	22	0.015	2.03	23	0.021
pde030	27000	1.6	31	0.044	3.1	31	0.043	35.23	31	0.064
pde040	64000	5.4	40	0.090	7.8	40	0.076	238.15	37	0.095
pde050	125000	14.3	48	0.129	15.6	48	0.124	871.01	45	0.136
pde060	216000	32.5	57	0.187	27.9	57	0.179	2538.94	54	0.194
pde070	343000	66.1	65	0.264	45.1	65	0.256	6425.99	62	0.276
pde080	512000	126.3	74	0.383	77.8	74	0.375			
pde090	729000	225.6	83	0.551	112.8	83	0.542			
pde100	1000000	375.0	91	0.772	154.4	91	0.760			

Approximate inverses: Engine application examples

Case	size	NOPREC			AINVK 0,1			AINVT 0,1,.01,.001		
		tpr	it	tslv	tpr	it	tslv	tpr	it	tslv
kivap001	86304	0.0	99	0.193	4.31	22	0.102	5.120	26	0.075
kivap002	76504	0.0	186	0.330	3.86	50	0.213	5.420	62	0.172
kivap003	59354	0.0	247	0.399	2.88	57	0.189	4.100	81	0.184
kivap004	42204	0.0	643	0.969	2.04	190	0.546	2.230	148	0.306
kivap005	25054	0.0	800	1.057	1.18	800	1.686			
kivap006	42204	0.0	745	1.105	2.05	166	0.477	2.030	230	0.468
kivap007	56904	0.0	251	0.400	2.77	51	0.165	3.940	69	0.154
kivap008	76504	0.0	179	0.318	3.86	47	0.199	5.430	62	0.171
kivap009	86304	0.0	201	0.369	4.30	50	0.226	5.850	61	0.175

Note: kivap005 very difficult to control.

Test case from KIVA, see Bella et al, HPCC 2005.

Approximate inverses: Engine application examples

Case	size	NOPREC			AORTH LL 5, 1e-5			AINVT 5,5,.001,.0001		
		tpr	it	tslv	tpr	it	tslv	tpr	it	tslv
kivap001	86304	0.0	99	0.193	23.38	35	0.099	25.61	22	0.081
kivap002	76504	0.0	186	0.330	19.13	81	0.217	29.42	48	0.165
kivap003	59354	0.0	247	0.399	12.78	90	0.197	18.90	65	0.182
kivap004	42204	0.0	643	0.969	6.77	134	0.274	8.73	141	0.354
kivap005	25054	0.0	800	1.057	2.88	100	0.158	3.98	219	0.398
kivap006	42204	0.0	745	1.105	6.73	128	0.260	8.31	175	0.435
kivap007	56904	0.0	251	0.400	11.82	95	0.204	17.95	67	0.184
kivap008	76504	0.0	179	0.318	19.11	71	0.188	29.21	50	0.172
kivap009	86304	0.0	201	0.369	23.48	72	0.200	34.60	44	0.161

AINVT with these parameters is better for the easier cases, but it costs a lot.

Approximate inverses: Florida Sparse Collection

Case	size	IK 0,1			ORTH LL 5, 1e-5			IT 0,1,.01,.001		
		tpr	it	tslv	tpr	it	tslv	tpr	it	tslv
bcsstk04	132	0.0	39	0.033	0.0	78	0.054	0.0	800	0.577
bcsstk05	153	0.0	44	0.031	0.0	310	0.218	0.0	64	0.043
bcsstk07	420	0.0	800	0.549	0.0	800	0.763	0.0	800	0.543
bcsstk08	1074	0.1	46	0.121	0.1	137	0.176	0.0	800	1.293
bcsstk09	1083	0.0	800	0.557	0.0	484	0.304	0.1	287	0.185
bcsstk16	4884	0.9	35	0.043	0.6	67	0.050	0.9	31	0.027
bcsstk18	11948	0.4	553	0.619	1.1	800	1.663	0.0	0	0.000
bcsstk21	3600	0.0	800	0.491	0.1	800	0.540	0.1	162	0.097
bcsstk22	138	0.0	95	0.058	0.0	97	0.057	0.0	73	0.043
bcsstk23	3134	0.1	800	0.654	0.5	800	3.354	0.0	0	0.000
bcsstk27	1224	0.1	64	0.056	0.1	800	0.766	0.2	800	0.733
sherman4	1104	0.0	28	0.018	0.0	29	0.018	0.0	32	0.019
sherman5	3312	0.0	23	0.016	0.1	55	0.035	0.1	800	0.537
t1000x600	600000	2.9	470	2.598	* 23.0	800	4.264	10.1	657	3.073
A-1M	995100	6.3	59	0.662	* 245.7	73	0.709	15.6	85	0.686
A-500k	531612	4.3	74	0.640	* 49.1	98	0.603	7.0	96	0.581

* result obtained with Cuthill-McKee renumbering

Approximate inverses: comparison

INVK: Very fast setup; memory occupation grows quickly with level of fill;

INVT: Need to choose four parameters; very dependent upon scaling (PDE example without scaling quickly gets out of control); allows better control on the number of nonzeros, and puts to good use the freedom in their placement;

AORTH RL: Can be made very robust, best number of iterations with respect to nonzero count (but not by much); very expensive to compute;

AORTH LL: Much better computation cost, similar quality (wrt RL); two implementation variants, depends on matrix size/pattern.

Approximate inverses: Effects of Renumbering

Alternatives:

- Band reducing algorithms (Reverse Cuthill-McKee, using the variant of Gibbs-Poole-Stockmeyer)
- Fill reducing (Approximate Miminum Degree, Davis, Dufff and Amestoy)

Should we favour AMD as it reduces FLOPS?

Approximate inverses: Effects of Renumbering INVK

Case	size	IK 0,1 NONE			IK 0,1 AMD			IK 0,1 GPS		
		tpr	it	tslv	tpr	it	tslv	tpr	it	tslv
bcsstk04	132	0.0	39	0.033	0.0	32	0.028	0.0	55	0.045
bcsstk05	153	0.0	44	0.031	0.0	48	0.035	0.0	44	0.030
bcsstk07	420	0.0	800	0.549	0.0	800	0.628	0.0	800	0.548
bcsstk08	1074	0.1	46	0.121	0.1	34	0.089	0.1	55	0.139
bcsstk09	1083	0.0	800	0.557	0.0	800	0.640	0.0	800	0.555
bcsstk16	4884	0.9	35	0.043	0.8	37	0.061	0.9	39	0.048
bcsstk22	138	0.0	95	0.058	0.0	81	0.050	0.0	81	0.050
bcsstk23	3134	0.1	800	0.654	0.1	800	0.750	0.1	800	0.661
bcsstk27	1224	0.1	64	0.056	0.1	64	0.065	0.1	72	0.062
kivap001	86304	4.4	21	0.100	4.4	25	0.156	4.3	22	0.102
kivap002	76504	4.0	49	0.211	3.9	55	0.308	3.9	50	0.213
kivap003	59354	3.0	56	0.197	3.0	66	0.288	2.9	57	0.189
kivap004	42204	2.0	139	0.402	2.0	178	0.623	2.0	190	0.546
kivap006	42204	2.0	223	0.646	2.0	204	0.711	2.0	166	0.477
kivap007	56904	2.9	55	0.188	2.8	61	0.260	2.8	51	0.165
kivap008	76504	3.9	49	0.210	3.9	52	0.289	3.9	47	0.199
kivap009	86304	4.4	46	0.216	4.3	53	0.327	4.3	50	0.226
t1000x600	600000	2.9	470	2.598	0.0	0	0.000	2.8	800	4.426
A-1M	995100	6.3	59	0.662	0.0	0	0.000	6.3	59	0.660
A-500k	531612	4.3	74	0.640	0.0	0	0.000	3.5	91	0.655

Approximate inverses: Effects of Renumbering ORTH LL

Case	size	LL 5,1e-5 NONE			LL 5,1e-5 GPS		
		tpr	it	tslv	tpr	it	tslv
bcsstk04	132	0.0	78	0.054	0.0	82	0.055
bcsstk05	153	0.0	310	0.218	0.0	159	0.103
bcsstk07	420	0.0	800	0.763	0.0	800	0.761
bcsstk08	1074	0.1	137	0.176	0.1	139	0.169
bcsstk09	1083	0.0	484	0.304	0.0	484	0.303
bcsstk16	4884	0.6	67	0.050	0.6	84	0.064
bcsstk18	11948	1.1	800	1.663	0.8	800	1.032
bcsstk22	138	0.0	97	0.057	0.0	491	0.294
bcsstk23	3134	0.5	800	3.354	0.3	800	2.050
bcsstk27	1224	0.1	800	0.766	0.1	800	0.764
kivap001	86304	25.3	31	0.087	23.4	35	0.099
kivap002	76504	20.1	75	0.197	19.1	81	0.217
kivap003	59354	13.8	100	0.220	12.8	90	0.197
kivap004	42204	6.6	138	0.278	6.8	134	0.274
kivap005	25054	2.8	49	0.076	2.9	100	0.158
kivap006	42204	6.5	132	0.265	6.7	128	0.260
kivap007	56904	13.0	95	0.207	11.8	95	0.204
kivap008	76504	20.1	75	0.198	19.1	71	0.188
kivap009	86304	25.2	75	0.205	23.5	72	0.200

Approximate inverses: Effects of Renumbering INVT

Case	size	IT 0,1,.01,.001 NONE			AMD			GPS		
		tpr	it	tslv	tpr	it	tslv	tpr	it	tslv
bcsstk04	132	0.0	800	0.577	0.0	800	0.599	0.0	800	0.578
bcsstk05	153	0.0	64	0.043	0.0	800	0.521	0.0	58	0.038
bcsstk07	420	0.0	800	0.543	0.0	800	0.542	0.0	800	0.544
bcsstk08	1074	0.0	800	1.293	0.0	0	0.000	0.1	800	1.494
bcsstk09	1083	0.1	287	0.185	0.1	800	0.847	0.1	287	0.185
bcsstk16	4884	0.9	31	0.027	0.7	36	0.035	0.9	38	0.032
bcsstk18	11948	0.0	0	0.000	0.0	0	0.000	0.0	0	0.000
bcsstk22	138	0.0	73	0.043	0.0	800	0.475	0.0	800	0.474
bcsstk23	3134	0.0	0	0.000	0.0	0	0.000	0.0	0	0.000
kivap001	86304	4.5	25	0.073	3.7	26	0.092	5.1	26	0.075
kivap002	76504	4.9	66	0.179	3.6	61	0.202	5.4	62	0.172
kivap003	59354	4.1	88	0.201	2.9	71	0.191	4.1	81	0.184
kivap004	42204	2.4	156	0.314	1.6	113	0.257	2.2	148	0.306
kivap006	42204	2.1	166	0.331	1.5	196	0.443	2.0	230	0.468
kivap007	56904	4.0	76	0.170	2.8	82	0.218	3.9	69	0.154
kivap008	76504	4.9	65	0.177	3.7	59	0.196	5.4	62	0.171
kivap009	86304	5.1	59	0.166	3.9	56	0.197	5.8	61	0.175

Approximate inverses: Effects of Renumbering INVT

Case	size	IT 0,1,.01,.001 NONE			AMD			GPS		
		tpr	it	tslv	tpr	it	tslv	tpr	it	tslv
pde-20	8000	0.2	22	0.014	0.1	19	0.013	0.1	22	0.014
pde-50	125000	3.3	44	0.107	2.8	43	0.131	2.8	43	0.106
pde-60	216000	5.8	51	0.152	5.1	50	0.211	4.8	49	0.147
pde-80	512000	13.9	69	0.323	13.3	63	0.570	11.8	66	0.313
pde-90	729000	19.9	72	0.433	19.6	77	1.000	16.8	67	0.408
pde-100	1000000	26.7	75	0.574	28.1	76	1.366	22.9	75	0.580
lp600x600	360000	6.1	395	1.361	4.4	596	2.571	5.5	518	1.787
A-1M	995100	15.6	85	0.686	14.2	92	1.447	15.5	85	0.685
A-500k	531612	7.0	96	0.581	8.0	115	1.000	8.7	122	0.628

Lessons learned (so far)

1. Approximate inverses work, but are skittish (i.e. don't try this at home kid, not yet);
2. It is very dangerous to estimate costs from small/special pattern test cases;
3. It is very dangerous to estimate costs from floating-point operations only;
4. Robustness over multiple problem sets?
5. Non obvious results of renumbering schemes;
6. GPU performance on sparse kernels is much less than we would like: bandwidth limited.

Research directions

1. Formalize updates of preconditioners for sequences of linear systems (might be implemented on the GPU);
2. Is it possible to discover optimal parameters?
3. Is there a way to build on the GPU? Absolutely non-trivial (probably hopeless);
4. Embed in multilevel/domain decomposition framework;

Also, investigate other dependencies on the GPU architecture (which is evolving quickly!)

Collaborations & credits

- Daniele Bertaccini, Maths, Univ. Rome Tor Vergata;
- Alfredo Buttari, CNRS-IRIT, Toulouse
- Damian Rouson, Sandia Nat. Lab, Livermore (CA);
- Valeria Cardellini, Comp. Sci., Univ. Rome Tor Vergata;
- Marco Rorro, CASPUR;
- Pasqua D'Ambra, CNR-ICAR, Naples
- Daniela di Serafino, Maths, II Univ. Naples, Caserta

Bibliography

- Benzi, Bertaccini: Approximate Inverse Preconditioning for Shifted Linear Systems, BIT, 2003
- Bertaccini, Sgallari: Updated Preconditioners for nonlinear deblurring and denoising image restoration, APNUM 2010.
- D. Barbieri, V. Cardellini, S. Filippone, Generalized GEMM kernels on GPGPUs: experiments and applications. Proceedings of PARCO 2009,Lyon
- D. Barbieri, V. Cardellini, S. Filippone, D. Rouson, Design Patterns for Scientific Computations on Sparse Matrices, EuroPAR 2011, Bordeaux.
- D. Barbieri, V. Cardellini, S. Filippone Sparse computations on GPGPUs Technical Report RR-12.90, Dip. di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Italy, Jan 2012.
- A. Buttari, D. di Serafino, P. D'Ambra, S. Filippone: 2LEV-D2P4: a package of high-performance preconditioners, AAECC, 2007.
- A. Buttari, V. Eijkhout, J. Langou and S. Filippone: Performance optimization and modeling of sparse kernels, IJHPCA, 2007.
- P. D'Ambra, S. Filippone, D. di Serafino: On the Development of PSBLAS-based Parallel Two-level Schwarz Preconditioners, APNUM 2007
- P. D'Ambra, S. Filippone, D. di Serafino: MLD2P4: A Package of Parallel Algebraic Multilevel Domain Decomposition Preconditioners in Fortran 95, ACM TOMS 2010
- S. Filippone, A. Buttari: Object-Oriented Techniques for Sparse Matrix Computations in Fortran 2003, ACM TOMS, to appear.
- S. Filippone and M. Colajanni: PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices, ACM TOMS 2000

Thank you for your attention