S9226 Fast singular value decomposition on GPU

Lung-Sheng Chien, NVIDIA
lchien@nvidia.com

Samuel Rodriguez Bernabeu, NVIDIA
srodriguezbe@nvidia.com
Outline

▪ Issues of GESVD
▪ Approximate SVD
▪ Randomized SVD
▪ Conclusions
General Singular Value Decomposition

- The standard form is
  \[ A = USV^T \]

- LAPACK GESVD is most popular routine based on QR iteration

- cuSOLVER library provides two SVD routines
  - GESVD: same algorithm as LAPACK
  - GESVDJ: two-sided Jacobi method

- Tall skinny SVD is a common use case in data analytics
  - singular vectors are required
  - only requires few large singular values
  - typical size: 1.e+6 rows, 100 columns
Strategy of GESVD

- QR factorization: preprocess of tall skinny matrix \((m >> n)\)
  \[ A \rightarrow Q \cdot R \]

- SVD on square matrix \((\text{GEBRD + BDSQR + ORGBR})\)
  \[ R \rightarrow U \cdot S \cdot V^T \]

- GPU does not perform well on tall skinny QR factorization
- GPU does not perform well on QR iteration for small matrix
Review performance on square matrix

- The formula of flops is $2N^3$, same as SGEMM.
- The bigger, the faster.
- The runtime of SGESVD is about 50x of SGEMM.
- Jacobi method (GESVDJ) is faster than QR iteration (GESVD) when matrix size is less than 1024.

<table>
<thead>
<tr>
<th>n</th>
<th>cusolver SGESVD</th>
<th>cusolver SGESVDJ</th>
<th>MKL SGESVD</th>
<th>SGEMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0.04</td>
<td>0.12</td>
<td>0.11</td>
<td>1</td>
</tr>
<tr>
<td>64</td>
<td>0.13</td>
<td>0.45</td>
<td>0.12</td>
<td>7</td>
</tr>
<tr>
<td>128</td>
<td>0.48</td>
<td>1.63</td>
<td>1.16</td>
<td>74</td>
</tr>
<tr>
<td>256</td>
<td>1.31</td>
<td>4.84</td>
<td>2.80</td>
<td>558</td>
</tr>
<tr>
<td>512</td>
<td>5.22</td>
<td>13.56</td>
<td>10.26</td>
<td>2,828</td>
</tr>
<tr>
<td>1024</td>
<td>18.97</td>
<td>27.73</td>
<td>8.33</td>
<td>8,586</td>
</tr>
<tr>
<td>2048</td>
<td>63.15</td>
<td>40.90</td>
<td>19.80</td>
<td>12,514</td>
</tr>
<tr>
<td>4096</td>
<td>152.07</td>
<td>58.08</td>
<td>8.03</td>
<td>13,366</td>
</tr>
<tr>
<td>8192</td>
<td>264.11</td>
<td>49.19</td>
<td>5.52</td>
<td>13,956</td>
</tr>
</tbody>
</table>

CPU: Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz
MKL: compilers_and_libraries_2018.0.128 with 8 cores
GPU: V100
Review performance on tall skinny matrix

- $N = 32$, $M$ varies from 1,000 to $1e+6$

- SGEQRF (QR factorization) is proportional to $M$ because complexity is $2MN^2$ flops

- QR factorization is the bottleneck in SVD when $M$ becomes bigger and bigger (QR ratio from 0.17 to 0.93)

<table>
<thead>
<tr>
<th>$M$</th>
<th>SGEQRF (sec)</th>
<th>SGESVDJ (sec)</th>
<th>QR ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>0.00021</td>
<td>0.00128</td>
<td>0.17</td>
</tr>
<tr>
<td>10,000</td>
<td>0.00058</td>
<td>0.00147</td>
<td>0.40</td>
</tr>
<tr>
<td>100,000</td>
<td>0.00524</td>
<td>0.00654</td>
<td>0.80</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.05897</td>
<td>0.06336</td>
<td>0.93</td>
</tr>
</tbody>
</table>

$N$ is fixed to 32
Weakness of QR factorization

- $M = 8192$, $N$ varies from 32 to 4096
- Complexity of SGEQRF is $2MN^2$ flops, however the runtime is proportional to $N$
- Only trailing matrix uses BLAS-3, it is negligible on tall skinny matrix, the runtime is dominated by panel factorization, which is mainly BLAS-1

<table>
<thead>
<tr>
<th>$N$</th>
<th>SGEQRF (Gflops)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>32.6</td>
</tr>
<tr>
<td>64</td>
<td>66.3</td>
</tr>
<tr>
<td>128</td>
<td>116.6</td>
</tr>
<tr>
<td>256</td>
<td>159.5</td>
</tr>
<tr>
<td>512</td>
<td>338.1</td>
</tr>
<tr>
<td>1024</td>
<td>627.6</td>
</tr>
<tr>
<td>2048</td>
<td>990.8</td>
</tr>
<tr>
<td>4096</td>
<td>2487.6</td>
</tr>
</tbody>
</table>

$M$ is fixed to 8192
Goal and strategy

Goal

- Tall skinny matrix
  - Up to millions of rows
  - Up to hundreds of columns
- Few large singular values
- Left and right singular vectors

Pros and Cons

- QR factorization is not good on tall skinny matrix
- Jacobi method is faster than QR iteration on small matrix
- SGEMM is much faster

\[ A = U S V^T \]

QR factorization

\[ A^T A = V S^2 V^T \]

GEMM + Eig
Technical issues

- Rounding error
  $A^T A$ has rounding error proportional to $\|A\|^2$, so high precision GEMM is used to control rounding error

- Performance
  $A^T A$ is N-by-N, a small matrix compared to tall skinny $A$
  regular GEMM does not perform well
  Need special GEMM to improve the performance

Jacobi method to compute eigen-pair of $(A^T A)$ because it is faster than QR iteration on small matrix
GESVDA: approximate SVD

- \( B = A^T A \) by DGEMM
  DGEMM can reduce rounding errors

- \( (S, V) = \text{eig}(B) \) by DSYEVJ (Jacobi method)
  adjust stopping criteria to improve the performance

- \( U = AVS^{-1} \) by DGEMM and scaling
  left singular vector is not accurate when singular value is small
Quality of solution

- right singular vector is always accurate up to 1.e-6
- Singular values and left singular vectors depend on $M$ and $N$

\[
N \leq 100
\]
\[
M \leq 176,000 \times N
\]

For those numerical singular values bounded below by

\[
S_l(A) \geq 2.65 \times 10^{-3} \|A\|_F
\]

The singular value and singular vectors are accurate up to 1.e-6

Example: Largest singular value $S_l(A) \geq \frac{1}{\sqrt{N}} \|A\|_F$ is accurate up to 1.e-6
Performance of GESVDA

- $N = 32$, $M$ varies from $1,000$ to $1.e+6$
- Runtime of eigenvalue solver is independent of $M$
- Runtime of GESVDA is determined by GEMM
  it is up to $16x$ faster than GESVDJ
- The speedup comes from replacing QR factorization by GEMM

<table>
<thead>
<tr>
<th>$M$</th>
<th>SGEQRF (sec)</th>
<th>SGESVDJ (sec)</th>
<th>SGESVDA (sec)</th>
<th>QR ratio</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>0.00021</td>
<td>0.00128</td>
<td>0.00077</td>
<td>0.17</td>
<td>1.67</td>
</tr>
<tr>
<td>10,000</td>
<td>0.00058</td>
<td>0.00147</td>
<td>0.00078</td>
<td>0.40</td>
<td>1.89</td>
</tr>
<tr>
<td>100,000</td>
<td>0.00524</td>
<td>0.00654</td>
<td>0.00118</td>
<td>0.80</td>
<td>5.55</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.05897</td>
<td>0.06336</td>
<td>0.00376</td>
<td>0.93</td>
<td>16.84</td>
</tr>
</tbody>
</table>

$N$ is fixed to 32
GESVDA Breakdown

- DGEMM is $2MN^2$, DSYEVJ is $O(N^3)$, others are $O(MN)$
- DGEMM is very efficient, only 40% of total time
- The cost of DSYEVJ is independent of $M$, so it decreases as $M$ increases
- “compute U” is slower than “Residual” because it requires ‘double precision’

Ratio of each component in GESVDA

<table>
<thead>
<tr>
<th>$M$</th>
<th>DGEMM (sec)</th>
<th>DSYEVJ (sec)</th>
<th>Compute U (sec)</th>
<th>Residual (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>0.13</td>
<td>0.78</td>
<td>0.03</td>
<td>0.10</td>
</tr>
<tr>
<td>10,000</td>
<td>0.15</td>
<td>0.74</td>
<td>0.03</td>
<td>0.10</td>
</tr>
<tr>
<td>100,000</td>
<td>0.33</td>
<td>0.46</td>
<td>0.13</td>
<td>0.10</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.40</td>
<td>0.16</td>
<td>0.35</td>
<td>0.11</td>
</tr>
</tbody>
</table>

$N$ is fixed to 32
Performance of batched GESVDA

- SGESVDJ is performed by OpenMP
- SGESDVA is performed by multi-stream
- OpenMP can run “batchSize” GEQRF in parallel (GEQRF is 40%+ of GESVD), so speedup of batchSize 32 is not significant

<table>
<thead>
<tr>
<th>M</th>
<th>batchSize</th>
<th>SGESVDJ (sec)</th>
<th>SGESDVA (sec)</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>16,384</td>
<td>1</td>
<td>0.0022</td>
<td>0.0012</td>
<td>1.77</td>
</tr>
<tr>
<td>16,384</td>
<td>32</td>
<td>0.0562</td>
<td>0.0076</td>
<td>7.41</td>
</tr>
<tr>
<td>65,536</td>
<td>1</td>
<td>0.0051</td>
<td>0.0015</td>
<td>3.41</td>
</tr>
<tr>
<td>65,536</td>
<td>32</td>
<td>0.0977</td>
<td>0.0141</td>
<td>6.94</td>
</tr>
<tr>
<td>1,048,576</td>
<td>1</td>
<td>0.0770</td>
<td>0.0062</td>
<td>12.45</td>
</tr>
<tr>
<td>1,048,576</td>
<td>16</td>
<td>0.5329</td>
<td>0.0775</td>
<td>6.87</td>
</tr>
</tbody>
</table>

N is fixed to 35
Batched GESVDA breakdown

- DGEMM is $2MN^2$, DSYEVJ is $O(N^3)$, others are $O(MN)$
- DGEMM is less than 40% of total time
- The cost of DSYEVJ shrinks to 3% because of inhouse batched design—it is no longer kernel launch limited
- “compute U” becomes bottleneck

Ratio of each component in batched GESVDA

<table>
<thead>
<tr>
<th>M</th>
<th>batchSize</th>
<th>DGEMM (sec)</th>
<th>DSYEVJ (sec)</th>
<th>Compute U (sec)</th>
<th>Residual (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16,384</td>
<td>32</td>
<td>0.21</td>
<td>0.50</td>
<td>0.13</td>
<td>0.15</td>
</tr>
<tr>
<td>65,536</td>
<td>32</td>
<td>0.32</td>
<td>0.22</td>
<td>0.29</td>
<td>0.17</td>
</tr>
<tr>
<td>1,048,576</td>
<td>16</td>
<td>0.36</td>
<td>0.03</td>
<td>0.43</td>
<td>0.17</td>
</tr>
</tbody>
</table>

N is fixed to 35
**double-double (fp128) GEMM?**

- Question: low-rank SVD with accuracy up to 1.e-14?
- \( N = 32, \ M \) varies from 1,000 to 1.e+6
- QD package (http://crd-legacy.lbl.gov/~dhbailey/mpdist/)
- LGEMM: \( C \ (dd) \ += A \ (d) \ * B \ (dd) \)
- LGEMM is only useful when \( M > 100,000 \)

<table>
<thead>
<tr>
<th>( M )</th>
<th>DGEQRF (sec)</th>
<th>DGEMM / QR</th>
<th>LGEMM / QR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>0.00023</td>
<td>1.49</td>
<td>0.38</td>
</tr>
<tr>
<td>10,000</td>
<td>0.00077</td>
<td>7.39</td>
<td>0.96</td>
</tr>
<tr>
<td>100,000</td>
<td>0.00822</td>
<td>26.19</td>
<td>1.88</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.08447</td>
<td>14.53</td>
<td>5.34</td>
</tr>
</tbody>
</table>

\( N \) is fixed to 32
Conclusions

- GESVDA replaces SGEQRF by DGEMM to gain the speedup up to 16x
- GESVDA has inhouse batched eigenvalue solver to avoid limitation of kernel launches
- GESVDA can deliver good quality of singular values and singular vectors in common use case
- GESVDA is delivered in cuda 10.1 with batched API
Low-rank approximation of $A$

- Low-rank approximation of a given matrix $A$.
  - Fixed precision.
  - Fixed rank.
- Truncation of SVD gives best rank-$k$ approximation [Eckart-Young-Mirsky]
  - Huge cost. Time complexity is $O(n^3)$
Randomized SVD

- Compute top-k eigenpairs to **sufficient** accuracy.
  - Data analytics, PCA, clustering: $1e^{-2}$ could be enough
  - Physics simulations: $1e^{-2}$ may be useless

- Highlights:
  - Reduced time and space complexity.
  - Preserve sparsity of $A$
  - One-pass or streaming algorithms (touch $A$ once)
Core idea of Randomized LA

- **Data:** $A, k$
- **Result:** $Q, B$ such $A \approx QB$

1. $\Omega =$ SketchingMatrix($A, O(k)$)
2. $C = A\Omega$
3. $Q = \text{orth}(C)$
4. $B = Q^T A$

C is a n-by-$O(k)$ matrix that ensures with high probability:

$$
||A - QQ^H A|| \leq \lambda_{k+1} + \left(1 + 4\sqrt{\frac{2m}{k-1}}\right)^{\frac{1}{2q+1}} \lambda_{k+1}
$$

Time complexity

Data: \( A, k \)
Result: \( Q, B \) such \( A \approx QB \)

1. \( \Omega = \text{SketchingMatrix}( A, \mathcal{O}(k) ) \quad \mathcal{O}(m \times k) \)
2. \( C = A\Omega \quad \mathcal{O}(m \times n \times k) \)
3. \( Q = \text{orth}(C) \quad \mathcal{O}(m \times k^2) \)
4. \( B = Q^T A \quad \mathcal{O}(m \times n \times k) \)
Sketching matrix

- m-by-order(k) matrix which:
  - Captures column space of A
  - Easy to construct
  - Easy to apply.
- Some options:
  - Gaussian projection
  - Subsampled Randomized Hadamard Transform
  - Count sketch
  - Leverage-score subsampling (sparse cases)
SVDR, Randomized SVD

Data: $A$, $k$
Result: $\hat{U}, \Sigma_k, V_k^H$
1 $\Omega = \text{SketchingMatrix}(A, \mathcal{O}(k))$
2 $C = A\Omega$
3 $[Q, R] = qr(C)$
4 $[\hat{U}, \hat{\Sigma}, \hat{V}] = \text{svd}(Q^H A)$
5 $\hat{U} = Q\hat{U}$
6 $A_k = \hat{U}\Sigma_k V_k^H$

- Provided rounding error on SVD:

\[
\|A - \tilde{A}_k\| = \|A - \tilde{Q}\hat{\Sigma}_k \hat{V}^H\| \leq \lambda_{k+1} + \left(1 + 4\sqrt{\frac{2m}{k-1}}\right)^{\frac{1}{2q+1}}\lambda_{k+1}
\]
Numerical experiments. Error metric

- Do not look at

\[ \| U - \tilde{U}_k \| < \epsilon \]
\[ \| V - \tilde{V}_k \| < \epsilon \]

- But instead

\[ \| A - \tilde{A}_k \| = (1 + \eta) \lambda_{k+1} \]
Spectral norm estimator

\[ p_j(A) = \sqrt{\frac{\| (A^H A)^j \tilde{\omega} \|_2}{\| (A^H A)^{j-1} \tilde{\omega} \|_2}} \]

\[ \mathbb{E} \left| p_j(A) \right| \geq \frac{\| A \|_2}{10} > 1 - 4 \sqrt{\frac{n}{j-1}} 100^{-j} \]

- Bottom line: 6 iters estimate norm of A within a factor of 10.

Numerical experiments. Test cases.

- Accuracy depends on spectral gap
- Three test cases:
  - Fast decay
  - S-shape
  - Slow decay
Numerical experiments. Accuracy.
Numerical experiments. Accuracy.

Accuracy for s-shape

\[ 1 - \frac{A_k - \tilde{A}_k}{\lambda_k + 1} \]

- No power method iters.
- 1 power method iters.
Numerical experiments. Accuracy.
### Numerical experiments. Rank-10

#### Speed up for RSVD over GESVD

<table>
<thead>
<tr>
<th>Number of rows of $A$</th>
<th>256</th>
<th>512</th>
<th>768</th>
<th>1024</th>
<th>1280</th>
<th>1536</th>
<th>1792</th>
<th>2048</th>
<th>2304</th>
<th>2560</th>
<th>2816</th>
<th>3072</th>
<th>3328</th>
<th>3584</th>
<th>3840</th>
<th>4096</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>2.5</td>
<td>2.4</td>
<td>2.3</td>
<td>2.6</td>
<td>2.6</td>
<td>2.5</td>
<td>2.6</td>
<td>2.5</td>
<td>2.4</td>
<td>2.3</td>
<td>2.6</td>
<td>2.5</td>
<td>2.4</td>
<td>2.3</td>
<td>2.6</td>
<td>2.5</td>
</tr>
<tr>
<td>512</td>
<td>2.4</td>
<td>5.9</td>
<td>5.6</td>
<td>6.1</td>
<td>6.1</td>
<td>5.7</td>
<td>5.9</td>
<td>6.0</td>
<td>5.9</td>
<td>6.0</td>
<td>5.8</td>
<td>5.9</td>
<td>6.0</td>
<td>5.9</td>
<td>6.0</td>
<td>5.9</td>
</tr>
<tr>
<td>768</td>
<td>2.3</td>
<td>5.6</td>
<td>10.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>2.6</td>
<td>6.1</td>
<td>11.0</td>
<td>24.0</td>
<td>22.0</td>
<td>20.0</td>
<td>22.0</td>
<td>21.0</td>
<td>22.0</td>
<td>21.0</td>
<td>22.0</td>
<td>21.0</td>
<td>22.0</td>
<td>21.0</td>
<td>22.0</td>
<td>21.0</td>
</tr>
<tr>
<td>1280</td>
<td>2.6</td>
<td>6.1</td>
<td>10.0</td>
<td>22.0</td>
<td>35.0</td>
<td>35.0</td>
<td>36.0</td>
<td>34.0</td>
<td>36.0</td>
<td>34.0</td>
<td>36.0</td>
<td>34.0</td>
<td>36.0</td>
<td>34.0</td>
<td>36.0</td>
<td>34.0</td>
</tr>
<tr>
<td>1536</td>
<td>2.5</td>
<td>5.7</td>
<td>11.0</td>
<td>20.0</td>
<td>35.0</td>
<td>53.0</td>
<td>51.0</td>
<td>50.0</td>
<td>51.0</td>
<td>50.0</td>
<td>51.0</td>
<td>50.0</td>
<td>51.0</td>
<td>50.0</td>
<td>51.0</td>
<td>50.0</td>
</tr>
<tr>
<td>1792</td>
<td>2.6</td>
<td>5.9</td>
<td>11.0</td>
<td>22.0</td>
<td>36.0</td>
<td>51.0</td>
<td>77.0</td>
<td>70.0</td>
<td>77.0</td>
<td>70.0</td>
<td>77.0</td>
<td>70.0</td>
<td>77.0</td>
<td>70.0</td>
<td>77.0</td>
<td>70.0</td>
</tr>
<tr>
<td>2048</td>
<td>2.5</td>
<td>6.0</td>
<td>11.0</td>
<td>21.0</td>
<td>34.0</td>
<td>50.0</td>
<td>70.0</td>
<td>103.0</td>
<td>70.0</td>
<td>103.0</td>
<td>70.0</td>
<td>103.0</td>
<td>70.0</td>
<td>103.0</td>
<td>70.0</td>
<td>103.0</td>
</tr>
<tr>
<td>2304</td>
<td>2.5</td>
<td>6.1</td>
<td>10.0</td>
<td>21.0</td>
<td>34.0</td>
<td>51.0</td>
<td>74.0</td>
<td>92.0</td>
<td>74.0</td>
<td>92.0</td>
<td>74.0</td>
<td>92.0</td>
<td>74.0</td>
<td>92.0</td>
<td>74.0</td>
<td>92.0</td>
</tr>
<tr>
<td>2560</td>
<td>2.6</td>
<td>5.6</td>
<td>10.0</td>
<td>21.0</td>
<td>34.0</td>
<td>47.0</td>
<td>70.0</td>
<td>103.0</td>
<td>70.0</td>
<td>103.0</td>
<td>70.0</td>
<td>103.0</td>
<td>70.0</td>
<td>103.0</td>
<td>70.0</td>
<td>103.0</td>
</tr>
<tr>
<td>2816</td>
<td>2.6</td>
<td>6.3</td>
<td>10.0</td>
<td>20.0</td>
<td>35.0</td>
<td>49.0</td>
<td>69.0</td>
<td>98.0</td>
<td>69.0</td>
<td>98.0</td>
<td>69.0</td>
<td>98.0</td>
<td>69.0</td>
<td>98.0</td>
<td>69.0</td>
<td>98.0</td>
</tr>
<tr>
<td>3072</td>
<td>2.6</td>
<td>5.8</td>
<td>11.0</td>
<td>22.0</td>
<td>34.0</td>
<td>47.0</td>
<td>70.0</td>
<td>99.0</td>
<td>70.0</td>
<td>99.0</td>
<td>70.0</td>
<td>99.0</td>
<td>70.0</td>
<td>99.0</td>
<td>70.0</td>
<td>99.0</td>
</tr>
<tr>
<td>3328</td>
<td>2.7</td>
<td>5.9</td>
<td>11.0</td>
<td>22.0</td>
<td>34.0</td>
<td>48.0</td>
<td>64.0</td>
<td>96.0</td>
<td>64.0</td>
<td>96.0</td>
<td>64.0</td>
<td>96.0</td>
<td>64.0</td>
<td>96.0</td>
<td>64.0</td>
<td>96.0</td>
</tr>
<tr>
<td>3584</td>
<td>2.6</td>
<td>5.8</td>
<td>10.0</td>
<td>22.0</td>
<td>33.0</td>
<td>48.0</td>
<td>70.0</td>
<td>94.0</td>
<td>70.0</td>
<td>94.0</td>
<td>70.0</td>
<td>94.0</td>
<td>70.0</td>
<td>94.0</td>
<td>70.0</td>
<td>94.0</td>
</tr>
<tr>
<td>3840</td>
<td>2.6</td>
<td>5.7</td>
<td>10.0</td>
<td>21.0</td>
<td>33.0</td>
<td>46.0</td>
<td>63.0</td>
<td>92.0</td>
<td>63.0</td>
<td>92.0</td>
<td>63.0</td>
<td>92.0</td>
<td>63.0</td>
<td>92.0</td>
<td>63.0</td>
<td>92.0</td>
</tr>
<tr>
<td>4096</td>
<td>2.6</td>
<td>6.4</td>
<td>10.0</td>
<td>21.0</td>
<td>31.0</td>
<td>47.0</td>
<td>65.0</td>
<td>82.0</td>
<td>65.0</td>
<td>82.0</td>
<td>65.0</td>
<td>82.0</td>
<td>65.0</td>
<td>82.0</td>
<td>65.0</td>
<td>82.0</td>
</tr>
</tbody>
</table>
Numerical experiments. Rank-20

Speed up for RSVD over GESVD

<table>
<thead>
<tr>
<th></th>
<th>256</th>
<th>512</th>
<th>768</th>
<th>1024</th>
<th>1280</th>
<th>1536</th>
<th>1792</th>
<th>2048</th>
<th>2304</th>
<th>2560</th>
<th>2816</th>
<th>3072</th>
<th>3328</th>
<th>3584</th>
<th>3840</th>
<th>4096</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.1</td>
<td>1.2</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>512</td>
<td>1.2</td>
<td>3.1</td>
<td>2.8</td>
<td>2.9</td>
<td>2.8</td>
<td>2.6</td>
<td>2.8</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td>768</td>
<td>1.2</td>
<td>2.8</td>
<td>5.5</td>
<td>5.3</td>
<td>5.1</td>
<td>5.2</td>
<td>5.0</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
</tr>
<tr>
<td>1024</td>
<td>1.2</td>
<td>2.9</td>
<td>5.3</td>
<td>10</td>
<td>9.2</td>
<td>9.1</td>
<td>9.6</td>
<td>9.3</td>
<td>9.3</td>
<td>9.3</td>
<td>9.3</td>
<td>9.3</td>
<td>9.3</td>
<td>9.3</td>
<td>9.3</td>
<td>9.3</td>
</tr>
<tr>
<td>1280</td>
<td>1.2</td>
<td>2.8</td>
<td>5.1</td>
<td>9.8</td>
<td>16</td>
<td>15</td>
<td>15</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>1536</td>
<td>1.1</td>
<td>2.6</td>
<td>5.2</td>
<td>9.1</td>
<td>15</td>
<td>24</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>1792</td>
<td>1.2</td>
<td>2.8</td>
<td>5.0</td>
<td>9.6</td>
<td>15</td>
<td>21</td>
<td>34</td>
<td>30</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>2048</td>
<td>1.1</td>
<td>3.3</td>
<td>5.1</td>
<td>9.3</td>
<td>14</td>
<td>21</td>
<td>30</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>2304</td>
<td>1.1</td>
<td>2.7</td>
<td>4.8</td>
<td>9.3</td>
<td>14</td>
<td>21</td>
<td>30</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>2560</td>
<td>1.2</td>
<td>2.6</td>
<td>4.9</td>
<td>9.1</td>
<td>14</td>
<td>19</td>
<td>29</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>2816</td>
<td>1.2</td>
<td>2.6</td>
<td>5.0</td>
<td>8.7</td>
<td>15</td>
<td>20</td>
<td>29</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>3072</td>
<td>1.2</td>
<td>2.7</td>
<td>5.3</td>
<td>9.7</td>
<td>14</td>
<td>19</td>
<td>30</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>3328</td>
<td>1.2</td>
<td>2.6</td>
<td>5.0</td>
<td>9.3</td>
<td>14</td>
<td>20</td>
<td>27</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>3584</td>
<td>1.1</td>
<td>2.7</td>
<td>5.0</td>
<td>9.3</td>
<td>13</td>
<td>20</td>
<td>28</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>3840</td>
<td>1.2</td>
<td>2.7</td>
<td>5.0</td>
<td>9.4</td>
<td>14</td>
<td>18</td>
<td>26</td>
<td>39</td>
<td>39</td>
<td>39</td>
<td>39</td>
<td>39</td>
<td>39</td>
<td>39</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>4096</td>
<td>1.2</td>
<td>2.9</td>
<td>4.7</td>
<td>9.1</td>
<td>13</td>
<td>19</td>
<td>28</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
</tr>
</tbody>
</table>

Number of rows of A
Number of columns of A
Conclusions SVDR

- **Concern**: lack of error estimator for accuracy of spectrum
  - Bounds very pessimistic in practice.
- SVDR provides good accuracy for top-k eigenvalues.
  - Works out of the box for k<10 in practice
- Good alternative PCA, not substitute of GESVD.
  - Can get impressive performance if you know your data
- Internal research project, not included in CUDA 10.1
  - We’d love feedback from you!