High Performance Parallel Tucker Decomposition of Sparse Tensors

Oguz Kaya

INRIA and LIP, ENS Lyon, France

SIAM PP’16, April 14, 2016, Paris, France

Joint work with:

Bora Uçar, CNRS and LIP, ENS Lyon, France
Tucker Tensor Decomposition

- Tucker decomposition
  - provides a rank-\((R_1, \ldots, R_N)\) approximation of a tensor.
  - consists of a core tensor \(G \in \mathbb{R}^{R_1 \times \cdots \times R_N}\) and \(N\) matrices having \(R_1, \ldots, R_N\) columns.

- We are interested in the case when \(X\) is big, sparse, and is of low rank.
- Example: Google web queries, Netflix movie ratings, Amazon product reviews, etc.
Tucker decomposition

- provides a rank-\((R_1, \ldots, R_N)\) approximation of a tensor.
- consists of a core tensor \(G \in \mathbb{R}^{R_1 \times \cdots \times R_N}\) and \(N\) matrices having \(R_1, \ldots, R_N\) columns.

We are interested in the case when \(X\) is **big**, **sparse**, and is of **low rank**.
- Example: Google web queries, Netflix movie ratings, Amazon product reviews, etc.
Related Work:
- Matlab Tensor Toolbox by Kolda et al.
- Efficient and scalable computations with sparse tensors (Baskaran et al., '12)
- Parallel Tensor Compression for Large-Scale Scientific Data (Austin et al., '15)
- Haten2: Billion-scale tensor decompositions (Jung et al., '15)

Applications (in data mining):
- CubeSVD: A Novel Approach to Personalized Web Search (Sun et al., '05)
- Tag Recommendations Based on Tensor Dimensionality Reduction (Symeonidis et al., '08)
- Extended feature combination model for recommendations in location-based mobile services (Sattari et al. '15)

Goal: To compute sparse Tucker decomposition in parallel (shared/distributed memory).
Tucker Tensor Decomposition

- Related Work:
  - Matlab Tensor Toolbox by Kolda et al.
  - Efficient and scalable computations with sparse tensors (Baskaran et al., '12)
  - Parallel Tensor Compression for Large-Scale Scientific Data (Austin et al., '15)
  - Haten2: Billion-scale tensor decompositions (Jung et al., '15)

- Applications (in data mining):
  - CubeSVD: A Novel Approach to Personalized Web Search (Sun et al., '05)
  - Tag Recommendations Based on Tensor Dimensionality Reduction (Symeonidis et al., '08)
  - Extended feature combination model for recommendations in location-based mobile services (Sattari et al. '15)

- Goal: To compute sparse Tucker decomposition in parallel (shared/distributed memory).
Tucker Tensor Decomposition

- Related Work:
  - Matlab Tensor Toolbox by Kolda et al.
  - Efficient and scalable computations with sparse tensors (Baskaran et al., '12)
  - Parallel Tensor Compression for Large-Scale Scientific Data (Austin et al., '15)
  - Haten2: Billion-scale tensor decompositions (Jung et al., '15)

- Applications (in data mining):
  - CubeSVD: A Novel Approach to Personalized Web Search (Sun et al., '05)
  - Tag Recommendations Based on Tensor Dimensionality Reduction (Symeonidis et al., '08)
  - Extended feature combination model for recommendations in location-based mobile services (Sattari et al. '15)

- **Goal:** To compute sparse Tucker decomposition in parallel (shared/distributed memory).
Higher Order Orthogonal Iteration (HOOI) Algorithm

**Algorithm:** HOOI for 3rd order tensors

```
repeat
  1. \( \hat{A} \leftarrow [\mathcal{X} \times_2 B \times_3 C]_{(1)} \)
  2. \( \hat{A} \leftarrow \text{TRSVD}(\hat{A}, R_1) // R_1 \text{ leading left singular vectors} \)
  3. \( \hat{B} \leftarrow [\mathcal{X} \times_1 A \times_3 C]_{(2)} \)
  4. \( B \leftarrow \text{TRSVD}(\hat{B}, R_2) \)
  5. \( \hat{C} \leftarrow [\mathcal{X} \times_1 A \times_2 B]_{(2)} \)
  6. \( C \leftarrow \text{TRSVD}(\hat{C}, R_3) \)
until no more improvement or maximum iterations reached
7. \( \mathcal{G} \leftarrow \mathcal{X} \times_1 A \times_2 B \times_3 C \)
8. return \( \{\mathcal{G}; A, B, C\} \)
```

- We discuss the case where \( R_1 = R_2 = \cdots = R_N = R \) and \( N = 3 \).
- \( A \in \mathbb{R}^{I \times R}, B \in \mathbb{R}^{J \times R}, \) and \( C \in \mathbb{R}^{K \times R} \) are dense.
- \( \hat{A} \leftarrow [\mathcal{X} \times_2 B \times_3 C]_{(1)} \in \mathbb{R}^{I \times R^2} \) is called tensor-times-matrix multiply (TTM).
- \( \hat{A} \in \mathbb{R}^{I \times R^{N-1}}, \hat{B} \in \mathbb{R}^{J \times R^{N-1}}, \) and \( \hat{C} \in \mathbb{R}^{K \times R^{N-1}} \) are dense. (\( R^2 \) columns for \( N = 3 \))
Higher Order Orthogonal Iteration (HOOI) Algorithm

Algorithm: HOOI for 3rd order tensors

repeat
1. \( \hat{A} \leftarrow [\mathcal{X} \times_2 B \times_3 C]_1 \)
2. \( A \leftarrow \text{TRSVD}(\hat{A}, R_1) \) // \( R_1 \) leading left singular vectors
3. \( \hat{B} \leftarrow [\mathcal{X} \times_1 A \times_3 C]_2 \)
4. \( B \leftarrow \text{TRSVD}(\hat{B}, R_2) \)
5. \( \hat{C} \leftarrow [\mathcal{X} \times_1 A \times_2 B]_2 \)
6. \( C \leftarrow \text{TRSVD}(\hat{C}, R_3) \)
until no more improvement or maximum iterations reached
7. \( G \leftarrow \mathcal{X} \times_1 A \times_2 B \times_3 C \)
8. return \( [G; A, B, C] \)

- We discuss the case where \( R_1 = R_2 = \cdots = R_N = R \) and \( N = 3 \).
- \( A \in \mathbb{R}^{I \times R}, B \in \mathbb{R}^{J \times R}, \) and \( C \in \mathbb{R}^{K \times R} \) are dense.
- \( \hat{A} \leftarrow [\mathcal{X} \times_2 B \times_3 C]_1 \in \mathbb{R}^{I \times R^2} \) is called tensor-times-matrix multiply (TTM).
- \( \hat{A} \in \mathbb{R}^{I \times R^{N-1}}, \hat{B} \in \mathbb{R}^{J \times R^{N-1}}, \) and \( \hat{C} \in \mathbb{R}^{K \times R^{N-1}} \) are dense. (\( R^2 \) columns for \( N = 3 \))
Higher Order Orthogonal Iteration (HOOI) Algorithm

\[ \hat{A} \leftarrow [X \times_2 B \times_3 C]_{(1)} \]

\[ A \leftarrow \text{TRSV}(\hat{A}, R_1) \quad // R_1 \text{ leading left singular vectors} \]

\[ \hat{B} \leftarrow [X \times_1 A \times_3 C]_{(2)} \]

\[ B \leftarrow \text{TRSV}(\hat{B}, R_2) \]

\[ \hat{C} \leftarrow [X \times_1 A \times_2 B]_{(2)} \]

\[ C \leftarrow \text{TRSV}(\hat{C}, R_3) \]

until no more improvement or maximum iterations reached

\[ G \leftarrow X \times_1 A \times_2 B \times_3 C \]

return \([G; A, B, C]\)

We discuss the case where \( R_1 = R_2 = \cdots = R_N = R \) and \( N = 3 \).

\[ A \in \mathbb{R}^{I \times R}, B \in \mathbb{R}^{J \times R}, \text{ and } C \in \mathbb{R}^{K \times R} \text{ are dense.} \]

\[ \hat{A} \leftarrow [X \times_2 B \times_3 C]_{(1)} \in \mathbb{R}^{I \times R^2} \text{ is called tensor-times-matrix multiply (TTM).} \]

\[ \hat{A} \in \mathbb{R}^{I \times R^{N-1}}, \hat{B} \in \mathbb{R}^{J \times R^{N-1}}, \text{ and } \hat{C} \in \mathbb{R}^{K \times R^{N-1}} \text{ are dense. (}R^2\text{ columns for } N = 3) \]
Higher Order Orthogonal Iteration (HOOI) Algorithm

**Algorithm:** HOOI for 3rd order tensors

repeat
1. \( \hat{A} \leftarrow [\mathbf{X} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)} \)  
2. \( \mathbf{A} \leftarrow \text{TRSVD}(\hat{\mathbf{A}}, R_1) \) // \( R_1 \) leading left singular vectors  
3. \( \hat{\mathbf{B}} \leftarrow [\mathbf{X} \times_1 \mathbf{A} \times_3 \mathbf{C}]_{(2)} \)  
4. \( \mathbf{B} \leftarrow \text{TRSVD}(\hat{\mathbf{B}}, R_2) \)  
5. \( \hat{\mathbf{C}} \leftarrow [\mathbf{X} \times_1 \mathbf{A} \times_2 \mathbf{B}]_{(2)} \)  
6. \( \mathbf{C} \leftarrow \text{TRSVD}(\hat{\mathbf{C}}, R_3) \)  
until no more improvement or maximum iterations reached
7. \( \mathbf{G} \leftarrow [\mathbf{X} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}] \)  
8. return \([\mathbf{G}; \mathbf{A}, \mathbf{B}, \mathbf{C}]\)

- We discuss the case where \( R_1 = R_2 = \cdots = R_N = R \) and \( N = 3 \).
- \( \mathbf{A} \in \mathbb{R}^{I \times R}, \mathbf{B} \in \mathbb{R}^{J \times R}, \) and \( \mathbf{C} \in \mathbb{R}^{K \times R} \) are dense.
- \( \hat{\mathbf{A}} \leftarrow [\mathbf{X} \times_2 \mathbf{B} \times_3 \mathbf{C}]_{(1)} \in \mathbb{R}^{I \times R^2} \) is called tensor-times-matrix multiply (TTM).
- \( \hat{\mathbf{A}} \in \mathbb{R}^{I \times R^{N-1}}, \hat{\mathbf{B}} \in \mathbb{R}^{J \times R^{N-1}}, \) and \( \hat{\mathbf{C}} \in \mathbb{R}^{K \times R^{N-1}} \) are dense. (\( R^2 \) columns for \( N = 3 \))
Higher Order Orthogonal Iteration (HOOI) Algorithm

### Algorithm: HOOI for 3rd order tensors

repeat
1. \( \hat{A} \leftarrow \mathcal{X} \times_2 B \times_3 C \) (1)
2. \( A \leftarrow \text{TRSVD}(\hat{A}, R_1) \) (// \( R_1 \) leading left singular vectors)
3. \( \hat{B} \leftarrow \mathcal{X} \times_1 A \times_3 C \) (2)
4. \( B \leftarrow \text{TRSVD}(\hat{B}, R_2) \)
5. \( \hat{C} \leftarrow \mathcal{X} \times_1 A \times_2 B \) (2)
6. \( C \leftarrow \text{TRSVD}(\hat{C}, R_3) \)
until no more improvement or maximum iterations reached
7. \( \mathcal{G} \leftarrow \mathcal{X} \times_1 A \times_2 B \times_3 C \)
8. return \([\mathcal{G}; A, B, C]\)

- We discuss the case where \( R_1 = R_2 = \cdots = R_N = R \) and \( N = 3 \).
- \( A \in \mathbb{R}^{I \times R}, B \in \mathbb{R}^{J \times R}, \) and \( C \in \mathbb{R}^{K \times R} \) are dense.
- \( \hat{A} \leftarrow \mathcal{X} \times_2 B \times_3 C \) (1) \( \in \mathbb{R}^{I \times R^2} \) is called tensor-times-matrix multiply (TTM).
- \( \hat{A} \in \mathbb{R}^{I \times R^{N-1}}, \hat{B} \in \mathbb{R}^{J \times R^{N-1}}, \) and \( \hat{C} \in \mathbb{R}^{K \times R^{N-1}} \) are dense. (\( R^2 \) columns for \( N = 3 \))
Tensor-Times-Matrix Multiply

\[
\hat{A} \leftarrow [X \times_2 B \times_3 C]_{(1)}, \hat{A} \in \mathbb{R}^{l \times R^2}
\]

\[B(j,:) \otimes C(k,:) \in \mathbb{R}^{R^2}\text{ is a Kronecker product.}\]

- For each nonzero \(x_{i,j,k}\);
  - \(\hat{A}(i,:)\) receives the update \(x_{i,j,k}[B(j,:) \otimes C(k,:)].\)

---

**Algorithm:**

\[\hat{A} \leftarrow [X \times_2 B \times_3 C]_{(1)}\]

1. \(\hat{A} \leftarrow \text{zeros}(l, R^2)\)
2. \(\text{foreach } x_{i,j,k} \in \mathcal{X} \text{ do}\)
3. \(\hat{A}(i,:) \leftarrow \hat{A}(i,:) + x_{i,j,k}[B(j,:) \otimes C(k,:)]\)
**Algorithm:** Computing $\mathbf{A}$ in fine-grain HOOI at process $p$

foreach $x_{i,j,k} \in \mathcal{X}_p$ do

1. $\hat{\mathbf{A}}(i,:)$ ← $\hat{\mathbf{A}}(i,:)$ + $x_{i,j,k}[\mathbf{B}(j,:) \otimes \mathbf{C}(k,:)]$

2. Send/Receive and sum up “partial” rows of $\hat{\mathbf{A}}$

3. $\mathbf{A}(I_p,:)$ ← TRSVD($\hat{\mathbf{A}}, R$)

4. Send/Receive rows of $\mathbf{A}$
Number of rows sent/received in fold/expand are equal.
  - Each communication unit of expand has size $R$.
  - Each communication unit of fold has size $R^{N-1}$.

We want to avoid assembling $\hat{A}$ in fold communication.
We need to compute $\text{TRSVD}(\hat{A}, R)$.

**Algorithm:** Computing $A$ in fine-grain HOOI at process $p$

1. $\hat{A}(i,:) \leftarrow \hat{A}(i,:) + x_{i,j,k}[B(j,:) \otimes C(k,:)]$
2. Send/Receive and sum up “partial” rows of $\hat{A}$
3. $A(I_p,:) \leftarrow \text{TRSVD}(\hat{A}, R)$
4. Send/Receive rows of $A$
(Bad) Fine-Grain Parallel TTM within Tucker-ALS

- Number of rows sent/received in fold/expand are equal.
  - Each communication unit of expand has size $R$.
  - Each communication unit of fold has size $R^{N-1}$.
- We want to avoid assembling $\hat{A}$ in fold communication.
- We need to compute $\text{TRSVD}(\hat{A}, R)$.

**Algorithm:** Computing $A$ in fine-grain HOOI at process $p$

```plaintext
1. $\hat{A}(i,:)$ ← $\hat{A}(i,:)$ + $x_{i,j,k}[B(j,:) \otimes C(k,:)]$
2. Send/Receive and sum up “partial” rows of $\hat{A}$
3. $A(I_p,:)$ ← $\text{TRSVD}(\hat{A}, R)$
4. Send/Receive rows of $A$
```
Computing TRSVD

Gram matrix $\tilde{A}\tilde{A}^T$?

Iterative solvers?
  - Need to perform $\tilde{A}x$ and $\tilde{A}^T x$ efficiently.
Computing TRSVD

Gram matrix $\tilde{A}\tilde{A}^T$?

Iterative solvers?

Need to perform $\tilde{A}x$ and $\tilde{A}^T x$ efficiently.
Computing TRSVD

\[ \tilde{A} = \tilde{A}_1 + \tilde{A}_2 + \tilde{A}_3 \]

- Gram matrix \( \tilde{A}\tilde{A}^T \)?
- Iterative solvers?
  - Need to perform \( \tilde{A}x \) and \( \tilde{A}^Tx \) efficiently.
Computing $y \leftarrow \tilde{A}x$
Computing $y \leftarrow \tilde{A}x$
Computing $y \leftarrow \tilde{A}x$

- For each unit of communication, we perform extra work in MxV.
Computing $y \leftarrow \hat{A}x$

Instead of communicating $R^{N-1}$ entries, we communicate 1! (per SVD iteration)
Computing $y \leftarrow \hat{A} x$

$y \leftarrow \hat{A}^T x$ works in reverse with the same communication cost.

Row distribution of $y$ and left-singular vectors are the same as $\hat{A}$
- $A$ gets the same row distribution as $\hat{A}$. 
(Good) Fine-Grain Parallel TTM within Tucker-ALS

Algorithm: Computing $A$ in fine-grain HOOI at process $p$

1. $\hat{A}(i,:) \leftarrow \hat{A}(i,:) + x_{i,j,k}[B(j,:) \otimes C(k,:)]$
2. $A(l_p,:) \leftarrow \text{TRSV}(\hat{A}, R, MxV(\ldots), MTxV(\ldots))$
3. Send/Receive rows of $A$
Hypergraph Model for Parallel HOOI

- Multi-constraint hypergraph partitioning
  - We balance computation and memory costs.
- By minimizing the cutsize of the hypergraph, we minimize:
  - the total communication volume of $Mtv/MTxV$,
  - the total extra $MxV/MTxV$ work,
  - and the total volume of communication for $TTM$.
- Ideally, should minimize the **maximum**, not **total**

$\mathcal{X} = \{(1, 2, 3), (2, 3, 1), (3, 1, 2)\}$
1 Introduction
2 Parallel HOOI
3 Results
4 Conclusion
Experimental Setup

- HyperTensor
  - Hybrid OpenMP/MPI code in C++
  - Dependencies to BLAS, LAPACK, and C++11 STL
  - SLEPc/PETSc for distributed memory TRSVD computations

- IBM BlueGene/Q Machine
  - 16GB memory and 16 cores (at 1.6GHz) per node
  - Experiments using up to 4096 cores (256 nodes)

- $R_i$ is set to 5/10 for 4/3-dimensional tensors.

Tensor sizes

<table>
<thead>
<tr>
<th>Tensor</th>
<th>$l_1$</th>
<th>$l_2$</th>
<th>$l_3$</th>
<th>$l_4$</th>
<th>#nonzeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflix</td>
<td>480K</td>
<td>17K</td>
<td>2K</td>
<td>-</td>
<td>100M</td>
</tr>
<tr>
<td>NELL</td>
<td>3.2M</td>
<td>301</td>
<td>638K</td>
<td>-</td>
<td>78M</td>
</tr>
<tr>
<td>Delicious</td>
<td>1K</td>
<td>530K</td>
<td>17M</td>
<td>2.4M</td>
<td>140M</td>
</tr>
<tr>
<td>Flickr</td>
<td>713</td>
<td>319K</td>
<td>28M</td>
<td>1.6M</td>
<td>112M</td>
</tr>
</tbody>
</table>
Results - Flickr/Delicious

Per iteration runtime of the parallel HOOI (in seconds)

| #nodes×#cores | Delicious | | | | | Flickr | | | | |
| | fine-hp | fine-rd | coarse-hp | coarse-bl | fine-hp | fine-rd | coarse-hp | coarse-bl |
| 1 × 16 | - | - | - | - | - | - | - | - |
| 2 × 16 | - | - | - | - | - | - | - | - |
| 4 × 16 | - | - | - | - | - | - | - | - |
| 8 × 16 | 164.9 | - | 235.3 | 400.5 | 206.2 | - | 287.5 | 308.5 |
| 16 × 16 | 85.2 | 162.0 | 197.5 | 302.4 | 115.6 | 221.8 | 210.5 | 230.1 |
| 32 × 16 | 47.6 | 96.2 | 155.6 | 206.5 | 64.6 | 124.5 | 166.3 | 190.1 |
| 64 × 16 | 27.2 | 57.8 | 98.9 | 159.6 | 36.8 | 69.9 | 124.1 | 129.0 |
| 128 × 16 | 18.2 | 34.7 | 80.8 | 96.4 | 22.6 | 42.9 | 87.9 | 102.3 |
| 256 × 16 | 12.2 | 22.1 | 65.1 | 77.1 | 20.0 | 29.2 | 73.8 | 86.3 |

- Coarse-grain kernel is slow due to load imbalance and communication.
- On Delicious, fine-hp is 1.8x/5.4x/6.4x faster than fine-rd/coarse-hp/coarse-bl.
- On Flickr, fine-hp is 1.5x/3.7x/4.3x faster than fine-rd/coarse-hp/coarse-bl.
- All instances achieve scalability to 4096 cores.
## Results - NELL/Netflix

Per iteration runtime of the parallel HOOI (in seconds)

<table>
<thead>
<tr>
<th>#nodes×#cores</th>
<th>NELL</th>
<th></th>
<th></th>
<th></th>
<th>Netflix</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fine-hp</td>
<td>fine-rd</td>
<td>coarse-hp</td>
<td>coarse-bl</td>
<td>fine-hp</td>
<td>fine-rd</td>
<td>coarse-hp</td>
<td>coarse-bl</td>
</tr>
<tr>
<td>1×16</td>
<td>222.1</td>
<td>222.1</td>
<td>240.1</td>
<td>240.1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2×16</td>
<td>151.6</td>
<td>137.6</td>
<td>198.5</td>
<td>164.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4×16</td>
<td>87.7</td>
<td>75.9</td>
<td>180.6</td>
<td>131.4</td>
<td>33.7</td>
<td>39.2</td>
<td>46.0</td>
<td>42.8</td>
</tr>
<tr>
<td>8×16</td>
<td>67.8</td>
<td>46.9</td>
<td>172.5</td>
<td>109.7</td>
<td>18.6</td>
<td>26.1</td>
<td>30.6</td>
<td>33.4</td>
</tr>
<tr>
<td>16×16</td>
<td>54.9</td>
<td>28.3</td>
<td>112.4</td>
<td>94.1</td>
<td>10.3</td>
<td>18.3</td>
<td>32.2</td>
<td>27.8</td>
</tr>
<tr>
<td>32×16</td>
<td>43.9</td>
<td>17.2</td>
<td>73.8</td>
<td>68.2</td>
<td>5.7</td>
<td>13.9</td>
<td>26.2</td>
<td>26.7</td>
</tr>
<tr>
<td>64×16</td>
<td>35.4</td>
<td>11.9</td>
<td>67.1</td>
<td>54.5</td>
<td>3.9</td>
<td>10.9</td>
<td>26.2</td>
<td>21.7</td>
</tr>
<tr>
<td>128×16</td>
<td>26.7</td>
<td>8.4</td>
<td>50.3</td>
<td>48.5</td>
<td>2.9</td>
<td>8.7</td>
<td>19.8</td>
<td>18.7</td>
</tr>
<tr>
<td>256×16</td>
<td>14.8</td>
<td>7.7</td>
<td>48.1</td>
<td>44.9</td>
<td>3.8</td>
<td>8.3</td>
<td>14.7</td>
<td>16.1</td>
</tr>
</tbody>
</table>

- On Netflix, fine-hp is **2.8x/5x/5.5x** faster than fine-rd/coarse-hp/coarse-bl.
- On NELL, fine-rd is faster than fine-hp (**5x less** total comm. but **2x more** max comm.)
Conclusion

- We provide
  - the first high performance shared/distributed memory parallel algorithm/implementation for the Tucker decomposition of sparse tensors
  - hypergraph partitioning models of these computations for better scalability.
- We achieve scalability up to 4096 cores even with random partitioning.
- We enable Tucker-based tensor analysis of very big sparse data.
We provide the first high performance shared/distributed memory parallel algorithm/implementation for the Tucker decomposition of sparse tensors and hypergraph partitioning models of these computations for better scalability.

We achieve scalability up to 4096 cores even with random partitioning.

We enable Tucker-based tensor analysis of very big sparse data.
We provide

- the first high performance shared/distributed memory parallel algorithm/implementation for the **Tucker decomposition of sparse tensors**
- **hypergraph partitioning models** of these computations for better scalability.

We achieve scalability **up to 4096 cores** even with random partitioning.

We enable Tucker-based tensor analysis of very big sparse data.
References


Contact

oguz.kaya@ens-lyon.fr
www.oguzkaya.com

bora.ucar@ens-lyon.fr
perso.ens-lyon.fr/bora.ucar