Performance of computing low-rank matrix approximatoin on a hybrid CPU/GPU architecture

Ichitaro Yamazaki[1], Theo Mary[2],
Jakub Kurzak[1], Stanimire Tomov[1], Jack Dongrra[1]

[1]University of Tennessee, Knoxville, USA
[2]Universite de Toulouse, UPS-IRIT, France

SIAM Conference on Computational Science and Engineering
Salt Lake City, Utah, USA, 03-18-2015

Innovative Computing Lab. at EECS of University of Tennessee, Knoxville

HP Linear Algebra (LA) Packages on emerging computers:

- Linear Algebra:
  - LAPACK/ScaLAPACK: dense LA on shared/distributed system
  - PLASMA/MAGMA: dense LA on manycore/hybrid node (NVIDIA/Intel/AMD)
    $\rightarrow$ sparse LA on distributed system
  - Sparse LA
    - Distributed-memory sparse linear/eigen solvers: (SuperLU_DIST/TRLan/PDSLin)
    - Collaboration to accelerate sparse/application codes (PaStiX, DOD, SciDB, etc.)
- Runtime Systems: QUARK/PULSAR
- Distributed Computing: OpenMPI, ParSEC, DPLASMA, etc.
- Performance Profiling/Modeling: PAPI, etc.
- Bench-marking: HPL, HPCG, etc.
- Auto-tuning: BEAST, etc.

(more info at www.icl.utk.edu)

Can we learn from or contribute to randomized algorithms?

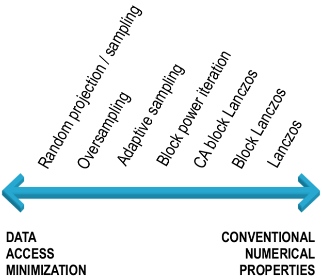
truncated singular value decompositions (SVD)

Compute k-rank approximation of m-by-n sparse matrix A,

$$A \approx U_k \Sigma_k V_k^T \text{ to minimize } \|A - U_k \Sigma_k V_k^T\|_2,$$

where

- U_k and V_k are k left/right singular vectors (i.e., $U^T U = I$ and $V^T V = I$)
- Σ is diagonal with k largest singular values
- it is used for PCA, clustering, ranking, etc.
- many variants with different constraints (i.e., matrix completition)



Outline: Computing truncated SVDs with GPUs

- Performance of random and Lanczos (block, thick-restart, CA)
- Performance of updating SVD
  for Latent Semantic Inedexing and population clustering
- Final Remarks

Subspace projection framework

1. Generate $k + \ell$ orthonormal P and Q approximating ranges of A and A^T,

$$A \approx PQ^T,$$

where ℓ is "oversampling" to improve performance/robustness.

2. Compute SVD of the projected matrix B,

$$B = X\widehat{\Sigma}Y^T,$$

where $B = P^T AQ$.

3. Compute approximation,

$$A \approx \widehat{U}_k \widehat{\Sigma}_k \widehat{V}_k^T,$$

where $\widehat{U}_k = PX_k$ and $\widehat{V}_k = QY_k$.

“Randomization” framework: normalized block power iteration

Input Q: “random” sampling/projection

do

2. SpMM + Ortho

   $\widehat{P} = AQ$, and

   $PR_p = \text{TSQR}(\widehat{P})$

3. Restart (if not done)

   $\widehat{Q} = A^T P$, and

   $QR_q = \text{TSQR}(\widehat{Q})$

while

- iteration to improve approximation when singular values decay slowly.
- “normalized” to maintain stability.
- “randomization” only in starting vectors (e.g., Gaussian random vectors).

"Traditional" algorithm: block Lanczos method

1. Initial + Ortho
   $\widehat{q}_1 = \text{randn}(n, b)$, and $q_1 b_{0,1} = \text{orth}(\widehat{q})$

```
do
```

2. SpMM + Ortho to generate $Q = \mathcal{K}(AA^T, q_1)$ and $P = \mathcal{K}(AA^T, Aq_1)$

   for $j = 1, 2, \ldots, s$ **do**

   $\widehat{p}_j = Aq_j$, and

   $p_j b_{j,j} = \text{orth}([p_{j-1}, \widehat{p}_j])$

   $\widehat{q}_{j+1} = A^T p_j$, and

   $q_{j+1} b_{j,j+1} = \text{orth}([q_j, \widehat{q}_{j+1}])$

   end for

3. Restart (if not done)

   "recycle" a few current approximation

```
while
```

- we use "thick" restart to "recyle" current approximation
  to improve convergence and reduce cost of generating P and Q
- Krylov often converges faster, but with more passes over A
  splitting big SpMM into smaller blocks

s-step Block Lanczos Method

1. Initial + Ortho
   $\widehat{q}_1 = \text{random}(n, b)$ and $q_1 b_{0,1} = \text{orth}(\widehat{q})$

do

2. MPK

   for $j = 1, 2, \ldots, s$ do

   $\widehat{p}_j = A\widehat{q}_j$ then

   $\widehat{q}_{j+1} = A^T \widehat{p}_j$

   end for

3. Ortho

   $QR_q = \text{TSQR}(\widehat{Q})$ and

   $PR_p = \text{TSQR}(\widehat{P})$

4. Restart (if not done)

   $q_1 = q_{c+1}$ (explicit restart)

while

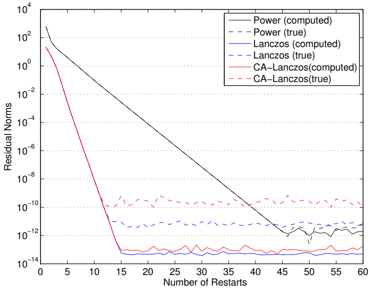
- groups s SpMM/Orthos into one
- "Communication-avoiding" implementation:
  - s block basis vectors with comm cost of one
  - potentially same/less comm than power method
  - overhead to perform comm/comp/store boundary elements

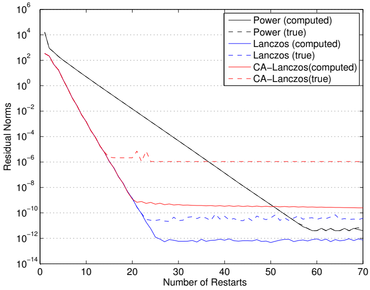
Experimental Setups

Name	Source	m	n	$\frac{nnz}{m}$	σ_1
`BerkStan`	`snap.stanford.edu`	$685,230$	$685,230$	11.1	6.7×10^2
`Netflix`	`netflixprize.com`	$2,649,429$	$17,770$	37.9	1.9×10^4

- One node (two 6-core Intel Xeon) with multiple GPUs (three NDIVIA M2090)
- Compute 50 and 30 largest singular values/vectors for `BerkStan` and `Netflix` (i.e., $n_d = 50$ and 30)
- Projection subspace dimension is $2 \times n_d$
  - Power and explicit-restart Lanczos have same computational cost
- Block size is 10 (i.e., $b = 10$)
- Thick-restart Lanczos recycles $n_d + 2b$ Ritz vectors
  - Lanczos has less computation per restart
  - $s = 2$ for s-step Lanczos
- Orthogonalization schemes: CGS and CholQR with reorthogonalization
- Max. residual norm $\|A\mathbf{u}_i - \sigma_i \mathbf{v}_i\|_2$ for stopping criteria

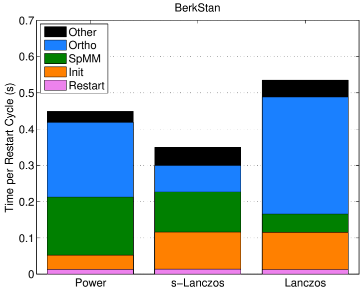
Computed/true residual norms vs. restart

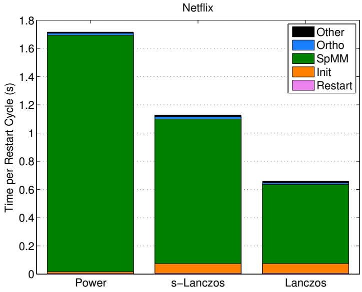




- Lanczos converges faster than Power method (in term of restart count)
- CA-Lanczos' convergence matches with Lanczos (in term of computed residual norm)
- true residual norm diverges from computed one (working to fix this)

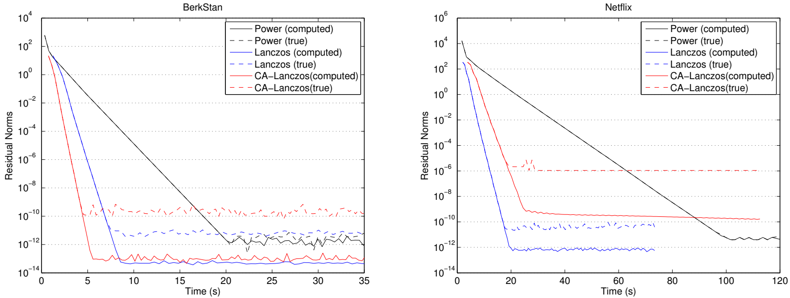
Iteration time breakdown





- SpMM time per Lanczos cycle was shorter due to thick-restarting
- Ortho time per Lanczos cycle was longer due to lower-perf. of dense kernels
- SpMM time increase in *s*-step Lanczos due to overhead of MPK
- each restart cycle (i.e., $O(100)$ SpMMs+Orths) requires < 2 seconds on GPUs

Computed/true residual norms vs. time

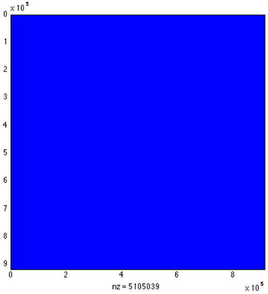


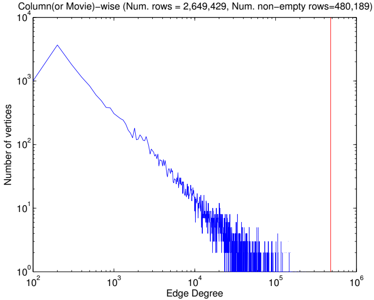
- CA-Lanczos and Lanczos were fastest to converge for `BerkStan` and `Netflix`, respectively (in term of time, if solution requires a few iterations)
- For `Netflix`, Lanczos was competitve even after 1st restart

  - a few smaller SpMMs were as fast as a big SpMM
- CA-Lanczos was slower than Lanczos for `Netflix` due to irregular sparsity

Several un-answered question

- how does it perform at larger-scale?
- how do I measure quality of approximation?
- is there any case where the matrix can be partitioned well?





Outline: Computing truncated SVDs with GPUs

- Performance of randomized with Lanczos (block, thick-restart, CA)
- Performance of sampling to update SVD on a GPU cluster
  for LSI and populartion clustering
- Final Remarks

Adding "document" problem

Given a rank-k approximation of $A \approx U_k \Sigma_k V_k^T$, we compute

$$[A, D] \approx \widehat{U}_k \widehat{\Sigma}_k \widehat{V}_k^T,$$

where D is m-by-d.

- D may be big (e.g., $d = O(10^3)$), but
- is still much smaller than A (i.e., $d \ll m$)
- two other updating problems exist (term-update and weight-correction)

"Fold-in" algorithm by Zha and Simon, 99

1. Orthogonalize D against U_k,

$$\widehat{D} := D - U_k(U_k^T D) \text{ and } \widehat{P}R = \mathrm{TSQR}(\widehat{D}).$$

2. Compute SVD of the projected matrix $B = P^T A Q$, where

$$P = [U_k, \widehat{P}] \text{ and } Q = \begin{pmatrix} V_k & 0 \\ 0 & I \end{pmatrix}$$

Hence,

$$B = \begin{pmatrix} \Sigma_k & U_k^T D \\ & R \end{pmatrix}.$$

3. Compute approximation,

$$A \approx \widehat{U}_k \widehat{\Sigma}_k \widehat{V}_k^T,$$

where $\widehat{U}_k = PX_k$ and $\widehat{V}_k = QY_k$.

- if d is large, infeasibly large memory to store $\widehat{P}$.
- incremental update reduces cost, but still ortho(D) and SVD(B) could be expensive (may lower accuracy, and may be slower).

“Lanczos” algorithm by Vecharynski and Saad, 14

1. Run column-wise Lanczos on $(I - U_k U_k^T)D$ to generate ℓ basis vectors $\widehat{P}_\ell$ and $\widehat{Q}_\ell$
2. Compute SVD of the projected matrix $B = P^T AQ$, where

$$P_{k+\ell} = [U_k, \widehat{P}_\ell] \text{ and } Q_{k+d} = \begin{pmatrix} V_k & 0 \\ 0 & I_d \end{pmatrix}$$

   Hence,

$$B = \begin{pmatrix} \Sigma_k & U_k^T D \\ & \widehat{P}_\ell^T D \end{pmatrix}.$$

3. Compute approximation,

$$A \approx \widehat{U}_k \widehat{\Sigma}_k \widehat{V}_k^T,$$

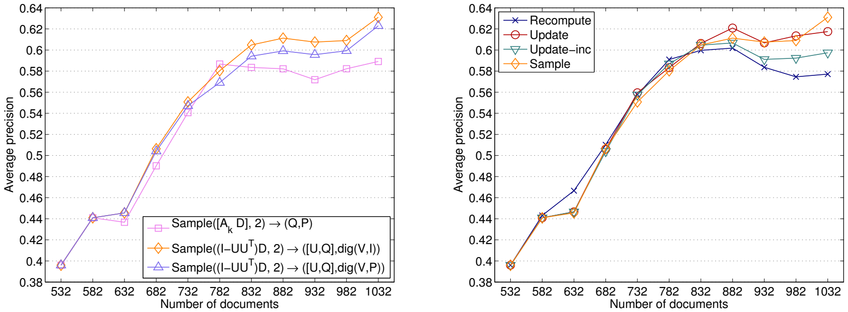
   where $\widehat{U}_k = P_{k+\ell} X_k$ and $\widehat{V}_k = Q_{k+\ell} Y_k$.

Our **“Sampling” algorithms** for updating SVD

To reduce cost of generating P and Q, run block power iteration,

1. on $[U_k \Sigma_k V_k^T, D]$ which generates $P_{k+\ell}$ and $Q_{k+\ell}$
2. on $(I - UU^T)D$ which generates $\widehat{P}_\ell$ and $\widehat{Q}_\ell$,
   and then let $P_{k+\ell} = [U_k, \widehat{P}_\ell]$ and
   2.1 $Q_{k+d} = \begin{pmatrix} V_k & 0 \\ 0 & I_d \end{pmatrix}$ [Vecharynski and Saad, 14],
   or
   2.2 $Q_{k+\ell} = \begin{pmatrix} V_k & 0 \\ 0 & \widehat{Q}_\ell \end{pmatrix}$.

Precision for 5735-by-1033 `MEDLINE matrix` with 30 queries ($s = 50$)



- Sampling performs two iterations (three SpMMs)
- All obtained similar precision.
- CholQR/SVQR for sampling/updating with reorthogonalization

Updating to cluster population by SNP

	JPT+MEX	+ ASW	+ GIH	+CHU
recompute	1.00	1.00	1.00	0.97
no update	1.00	0.81	0.84	0.67
update	1.00	1.00	0.89	0.70
sample	1.00	0.95	0.92	0.86

– average crrelation coefficient of clusters –

- compute rank-5 approximation of JPT and MEX with 116,565 SNP (86 Japanese in Tokyo and 77 Mexican ancestry in LA)
- add ASW, GIH, and CHU (83 African ancestry in SW USA, 88 Gujarati Indian in Houston, and European ancestry in Utah)
- sample with two iterations (three SpMMs).

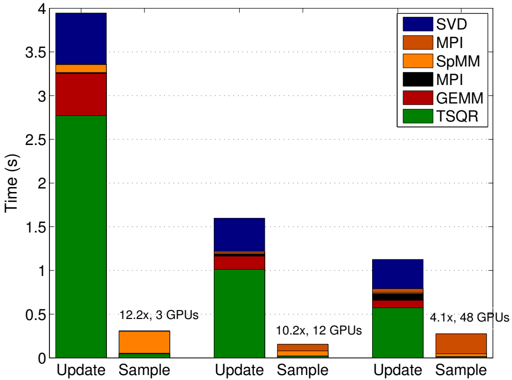
Netflix matrix for performance study

	Incremental update	Sampling
1	The World Is Not Enough	Mission to Mars
2	Mrs. Doubtfire	The World Is Not Enough
3	Mission: Impossible	Armageddon
4	Die Another Day	Crimson Tide
5	The 6th Day	Mission: Impossible
6	Mission to Mars	Die Another Day
7	The Mummy	Entrapment
8	Die Hard 2: Die Harder	Patriot Games
9	Charlie's Angels	Die Hard 2: Die Harder
10	The Santa Clause	Men of Honor

– Query results for "Tomorrow Never Dies" –

- given rank-30 approximation of $5,000$ movies, add $5,000$ more.

Time-breakdown and Parallel scaling



- Sampling is fast (3MPIs, 1GPU/MPI), but
- spends more time in SpMM (i.e., accesses D twice per iteration).

Final Remarks

- Starting effort on linear algebra + randomization package
  - combining linear algebra, randomization, and HPC efforts
  - RBT is integrated in our package for solving dense linear systems

Curent work

- HPC implementation (e.g., matrix partitioning, simple/special of MPK by Knight, Carson, Demmel)
- Other randomization/sampling techniques (e.g., compare/combine with PCA-correlated SNP)
- Larger "sparse" data sets with suggestions on parameter selection (still losts of parameters to tune)

Thank You!!