Towards parallel bipartite matching algorithms

Bora Uçar

CNRS and GRAAL, ENS Lyon, France

Scheduling for large-scale systems, 13–15 May 2009, Knoxville

Joint work with
Patrick R. Amestoy (ENSEEIHT-IRIT, Toulouse)
Iain S. Duff (CERFACS, Toulouse and RAL, Oxon)
Daniel Ruiz (ENSEEIHT-IRIT)

Supported by Agence Nationale de la Recherche through SOLSTICE project number ANR-06-CIS6-010.
1. Matchings

2. Matrix scaling

3. Matchings (cont’)
   - Achieving an exact solution
   - Obtaining a sub-optimal solution

4. Concluding remarks
Matching

Definitions

Given an $n \times n$ matrix $A$, find a permutation $M$ such that the diagonal product of the permuted matrix, $\prod \text{diag}(AM)$, is maximum (in magnitude) among all permutations. Assume $a_{ij} \geq 0$ and there is at least one nonzero product diagonal (full structural rank).

Alternatively, select $n$ entries from a given matrix such that no two are in a common row and column, and their product is maximum. Also called transversal and bipartite matching.

Motivations


Combined with scaling can avoid many numerical difficulties in factorization and linear system solution [Duff and Pralet, SIAM SIMAX(2005)].
**Current state-of-the-art**

### Sequential

- Polynomial time solvable; best known polynomial algorithm $O(n(\tau + n \log n))$, where $\tau = \text{nnz}(A)$ [Fredman and Tarjan, J. ACM (1987)],
- HSL subroutine MC64 [Duff and Koster, SIAM SIMAX(1999)] provides algorithms for a family of bipartite matching problems,
- MC64 has a higher polynomial (worst case) time complexity; but behaves faster than that bound.

### Parallel

- Standard algorithms use depth-first/breadth-first search; inherently sequential,
- Some newer efforts [Riedy and Demmel, PP04]; some moderate speed-ups (around 5 across 5–30 processors); slow downs too.
- $\frac{1}{2}$-approximation algorithm: [Manne and Bisseling, PPAM 2007]–Scales well up to 32 processors; [Halappanavar and Pothen]–CSCAPES Seminar, 2008.
Invariance

If \( Q \) and \( R \) are two matchings and

\[
\prod \text{diag}(AQ) > \prod \text{diag}(AR)
\]

then

\[
\prod \text{diag}(\hat{A}Q) > \prod \text{diag}(\hat{A}R)
\]

for \( \hat{A} = D_1AD_2 \) with \( D_1 \) and \( D_2 \) being diagonal matrices.

Invariance under scaling

- \( Q \) is optimal for \( A \) iff it is optimal for \( \hat{A} \).
- In other words, the matching that gives the maximum diagonal product is invariant under row/column scaling. Also discussed in [Olschowka and Neumaier, Linear Algebra Appl., (1996)].
Suppose we have obtained a scaled matrix $\hat{A} = D_1 AD_2$ such that

1. $\hat{a}_{ij} \leq 1.0$,
2. all rows and columns has at least one entry equal to 1.0.

**Observation**

Any perfect matching $Q$ with $\text{diag}(\hat{A}Q)$ consisting only entries of magnitude 1.0 is optimal.

**Algorithm starts to shape up...**

1. $\hat{A} \leftarrow \text{scale}(A)$
2. $\hat{A}_f \leftarrow \text{filter}(\hat{A} = 1)$
3. if there exist a perfect matching in $\hat{A}_f$ then
4. return the matching
5. else
6. ...
Matrix scaling

Definition

Given an $m \times n$ sparse matrix $A$, find diagonal matrices $D_1 > O$ and $D_2 > O$ such that all rows and columns of the scaled matrix

$$\hat{A} = D_1 AD_2$$

have equal norm.
The sequential algorithm \textbf{[Ruiz 2001]}

1. \( D_1^{(0)} \leftarrow I_{m \times m} \quad D_2^{(0)} \leftarrow I_{n \times n} \)
2. \textbf{for} \( k = 1, 2, \ldots \) \textbf{until} convergence \textbf{do}
3. \( D_R \leftarrow \text{diag} \left( \sqrt{\| r_i^{(k)} \|_\ell} \right) \quad i = 1, \ldots, m \)
4. \( D_C \leftarrow \text{diag} \left( \sqrt{\| c_j^{(k)} \|_\ell} \right) \quad j = 1, \ldots, n \)
5. \( D_1^{(k+1)} \leftarrow D_1^{(k)} D_R^{-1} \)
6. \( D_2^{(k+1)} \leftarrow D_2^{(k)} D_C^{-1} \)
7. \( A^{(k+1)} \leftarrow D_1^{(k+1)} A D_2^{(k+1)} \)

\begin{align*}
\text{Reminder} \\
\| x \|_\infty &= \max \{ |x_i| \} \\
\| x \|_1 &= \sum |x_i|
\end{align*}

Notes

\( \ell \): any vector norm (usually \( \infty \)- and 1-norms)
Convergence is achieved when

\[
\max_{1 \leq i \leq m} \left\{ |1 - \| r_i^{(k)} \|_\ell| \right\} \leq \varepsilon \quad \text{and} \quad \max_{1 \leq j \leq n} \left\{ |1 - \| c_j^{(k)} \|_\ell| \right\} \leq \varepsilon
\]
Features

Some properties

- Preserves symmetry; permutation independent; amenable to parallelization [Amestoy, Duff, Ruiz, and U. (proc. VecPar'08)].
- In $\infty$-norm, linear convergence with asymptotic rate of $1/2$,
- Scaling in $\infty$-norm is not unique,
- With 1-norm, results are similar to those of the other well-known algorithms [Sinkhorn and Knopp, Pacific J. Math (1967)]; convergence under certain conditions.
  - If each entry lie in a perfect matching, there is a unique scaled matrix,
  - If there exists a perfect matching but not all entries can be made to be in a perfect matching, iteration converges; those kind of entries must tend to zero.
## Summary of computational and communication requirements

### Computations (sequential execution) per iteration

<table>
<thead>
<tr>
<th>Op.</th>
<th>SpMxV</th>
<th>1-norm</th>
<th>∞-norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>(\text{nnz}(A))</td>
<td>(2 \times \text{nnz}(A))</td>
<td>(0)</td>
</tr>
<tr>
<td>mult</td>
<td>(\text{nnz}(A))</td>
<td>(2 \times \text{nnz}(A) + m + n)</td>
<td>(2 \times \text{nnz}(A) + m + n)</td>
</tr>
<tr>
<td>comparison</td>
<td>0</td>
<td>0</td>
<td>(2 \times \text{nnz}(A))</td>
</tr>
</tbody>
</table>

### Communication

The communication operations both in the 1-norm and ∞-norm algorithms are the same as those in the computations

\[
y \leftarrow Ax \\
x \leftarrow A^T y
\]

when the partitions on \(x\) and \(y\) are equal to the partitions on \(D_2\) and \(D_1\).
### Parallelization results: Speed-up values

<table>
<thead>
<tr>
<th>matrix</th>
<th>Seq. Time</th>
<th>Number of processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>olesnik</td>
<td>46.08</td>
<td>1.9</td>
</tr>
<tr>
<td>c-71</td>
<td>51.60</td>
<td>1.8</td>
</tr>
<tr>
<td>boyd1</td>
<td>70.34</td>
<td>1.9</td>
</tr>
<tr>
<td>twotone</td>
<td>74.76</td>
<td>1.9</td>
</tr>
<tr>
<td>lhr71</td>
<td>78.25</td>
<td>2.0</td>
</tr>
<tr>
<td>aug3dcqp</td>
<td>8.30</td>
<td>1.7</td>
</tr>
<tr>
<td>a5esindl</td>
<td>15.09</td>
<td>1.8</td>
</tr>
<tr>
<td>a2nnsnsl</td>
<td>20.71</td>
<td>1.8</td>
</tr>
<tr>
<td>a0nsdsil</td>
<td>20.92</td>
<td>1.8</td>
</tr>
<tr>
<td>blockqp1</td>
<td>32.55</td>
<td>1.9</td>
</tr>
</tbody>
</table>

- Averages of 10 different partitions obtained using PaToH [Çatalyürek and Aykanat, Tech.Rep (1999)],
- PC cluster with a Gigabit Ethernet switch. 16 nodes, each having Intel Pentium IV 2.6 GHz processor, 1GB RAM,
Algorithm

In the scaled matrix $\hat{a}_{ij} \leq 1.0$.

Algorithm: scaling (with $\varepsilon$ tolerance) is efficiently performed

1: $\hat{A} \leftarrow \text{scale}(A)$
2: $\hat{A}_f \leftarrow \text{filter}(1.0 - \varepsilon \leq \hat{A} \leq 1.0 + \varepsilon)$
3: if there exist a perfect matching in $\hat{A}_f$ then
4: return the matching
5: else
6: ...

What remains to be done?

- Step 3 can be performed sequentially, if there is only a little number of nonzeros in the filtered matrix $\hat{A}_f$.
- the "else" part can be addressed in two ways:
  - Solve the problem exactly,
  - Or, find a sub-optimal solution (quickly).
Solving the “else” part exactly

New entries scaled to $\leq 1.0$

Bring on new entries to the filtered matrix $\hat{A}_f$ by updating the scaling factors so that we have perfect matching at the end.

Dulmage-Mendelsohn decomposition

(from [Pothen and Fan, ACM TOMS (1990)])

<table>
<thead>
<tr>
<th></th>
<th>$H_C$</th>
<th>$S_C$</th>
<th>$V_C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_R$</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>$S_R$</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>$V_R$</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

- Unique Horizontal, Square, and Vertical blocks (defined by any maximum cardinality matching)
- $H_R$s are perfectly matched to $H_C$s,
- $S_R$s are perfectly matched to $S_C$s,
- $V_C$s are perfectly matched to $V_R$s.
Dulmage-Mendelsohn decomposition

The entries in $(H_R, S_C)$, $(H_R, V_C)$, and $(S_R, V_C)$ cannot be in a maximum matching.

All maximum matchings contain entries from the three diagonal blocks.
Implications for us

- The filtered matrix $\hat{A}_f$ is in this form,
- Matrix $A$ (hence $\hat{A}$) must have nonzeros in the blocks shown with 0,
- Find the maximum scaled entries ($< 1.0$) from each of those blocks,
- With a rule update the scaling matrices,
  - keep the 1s in the diagonal blocks,
  - keep everything else $\leq 1$,
  - have an entry from the 0 block scaled to 1.
Implications for us

Define the three max entries

\[
\begin{align*}
    m_1 &= \sqrt{\max \hat{A}(V_R, H_C)} \\
    m_2 &= \max \hat{A}(S_R, H_C) \\
    m_3 &= \max \hat{A}(V_R, S_C)
\end{align*}
\]

Set the updates

If \( m_1 \) is largest then

\[
\alpha = \frac{1}{m_1}
\]

ewse if \( m_2 \) is the largest then \( \alpha = \frac{1}{m_2} \)

\[\vdots\]

\[
\begin{array}{c|c|c|c}
    \alpha & 1 & \frac{1}{\alpha} \\
    \hline
    H_C & =1 & =1 & =1 \\
    S_C & <1 & =1 & =1 \\
    V_C & <1 & <1 & =1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
    =1 & <1 & <1 \\
    \hline
    <=1 & =1 & <1 \\
    =1 & <=1 & =1 \\
\end{array}
\]
Algorithm exposed

**Algorithm**

1: $\hat{A}_1 \leftarrow \text{scale-1-norm}(A)$
2: $\hat{A} \leftarrow \text{scale-}\infty\text{-norm}(\hat{A}_1)$
3: $\hat{A}_f \leftarrow \text{filter}(1.0 - \varepsilon \leq \hat{A} \leq 1.0 + \varepsilon)$
4: if there exist a perfect matching in $\hat{A}_f$ then
5: return the matching
6: else
7: Compute the dmperm of $\hat{A}_f$
8: for $k = 1, 2, \ldots$ do
9: Scale a particular entry in $\hat{A}$ to $1.0 \pm \varepsilon$
10: Update dm-structure and scaling matrices
11: if perfect matching exists then
12: return the matching

**Reminder**

- In 1-norm scaling, any entry not in a perfect matching tends to zero,
- 1-norm scaling is unique; $\infty$-norm is not,
Experiments (Looking for an exact solution)

- Matrices from University of Florida sparse matrix collection, satisfying the following properties
  - Square, with $1000 \leq n < nnz \leq 2.0e+6$,
  - total support (no nonzeros in off diagonal blocks of the dmperm),
  - no explicit zeros, real, not $\{0, 1, -1\}^{n \times n}$.

A total of 276 matrices. 8 required special attention; excluding those 268. 192 are symmetric and 76 are unsymmetric.

Fast solutions

In 180 matrices, no iterations after the initial 1-norm (at most 40 iterations) and $\infty$-norm (at most 20 iterations) scaling steps with $\varepsilon = 1.0e-3$ (126/192 symmetric; 54/76 unsymmetric).
Experiments (Looking for an exact solution)—Cont’

Algorithm: first few steps

1: $\hat{A}_1 \leftarrow \text{scale-1-norm}(A)$
2: $\hat{A} \leftarrow \text{scale-}\infty\text{-norm}(\hat{A}_1)$
3: $\hat{A}_f \leftarrow \text{filter}(1.0 - \varepsilon \leq \hat{A} \leq 1.0 + \varepsilon)$
4: if there exist a perfect matching in $\hat{A}_f$ then
5: return the matching
6: else
7: ...  

Details of the fast solutions (among 180 matrices)

in 155, $\text{nnz}(\hat{A}_f) = n$; in the rest maximum three of $\text{nnz}(\hat{A}_f)/n$ are {5.87, 5.74, 1.03}

Memory requirements: $\hat{A}_f$ vs $A$ (of the 180 instances)

<table>
<thead>
<tr>
<th>$\text{nnz}(A)/n$</th>
<th>min</th>
<th>avg</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.25</td>
<td>22.05</td>
<td>132.36</td>
</tr>
<tr>
<td>$\text{nnz}(\hat{A}_f)/n$</td>
<td>1.00</td>
<td>1.06</td>
<td>5.87</td>
</tr>
</tbody>
</table>
Others (88/268 matrices, select and re-scale loop executed)

- averaging 8294 iterations after the initial 1-norm (at most 40 iterations) and $\infty$-norm (at most 20 iterations) scalings, mostly belonging to the matrix families Schenk\_IBMNA (27 matrices), GHS\_indef (25 matrices), and Nemeth (13 matrices).

- considerable savings in memory requirements:

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>avg</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{nnz}(A)/n$</td>
<td>3.22</td>
<td>24.97</td>
<td>159.03</td>
</tr>
<tr>
<td>$\text{nnz}(\hat{A}_f)/n$</td>
<td>1.00</td>
<td>1.53</td>
<td>2.60</td>
</tr>
</tbody>
</table>

- However, we do not want to do iterations.
  - although very sparse, the dmperm update requires DFS/BFS-like algorithms—inherently sequential.
  - we can reduce to a single processor and solve the problem there—too much iterations.

Sub-optimal alternatives may be acceptable.
A sub-optimal solution

Algorithm

1: $\hat{A}_1 \leftarrow \text{scale-1-norm}(A)$
2: $\hat{A} \leftarrow \text{scale-$\infty$-norm}(\hat{A}_1)$
3: $\hat{A}_f \leftarrow \text{filter}(1.0 - \varepsilon \leq \hat{A} \leq 1.0 + \varepsilon)$
4: if there exist a perfect matching in $\hat{A}_f$ then
5: return the matching
6: else
7: Compute a maximum matching using only the entries in $\hat{A}_f$
8: $\hat{A}_w \leftarrow \hat{A}_f$
9: $L \leftarrow \text{sort the entries of } \hat{A} - \hat{A}_f$
10: for $k = 1, 2, \ldots$ do
11: add entries from $L$ in decreasing order to $\hat{A}_w$ such that all unmatched rows and columns get at most one more entry
12: if not possible, add at most one more entry per each row and column
13: Augment the matching (weighted)
14: if a perfect matching obtained then
15: return the matching
Experiments (sub-optimal solution)

On 88/268 matrices (solution is not obtained after the first scaling steps)

### Quality of the matching

Compare $V = \sum \log \text{diag}(AM)$ and $V^* = \sum \log \text{diag}(AM^*)$

<table>
<thead>
<tr>
<th></th>
<th>$(V^* - V)/V^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>0.00</td>
</tr>
<tr>
<td>avg</td>
<td>0.17</td>
</tr>
<tr>
<td>max</td>
<td>12.15</td>
</tr>
</tbody>
</table>

Largest 5 values: 12.15 0.54 0.28 0.21 0.17

### Iterations and memory requirements

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>avg</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{nnz(A)}{n}$</td>
<td>3.22</td>
<td>24.97</td>
<td>159.03</td>
</tr>
<tr>
<td>$\frac{nnz(\hat{A}_w)}{n}$</td>
<td>2.00</td>
<td>2.52</td>
<td>3.84</td>
</tr>
<tr>
<td>iters</td>
<td>1</td>
<td>3.05</td>
<td>38</td>
</tr>
</tbody>
</table>

On 13 instances, number of augmentation iterations is greater than 3.
Summary

- On 155/268 matrices, at most 40 iterations of 1-norm scaling and then at most 20 iterations of \( \infty \)-norm scaling suffices to compute a maximum product matching.
- On another 25 matrices, with a little sequential overhead an optimum matching is obtained.
- On the others (88/268): Sub-optimal solutions can be found with fairly small additional, sequential work.

On going and future work

- The effects on factorization (already done a few experiments and observed that sub-optimal solutions are not worse than the optimal ones in terms of some factorization metrics)
- Sub-optimal solutions with approximation guarantee,
- Matrices with support but without total support.
So a new algorithm for an old problem?

- Not really 😐
- Anything that forms a barrier for the efficient parallelization of the standard algorithms for the bipartite matching problems apply to our case too.

Probably some good news too 😊

- Reap the developments on Hungarian algorithm.
- Our algorithms can be used to warm-start Hungarian algorithm for the serial execution case.
Further information

Thank you for your attention.

http://graal.ens-lyon.fr/~bucar/

bora.ucar@ens-lyon.fr
Number of iterations with error rate of $\varepsilon = 1.0e^{-6}$

- $\infty$-norm: Always converges very fast. Average 11.
- 1- and 2-norms: Did not converge on 10 and 17 matrices in 5000 iterations, respectively.
  - Average number of iterations in converged cases are 206 and 257,
  - Matrices from two groups (GHS_indef and Schenk_IBMNA) cause problems (larger number of iterations as well). 60 matrices from these groups.
  - Excluding those matrices, the averages are 26 and 29.