Efficient sparse linear solvers and AMG preconditioners on cluster of GPUs: first results on linear systems from ParFlow

Pasqua D'Ambra and Salvatore Filippone

Institute for Applied Computing (IAC) "Mauro Picone"
CNR, Naples, Italy

EoCoE F2F Meeting
Bath, 11-13 April, 2018

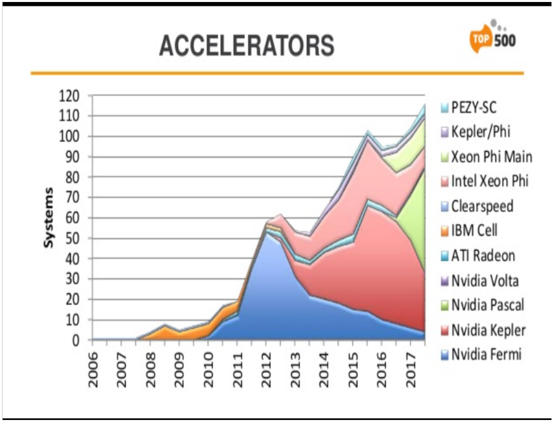
Collaborators:

- Ambra Abdullahi Hassan, University of Rome “Tor-Vergata”
- Massimo Bernaschi, IAC-CNR, Rome
- Daniele Bertaccini, University of Rome "Tor-Vergata"
- Valeria Cardellini, University of Rome "Tor-Vergata"
- Daniela di Serafino, University of Campania “L. Vanvitelli”
- Dario Pasquini, University of Rome “La Sapienza” and IAC-CNR, Rome
- Damian Rouson, Sourcery Inc., (Berkeley, CA), USA
- Panayot S. Vassilevski, CASC-LLNL (Livermore, CA) and Portland State University (Portland, OR), USA

Motivation to move PSBLAS & MLD2P4 towards GPUs:

Highlights of the 50th TOP500 List (November 2017)

Many machines have embedded GPUs or similar devices featuring thousands of simple cores



PSBLAS: Parallel Sparse BLAS

A Software development project started by S. Filippone at the end of 90's and stimulated by the work of Iain Duff et al. on standard for Sparse BLAS, ACM TOMS 23 (1997)

- parallel sparse matrix operations and data management, Krylov solvers (for spd and general matrices)
- General row-block matrix distribution, support infrastructure for mesh handling and sparse matrix I/O
- Data allocation through graph partitioning (METIS, ParMETIS, SCOTCH)
- Object oriented design in Fortran 2003
- Message-passing paradigm (MPI), plugin available for NVIDIA Cuda
- Internal matrix representation/storage: distributed sparse matrix in CSR/CSC/COO format (Sparse Matrix class) in the base library, extension plugins available for ELLPACK, JAD and GPU-enabled formats (e.g., hlg, csrg, elg)

Freely available from `https://github.com/sfilippone/psblas3`

PSBLAS extensions for EoCoE

Some extensions was prompted by the needs of EoCoE applications:

- new Krylov solvers (Flexible CG and Generalized CR) have been developed for using variable preconditioning
- software improvements to GPU plugin for run-time support to different sparse matrix storage schemes based on design patterns:
  - Data storage formats are essential for efficiency of sparse matrix computations, e.g., in sparse matrix-vector multiplication
  - Different computer architectures are best exploited by different formats
  - We want to be able to change the formats in response to machine changes and usage requirements
  - We want user's interface (almost) independent of the target machine

  Cardellini et al., Design Patterns for sparse-matrix computations on hybrid CPU/GPU platforms, Scientific Programming, 22, 2014;
  Filippone et al., Sparse matrix-vector multiplication on GPGPUs, ACM TOMS, 43, 2017.

- C and Octave interfaces (work in progress)

MLD2P4: Parallel Preconditioners based on PSBLAS

A software development project started by P. D'Ambra, D. di Serafino and S. Filippone in 2004

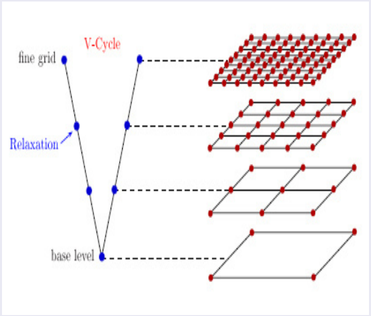
- Initially developed as a package of algebraic multigrid Schwarz preconditioners, extended to more general AMG preconditioning within EoCoE
- Object-oriented design in Fortran 2003, layered sw architecture on top of PSBLAS
  $\Longrightarrow$ modularity and flexibility
- Clear separation between interface and implementation of methods
  $\Longrightarrow$ performance and extensibility
- Separated users' interface for setup of the multigrid hierarchy and setup of the smoothers and solvers to have large flexibility at each level.
- Plugin for approximate inverses by Filippone et al., 2016
- C and Octave interfaces (work in progress)

Freely available from `https://github.com/sfilippone/mld2p4-2`

MLD2P4: Parallel AMG based on PSBLAS

Example: symmetric V-cycle

```
procedure V-cycle$(k, A^k, b^k, x^k)$
    if $(k \neq nlev)$ then
        $x^k = x^k + (M^k)^{-1}(b^k - A^k x^k)$
        $b^{k+1} = (P^{k+1})^T\,(b^k - A^k x^k)$
        $x^{k+1} =$ V-cycle$(k+1, A^{k+1}, b^{k+1}, 0)$
        $x^k = x^k + P^{k+1} x^{k+1}$
        $x^k = x^k + (M^k)^{-T}(b^k - A^k x^k)$
    else
        $x^k = \left(A^k\right)^{-1} b^k$
    endif
    return $x^k$
end
```



AMG methods do not explicitly use the (eventual) problem geometry and rely only on matrix entries to generate coarse grids (setup phase)

MLD2P4 plugin for Approximate Inverse (MLD-AINV)

Approximate Inverses

Compute a (sparse) matrix G approximating **the inverse of the matrix** A:

$$G \approx A^{-1}$$

so that the preconditioning operation reduces to a matrix-vector multiplication with G.

Alternative strategies in MLD-AINV:

AINV	Approximate bi-conjugation (Benzi et al., 1996): $W^T A Z = D \rightarrow A^{-1} = Z D^{-1} W^T$
INVK	Inversion of incomplete factors (e.g., Van Duin 1999): $A^{-1} \approx \overline{U}^{-1} \overline{L}^{-1}$

MLD2P4 plugin for Approximate Inverse (MLD-AINV)

Approximate Inverses

Compute a (sparse) matrix G approximating **the inverse of the matrix** A:

$$G \approx A^{-1}$$

so that the preconditioning operation reduces to a matrix-vector multiplication with G.

Alternative strategies in MLD-AINV:

AINV	Approximate bi-conjugation (Benzi et al., 1996): $W^T AZ = D \rightarrow A^{-1} = ZD^{-1}W^T$
INVK	Inversion of incomplete factors (e.g., Van Duin 1999): $A^{-1} \approx \overline{U}^{-1}\overline{L}^{-1}$

application of MLD-AINV exploiting GPU plugin of PSBLAS "moves" MLD2P4 preconditioners towards GPU

Abdullahi Hassan et al., Efficient algebraic multigrid preconditioners on cluster of GPUs, submitted to Euro-Par 2018

MLD2P4 extensions for EoCoE

Current version of MLD2P4 preconditioners can be obtained as any combination of

setup or coarsening phase: **GPU implementation is work in progress**

- decoupled smoothed aggregation based on the usual strength of connection measure (Vaněk and Brezina, 1996)
- plug in for decoupled aggregation based on compatible weighted matching (D'Ambra et al., 2013, 2016, 2018)
- distributed or replicated coarsest matrix

solve phase: **already available on GPU** for some choices of smoothers & coarsest solver

- cycles: V, W, K
- smoothers: Jacobi, hybrid (F/B) Gauss-Seidel, block-Jacobi / additive Schwarz with LU, ILU factorizations or sparse approximate inverses the blocks
- coarsest-matrix solvers: sparse LU, Jacobi, hybrid (F/B) Gauss-Seidel, block-Jacobi with LU, ILU factorizations or sparse approximate inverses of the blocks
- LU factorizations for smoothers & coarsest-level solvers: UMFPACK, MUMPS, SuperLU, SuperLU_Dist

Application code for CPU/GPU

```
! sparse matrix
type(psb_dspmat_type) :: A
! variable declaration needed for GPU running
type(psb_d_hlg_sparse_mat), target   :: ahlg
type(psb_d_vect_gpu)  :: vgmold
type(psb_i_vect_gpu)  :: igmold
! sparse matrix descriptor
type(psb_desc_type)   :: DESC_A
! preconditioner data
type(mld_dprec_type)  :: P

...
! inizialize parallel environment
call psb_init(ictxt)
call psb_info(ictxt,iam,np)

...
! read and assemble matrix A and rhs b using PSBLAS facilities
...
 ! setup preconditioner
call P%init('ML', info)
call P%set(<attribute>, value, info)
...
call P%set(<attribute>, value, info)
```

Application code for CPU/GPU (cont'd)

```
  ...
! build preconditioner
  call P%hierarchy_build(A,DESCA,info)
! last three optional parameters needed for GPU unning
  call P%smoothers_build(A,DESCA,info,amold=ahlg, vmold=vgmold, imold=igmold)
! conversions and vector assembly needed for GPU running
  call DESCA%cnv(mold=igmold)
  call A%cscnv(info,mold=ahlg)
  call psb_geasb(x,DESC_A,info,mold=vgmold)
  call psb_geasb(b,DESC_A,info,mold=vgmold)
! set solver parameters and initial guess
  ...
! solve Ax=b with precond RGMRES
 call psb_krylov('RGMRES',A,P,b,x,tol,DESC_A,info,...)
  ...
! cleanup storage
  call P%free(info)
  ...
!
! leave PSBLAS
  call psb_exit(ictxt)
```

Parflow Model

High resolution simulations of subsurface flow for regional hydrology studies

Richard's equation

Filtration through variably saturated porous media for incompressible flows (3D model based on Darcy's law):

$$\begin{aligned} \frac{\partial(\Phi s(p))}{\partial t} + \nabla \cdot \mathbf{u} &= f \\ \mathbf{u} &= -\mathbf{K}\nabla(p - z) \end{aligned}$$

- implicit time integration method
- finite difference discretization of spatial operator on a structured Cartesian mesh
- Newton-Krylov solver for non-linear algebraic equation by KINSOL coupled with a linear geometric preconditioner by Hypre
- MPI-based parallel code written in C

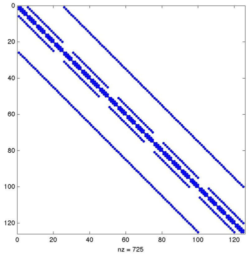
Test Cases for our experiments with PSBLAS and MLD2P4

Simplified steady-state model

$$-\nabla \cdot \mathbf{K}\nabla p = f$$

on unit cube, with no-flow boundary conditions

- discretization obtained by a Fortran code reproducing the Matlab mini-app provided by JSC
- anisotropic conductivity tensor: randomly generated from lognormal distribution with mean $\mu = 1$ and variable standard deviation $\sigma = 1, 2, 3$, corresponding to M1, M2 and M3 linear systems, respectively
- cartesian grid with uniform refinement along the coordinates for increasing mesh size
- hepta-diagonal spd matrices



Weak Scalability on the Jureca cluster operated by JSC

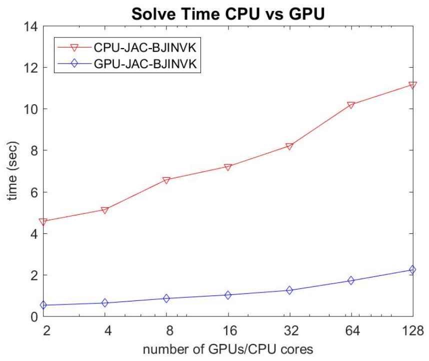
Selected PSBLAS/MLD2P4 preconditioned iterative solvers:

- Krylov Solver: Conjugate Gradient, with stopping criterion $\|r_k\| \leq 10^{-6}\|r_0\|$
- Preconditioners:
  - AMG based on decoupled smoothed aggregation by Vaněk and Brezina
  - V-cycle with 2 point-wise Jacobi sweeps as pre/post-smoother and 10 block-Jacobi sweeps on the (distributed) coarsest matrix (V-JAC-BJINVK)
  - V-cycle with 1 block-Jacobi sweep as pre/post-smoother and 10 block-Jacobi sweeps on the (distributed) coarsest matrix (V-BJINVK-BJINVK)
    Approximate inverse (INVK) by MLD-AINV plugin is applied to the blocks in the block-Jacobi sweeps

All the solve phase is based on PSBLAS plugin for GPUs and runs in a hybrid distributed/shared model exploiting GPU accelerators

Machine Configuration:
1872 compute nodes with 2 Intel Xeon E5-2680 v3 Haswell CPUs per node, 75 compute nodes equipped with 2 NVIDIA Tesla K80 GPUs, Infiniband connection

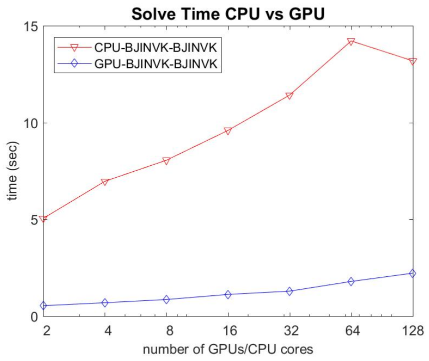
Weak Scalability on the Jureca cluster operated by JSC



NP	Levels	Iters	Titer (ms)
2	4	21	220/20
4	4	20	260/30
8	4	25	260/30
16	4	27	270/40
32	4	31	260/40
64	4	37	280/50
128	5	31	360/70

Row-block distribution of the matrix obtained by a 3d decomposition of the grid;
M1 matrix with 2×10^6 rows (dofs) per core up to 2.56×10^8 dofs on 128 GPUs

Weak Scalability on the Jureca cluster operated by JSC

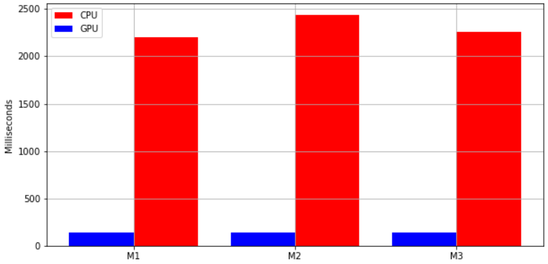


NP	Levels	Iters	Titer (ms)
2	4	15	340/40
4	4	18	390/40
8	4	20	400/40
16	4	24	400/50
32	4	29	390/40
64	4	34	420/50
128	5	29	450/80

Row-block distribution of the matrix obtained by a 3d decomposition of the grid;
M1 matrix with 2×10^6 rows (dofs) per core up to 2.56×10^8 dofs on 128 GPUs

Preliminary Tests for AMG Setup

Setup time for AMG based on compatible weighted matching: CPU vs GPU



1 core of Intel Xeon Platinum 8176 Processor vs 1 NVIDIA Volta Titan V
M1, M2 and M3 matrices with 10^6 rows.

D'Ambra et al., AMG based on compatible weighted matching on GPUs, submitted to PMAA18

Workplan proposed for EoCoE II

- co-design of the interface between PSBLAS/MLD2P4 and ParFlow
- improvement of the GPU support of the two libraries
- integration of parallel coupled matching-based aggregation
- new smoothers and coarse-level solvers based on sparse approximate inverses
- design of strategies for reusing preconditioners in multiple steps of Newton-type nonlinear solvers
- tuning and testing of the solver in complete ParFlow simulations
- *Construction of an interface for sliding meshes and support for remeshing in Alya*

Thanks for Your Attention