# A note on a GPU-based Network Simplex Algorithm

Stefano Gualandi[a] and L.M. Rousseau[b]

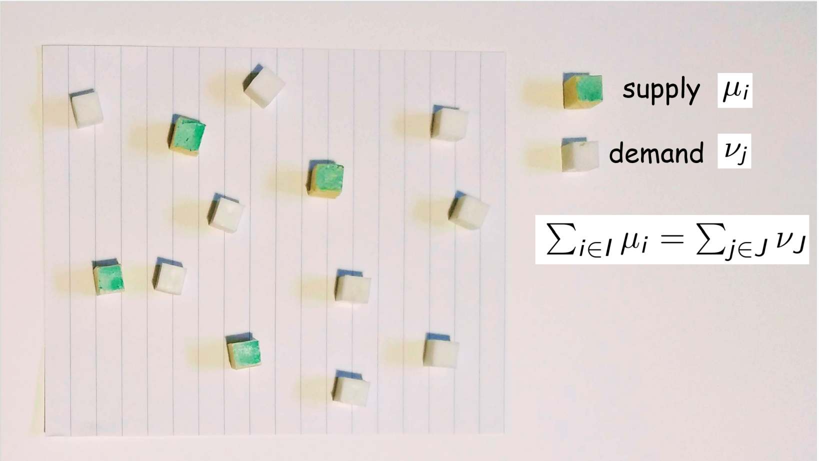
([a]) Università di Pavia, Dipartimento di Matematica

([b]) CIRRELT-Polytech. Montrèal, Dept de Math et Gènie Industriel

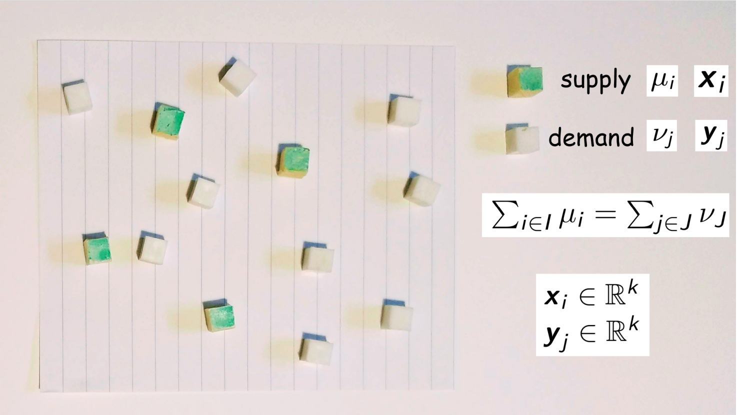
**Aussois 2020**

email: stefano.gualandi@unipv.it
twitter: @famo2spaghi
blog: http://stegua.github.com

# Balanced Transportation Problem

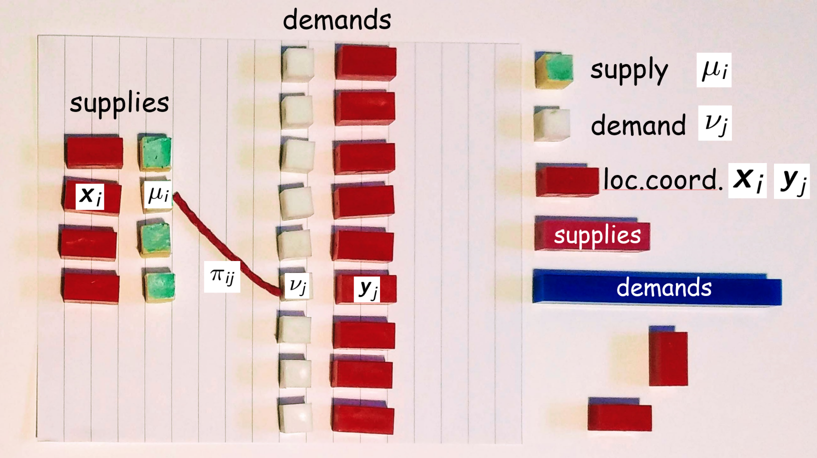


# Computation of Wasserstein distances [Cut13, PC+19]

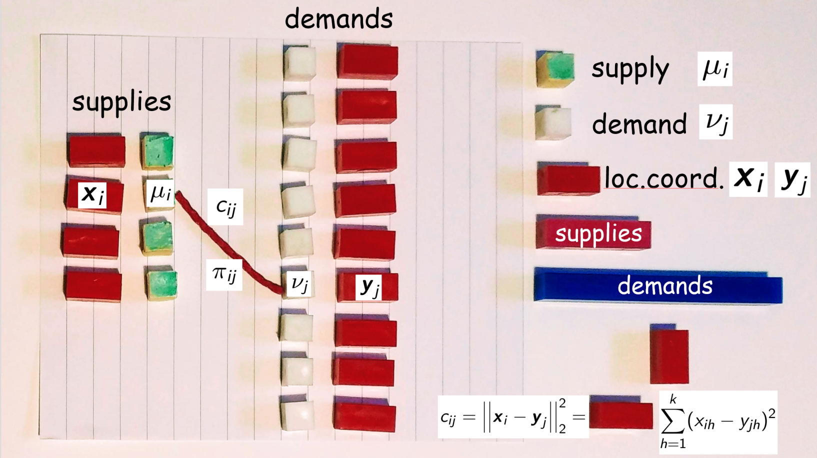



- $k = 2$: grey scale images [RTG00, BGV18, ABGV18]
- $k = 3$: color images [PW09], origin of universe [FMMS02]
- $k = 300$: word embedding [KSKW15]
- $k = 200$: gene-expression (work in progress)

# Transportation Problem as Min Cost Flow



# Geometric Transportation Problem



# Transportation Problem: LP model

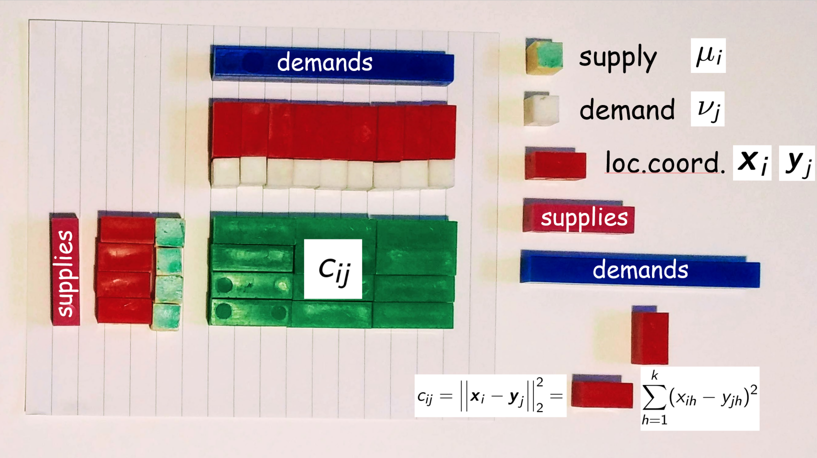
Given a bipartite graph $G = (I \cup J, E)$,

$$
\begin{aligned}
\min \quad & \sum_{\{i,j\} \in E} c_{ij}\pi_{ij} && && (1)\\
\text{s.t.} \quad & \sum_{\{i,j\} \in E} \pi_{ij} = \mu_i, && \forall i \in I && (2)\\
& \sum_{\{i,j\} \in E} \pi_{ij} = \nu_j, && \forall j \in J && (3)\\
\text{(flow variables)} \quad & \pi_{ij} \geq 0, && \forall \{i,j\} \in E. && (4)
\end{aligned}
$$

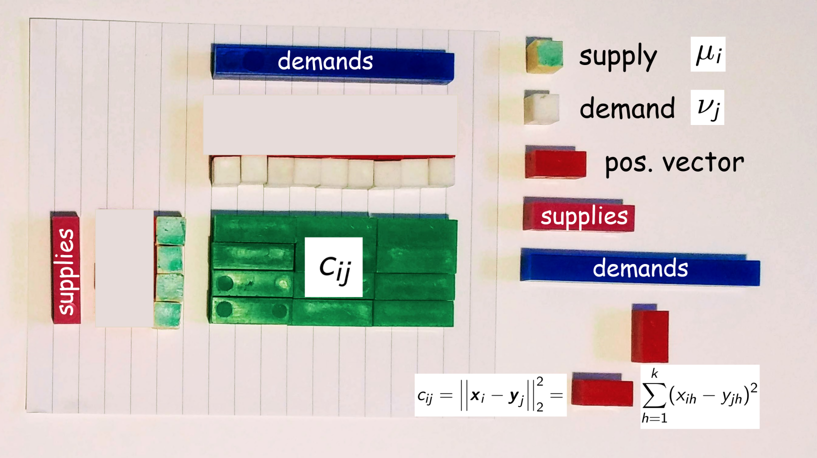
We consider **balanced** problem: $\sum_{i \in I} \mu_i = \sum_{j \in J} \nu_J$.

We have a linear number of constraints: $|I| + |J|$,
a quadratic number of variables: $|I| \times |J|$,
but only a linear number of basic variables: $|I| + |J| - 1$.

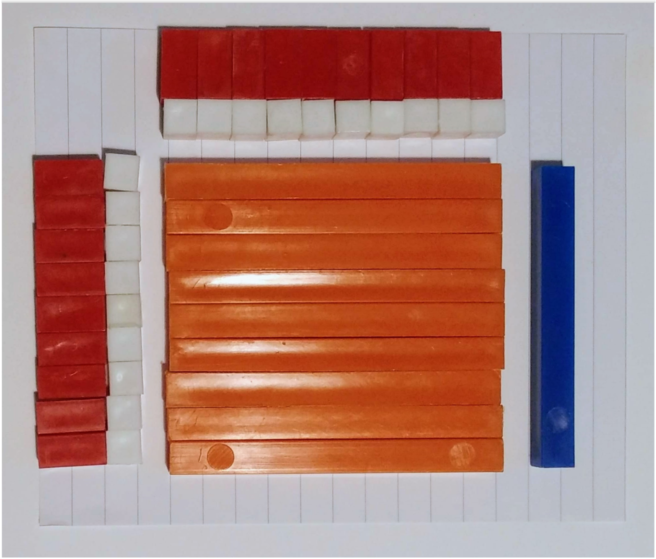
# Dense Geometric Transportation Problem



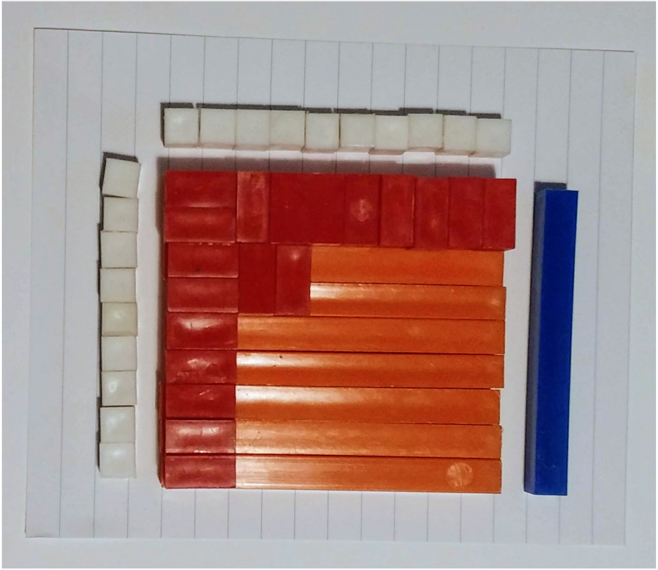
# Can we compute the cost coefficients $c_{ij}$ on the fly?



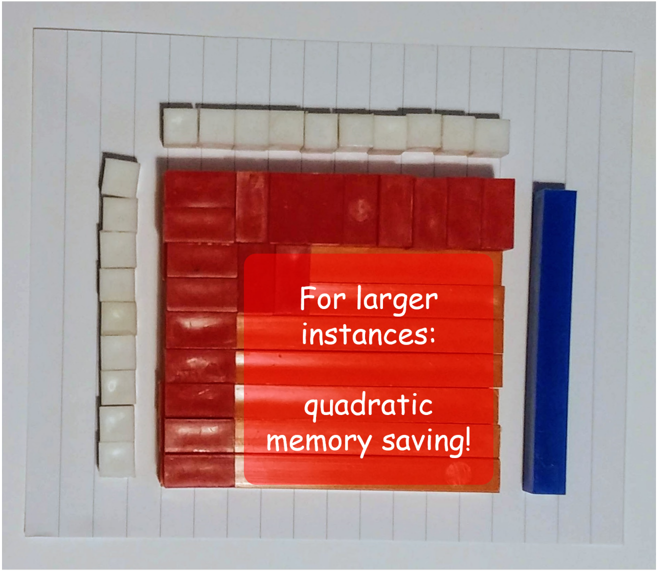
# Can we compute the cost coefficients $c_{ij}$ on the fly?



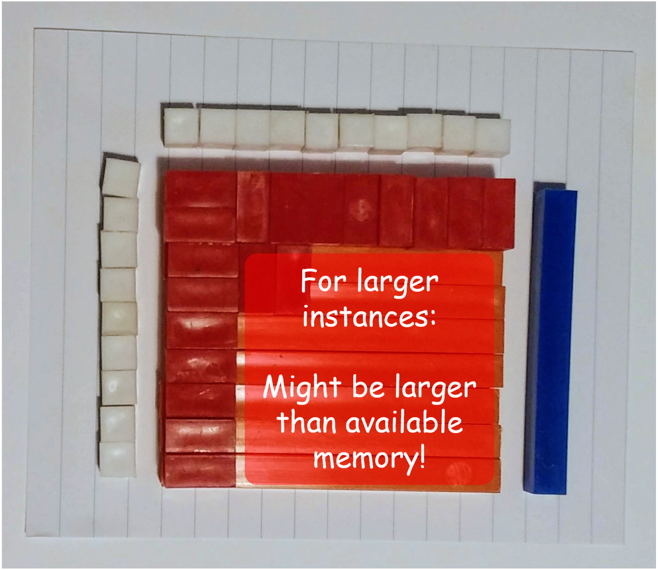
# Can we compute the cost coefficients $c_{ij}$ on the fly?



# Can we compute the cost coefficients $c_{ij}$ on the fly?



# Can we compute the cost coefficients $c_{ij}$ on the fly?



# Call for Column Generation


(a) Restricted Master Problem

$$\min \sum_{\{i,j\}\in\bar{E}} c_{ij}\pi_{ij} \tag{2}$$

$$\text{s.t.} \sum_{\{i,j\}\in\bar{E}} \pi_{ij} \geq \mu_i, \forall i \in I \tag{3}$$

$$\sum_{\{i,j\}\in\bar{E}} \pi_{ij} \leq \nu_j, \forall j \in J \tag{4}$$

$$\pi_{ij} \geq 0, \forall \{i,j\} \in \bar{E}. \tag{5}$$

# LP Simplex vs Network Simplex

| (a) **LP Simplex Algorithm** | (b) **Network Simplex Algorithm** |
|---|---|
| 1. Generate Initial BFS | 1. Generate Initial Basis Tree |
| 2. Choose Entering Variable | 2. Choose Entering Arc |
| 3. Determine Leaving Variable | 3. Determine Leaving Arc |
| 4. Move to New Basic Solution | 4. Move to New Basic Tree |

Steps 2–4 are repeated until an optimal solution is found (no negative reduced cost arc/variable exists). We refer to:

# LP Simplex vs Network Simplex

| (a) **LP Simplex Algorithm** | (b) **Network Simplex Algorithm** |
|---|---|
| 1. Generate Initial BFS | 1. Generate Initial Basis Tree |
| 2. Choose Entering Variable | 2. Choose Entering Arc |
| 3. Determine Leaving Variable | 3. Determine Leaving Arc |
| 4. Move to New Basic Solution | 4. Move to New Basic Tree |

Steps 2–4 are repeated until an optimal solution is found (no negative reduced cost arc/variable exists). We refer to:

The **best sequential implementation** of the Network Simplex Algorithm is contained in the COIN-OR Lemon Graph Library [Kov15]

# LP Simplex vs Network Simplex

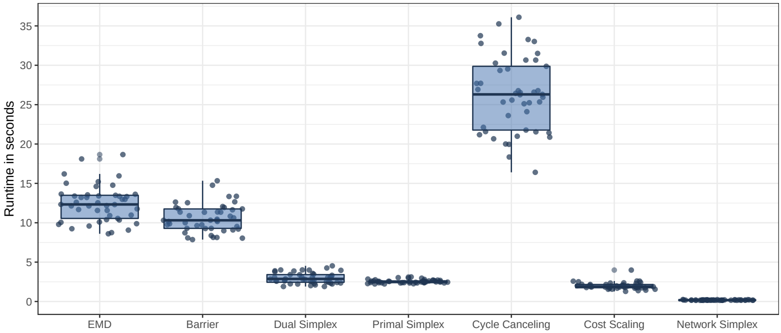
| (a) **LP Simplex Algorithm** | (b) **Network Simplex Algorithm** |
|---|---|
| 1. Generate Initial BFS | 1. Generate Initial Basis Tree |
| 2. Choose Entering Variable | 2. Choose Entering Arc |
| 3. Determine Leaving Variable | 3. Determine Leaving Arc |
| 4. Move to New Basic Solution | 4. Move to New Basic Tree |

Steps 2–4 are repeated until an optimal solution is found (no negative reduced cost arc/variable exists). We refer to:

The **best sequential implementation** of the Network Simplex Algorithm is contained in the COIN-OR Lemon Graph Library [Kov15]

The **best parallel implementation** of the Network Simplex Algorithm is given by [BVDPPH11], which is yet a fork of Lemon

# Network Simplex vs. Other Methods [BGV18]




Barrier, Primal, and Dual Simplex refer to Gurobi v8.0

Cycle Canceling, Cost Scaling, and Network Simplex to COIN-OR Lemon

# Parallel Network Simplex

| (a) **LP Simplex Algorithm** | (b) **Network Simplex Algorithm** |
| --- | --- |
| 1. Generate Initial BFS | 1. Generate Initial Basis Tree |
| 2. Choose Entering Variable | 2. **Choose Entering Arc (in parallel)** |
| 3. Determine Leaving Variable | 3. Determine Leaving Arc (2 threads) |
| 4. Move to New Basic Solution | 4. Move to New Basic Tree |

Steps 2–4 are repeated until an optimal solution is found (no negative reduced cost arc/variable exists). We refer to:

For a review of parallel implementation: *Towards a practical parallelisation of the simplex method*, by J.A.J Hall [Hal10].

For a parallel Network Simplex algorithm: *Parallel simplex for large pure network problems: Computational testing and sources of speedup* [BH94].

To avoid cycling: *Strong Feasible Basis* [Cun76]

# Parallel Network Simplex

| (a) **LP Simplex Algorithm** | (b) **Network Simplex Algorithm** |
| --- | --- |
| 1. Generate Initial BFS | 1. Generate Initial Basis Tree |
| 2. Choose Entering Variable | 2. **Choose Entering Arc (in parallel)** |
| 3. Determine Leaving Variable | 3. Determine Leaving Arc (2 threads) |
| 4. Move to New Basic Solution | 4. Move to New Basic Tree |

Steps 2–4 are repeated until an optimal solution is found (no negative reduced cost arc/variable exists). We refer to:

For a review of parallel implementation: *Towards a practical parallelisation of the simplex method*, by J.A.J Hall [Hal10].

For a parallel Network Simplex algorithm: *Parallel simplex for large pure network problems: Computational testing and sources of speedup* [BH94].

To avoid cycling: *Strong Feasible Basis* [Cun76]

We are not aware of any successful implementation of the Network Simplex using a **modern GPU**.

# Column (or cut) generation perspective

Considering a subset of the arc variables $\bar{E} \subset E$:

(a) Restricted Master Problem

$$
\begin{aligned}
\min \quad & \sum_{\{i,j\} \in \bar{E}} c_{ij} \pi_{ij} && (5)\\
\text{s.t.} \quad & \sum_{\{i,j\} \in \bar{E}} \pi_{ij} \geq \mu_i, \forall i \in I && (6)\\
& \sum_{\{i,j\} \in \bar{E}} \pi_{ij} \leq \nu_j, \forall j \in J && (7)\\
& \pi_{ij} \geq 0, \forall \{i,j\} \in \bar{E}. && (8)
\end{aligned}
$$

(b) Dual Restricted Master Problem

$$
\begin{aligned}
\max \quad & \sum_{i \in I} \mu_i u_i - \sum_{j \in J} \nu_j v_j && (9)\\
\text{s.t.} \quad & u_i - v_j \leq c_{ij}, \forall \{i,j\} \in \bar{E} && (10)\\
& u_i \geq 0, \forall i \in I && (11)\\
& v_j \geq 0, \forall j \in J. && (12)
\end{aligned}
$$

The pricing (separation) problem is:

$$
(\mathrm{P}_1) \quad c^*_{ij} = \min_{\{i,j\} \in E \setminus \bar{E}} c_{ij} - \bar{u}_i + \bar{v}_j. \qquad (13)
$$

Separation of constraint (10) is “embarrassingly simple”,

# Column (or cut) generation perspective

Considering a subset of the arc variables $\bar{E} \subset E$:

(a) Restricted Master Problem

$$\min \sum_{\{i,j\} \in \bar{E}} c_{ij} \pi_{ij} \tag{5}$$

$$\text{s.t.} \sum_{\{i,j\} \in \bar{E}} \pi_{ij} \geq \mu_i, \forall i \in I \tag{6}$$

$$\sum_{\{i,j\} \in \bar{E}} \pi_{ij} \leq \nu_j, \forall j \in J \tag{7}$$

$$\pi_{ij} \geq 0, \forall \{i,j\} \in \bar{E}. \tag{8}$$

(b) Dual Restricted Master Problem

$$\max \sum_{i \in I} \mu_i u_i - \sum_{j \in J} \nu_j v_j \tag{9}$$

$$\text{s.t.} \quad u_i - v_j \leq c_{ij}, \forall \{i,j\} \in \bar{E} \tag{10}$$

$$u_i \geq 0, \forall i \in I \tag{11}$$

$$v_j \geq 0, \forall j \in J. \tag{12}$$

The pricing (separation) problem is:

$$(\mathrm{P}_1) \quad c^*_{ij} = \min_{\{i,j\} \in E \setminus \bar{E}} c_{ij} - \bar{u}_i + \bar{v}_j. \tag{13}$$

Separation of constraint (10) is "embarrassingly simple", hence, well suited for **GPU computation**

# A closer look at the pricing subproblem

We can rewrite the pricing subproblem as

$$(\mathrm{P}_2) \quad c^*_{ij} = \min_{i \in I} \{\delta_i\} \tag{14}$$

$$\text{where} \quad \delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\} \tag{15}$$

# A closer look at the pricing subproblem

We can rewrite the pricing subproblem as

$$(\mathrm{P}_2) \quad c^*_{ij} = \min_{i \in I} \{\delta_i\} \tag{14}$$

$$\text{where} \quad \delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\} \tag{15}$$

Using the squared Euclidean distance, we get (for $||\cdot||_2$):

$$\delta_i = \min_{j \in J} \left\{ \left|\left| \boldsymbol{x}_i - \boldsymbol{y}_j \right|\right|^2 - \bar{u}_i + \bar{v}_j \right\} = \min_{j \in J} \left\{ \sum_{h=1}^{k} (x_{ih} - y_{jh})^2 - \bar{u}_i + \bar{v}_j \right\}$$

# A closer look at the pricing subproblem

We can rewrite the pricing subproblem as

$$(\mathrm{P}_2) \quad c_{ij}^* = \min_{i \in I} \{\delta_i\} \tag{14}$$

$$\text{where} \quad \delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\} \tag{15}$$

Using the squared Euclidean distance, we get (for $||\cdot||_2$):

$$\begin{aligned} \delta_i &= \min_{j \in J} \left\{ \left|\left| \boldsymbol{x}_i - \boldsymbol{y}_j \right|\right|^2 - \bar{u}_i + \bar{v}_j \right\} = \min_{j \in J} \left\{ \sum_{h=1}^{k} (x_{ih} - y_{jh})^2 - \bar{u}_i + \bar{v}_j \right\} \\ &= \min_{j \in J} \left\{ ||\boldsymbol{x}_i||^2 + ||\boldsymbol{y}_j||^2 - 2 \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle - \bar{u}_i + \bar{v}_j \right\} \end{aligned}$$

# A closer look at the pricing subproblem

We can rewrite the pricing subproblem as

$$(\mathrm{P}_2) \quad c^*_{ij} = \min_{i \in I} \{\delta_i\} \tag{14}$$

$$\text{where} \quad \delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\} \tag{15}$$

Using the squared Euclidean distance, we get (for $||\cdot||_2$):

$$\begin{aligned}
\delta_i &= \min_{j \in J} \left\{ \left|\left| \boldsymbol{x}_i - \boldsymbol{y}_j \right|\right|^2 - \bar{u}_i + \bar{v}_j \right\} = \min_{j \in J} \left\{ \sum_{h=1}^{k} (x_{ih} - y_{jh})^2 - \bar{u}_i + \bar{v}_j \right\} \\
&= \min_{j \in J} \left\{ ||\boldsymbol{x}_i||^2 + ||\boldsymbol{y}_j||^2 - 2 \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle - \bar{u}_i + \bar{v}_j \right\} \\
&= ||\boldsymbol{x}_i||^2 - \bar{u}_i + \min_{j \in J} \left\{ ||\boldsymbol{y}_j||^2 + \bar{v}_j - 2 \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle \right\}
\end{aligned}$$

# A closer look at the pricing subproblem

We can rewrite the pricing subproblem as

$$(\mathrm{P}_2) \quad c^*_{ij} = \min_{i \in I} \{\delta_i\} \tag{14}$$

$$\text{where} \quad \delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\} \tag{15}$$

Using the squared Euclidean distance, we get (for $||\cdot||_2$):

$$\begin{aligned}
\delta_i &= \min_{j \in J} \left\{ \left|\left| \boldsymbol{x}_i - \boldsymbol{y}_j \right|\right|^2 - \bar{u}_i + \bar{v}_j \right\} = \min_{j \in J} \left\{ \sum_{h=1}^{k} (x_{ih} - y_{jh})^2 - \bar{u}_i + \bar{v}_j \right\} \\
&= \min_{j \in J} \left\{ ||\boldsymbol{x}_i||^2 + ||\boldsymbol{y}_j||^2 - 2 \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle - \bar{u}_i + \bar{v}_j \right\} \\
&= ||\boldsymbol{x}_i||^2 - \bar{u}_i + \min_{j \in J} \left\{ ||\boldsymbol{y}_j||^2 + \bar{v}_j - 2 \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle \right\} \\
&= \tilde{u}_i + \min_{j \in J} \left\{ \tilde{v}_j - 2 \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle \right\} =
\end{aligned}$$

# A closer look at the pricing subproblem

We can rewrite the pricing subproblem as

$$(\mathrm{P}_2) \quad c^*_{ij} = \min_{i \in I} \{\delta_i\} \tag{14}$$

$$\text{where} \quad \delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\} \tag{15}$$

Using the squared Euclidean distance, we get (for $||\cdot||_2$):

$$\begin{aligned}
\delta_i &= \min_{j \in J} \left\{ \left|\left|\boldsymbol{x}_i - \boldsymbol{y}_j\right|\right|^2 - \bar{u}_i + \bar{v}_j \right\} = \min_{j \in J} \left\{ \sum_{h=1}^{k} (x_{ih} - y_{jh})^2 - \bar{u}_i + \bar{v}_j \right\} \\
&= \min_{j \in J} \left\{ ||\boldsymbol{x}_i||^2 + ||\boldsymbol{y}_j||^2 - 2\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle - \bar{u}_i + \bar{v}_j \right\} \\
&= ||\boldsymbol{x}_i||^2 - \bar{u}_i + \min_{j \in J} \left\{ ||\boldsymbol{y}_j||^2 + \bar{v}_j - 2\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle \right\} \\
&= \tilde{u}_i + \min_{j \in J} \left\{ \tilde{v}_j - 2\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle \right\} = \tilde{u}_i + \min_{j \in J} \left\{ \tilde{v}_j - 2\sum_{h=1}^{k} x_{ih}\, y_{jh} \right\}
\end{aligned} \tag{16}$$

# A closer look at the pricing subproblem

We can rewrite the pricing subproblem as

$$(\mathrm{P}_2) \quad c_{ij}^* = \min_{i \in I} \{\delta_i\} \tag{14}$$

$$\text{where} \quad \delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\} \tag{15}$$

Using the squared Euclidean distance, we get (for $||\cdot||_2$):

$$\begin{aligned}
\delta_i &= \min_{j \in J} \left\{ \left|\left| \boldsymbol{x}_i - \boldsymbol{y}_j \right|\right|^2 - \bar{u}_i + \bar{v}_j \right\} = \min_{j \in J} \left\{ \sum_{h=1}^{k} (x_{ih} - y_{jh})^2 - \bar{u}_i + \bar{v}_j \right\} \\
&= \min_{j \in J} \left\{ ||\boldsymbol{x}_i||^2 + ||\boldsymbol{y}_j||^2 - 2 \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle - \bar{u}_i + \bar{v}_j \right\} \\
&= ||\boldsymbol{x}_i||^2 - \bar{u}_i + \min_{j \in J} \left\{ ||\boldsymbol{y}_j||^2 + \bar{v}_j - 2 \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle \right\} \\
&= \tilde{u}_i + \min_{j \in J} \left\{ \tilde{v}_j - 2 \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle \right\} = \tilde{u}_i + \min_{j \in J} \left\{ \tilde{v}_j - 2 \sum_{h=1}^{k} x_{ih}\, y_{jh} \right\}
\end{aligned} \tag{16}$$

We can pre-compute $||\boldsymbol{x}_i||^2$ and $||\boldsymbol{y}_j||^2$ once for all, and $\tilde{u}_i$ and $\tilde{v}_j$ once per pricing. The important computation is the dot product $\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$.

# A closer look at the pricing subproblem

We can rewrite the pricing subproblem as

$$(\mathrm{P}_2) \quad c^*_{ij} \quad = \quad \min_{i \in I} \{\delta_i\} \tag{17}$$

$$\text{where} \qquad \delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\} \tag{18}$$

Using the squared Euclidean distance, we get (for $||\cdot||_2$):

$$\delta_i \quad = \quad \tilde{u}_i + \min_{j \in J} \left\{ \tilde{v}_j - 2 \left\langle \boldsymbol{x}_i, \boldsymbol{y}_j \right\rangle \right\}$$

Let $\boldsymbol{X}$ be the matrix with a row for each vector $\boldsymbol{x}_i$, and $\boldsymbol{Y}$ be the matrix with a column for each vector $\boldsymbol{y}_j$, then, in vector notation:

$$\boldsymbol{\delta} = \tilde{\boldsymbol{u}} + f(\tilde{\boldsymbol{v}}, \boldsymbol{X}\boldsymbol{Y})$$

# A closer look at the pricing subproblem

We can rewrite the pricing subproblem as

$$(P_2) \quad c^*_{ij} = \min_{i \in I} \{\delta_i\} \tag{17}$$

$$\text{where} \quad \delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\} \tag{18}$$

Using the squared Euclidean distance, we get (for $||\cdot||_2$):

$$\delta_i = \tilde{u}_i + \min_{j \in J} \left\{ \tilde{v}_j - 2 \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle \right\}$$

Let $\boldsymbol{X}$ be the matrix with a row for each vector $\boldsymbol{x}_i$, and $\boldsymbol{Y}$ be the matrix with a column for each vector $\boldsymbol{y}_j$, then, in vector notation:

$$\boldsymbol{\delta} = \tilde{\boldsymbol{u}} + f(\tilde{\boldsymbol{v}}, \boldsymbol{XY})$$

... matrix multiplication is exactly what GPU are good for!

# Pre-tests to skip and stop pricing subproblems 1/2

Still, whenever is possible we want to avoid to compute $\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$

# Pre-tests to skip and stop pricing subproblems 1/2

Still, whenever is possible we want to avoid to compute $\langle \mathbf{x}_i, \mathbf{y}_j \rangle$

**Lemma 1 (Bounding the pricing problem per node)**

*Given the following:*

1. $\underline{c}_i = \min_j\{c_{ij}\}$ *(Precomputed only once)*
2. $\underline{v} = \min_j\{\bar{v}_j\}$ *(Precomputed once per pricing)*
3. $\bar{\delta}_i < 0$ *current best cut violation* *(i-th incumbent)*

*Whenever*

$$\bar{u}_i \leq \underline{c}_i + \underline{v} - \bar{\delta}_i, \tag{19}$$

*Then, $\bar{\delta}_i$ is the optimal value for the i-th pricing suproblem:*

$$\delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\}$$

*(... and hence, we can skip or stop the computation for $\langle \mathbf{x}_i, \mathbf{y}_j \rangle$)*

# Pre-tests to skip and stop pricing subproblems 2/2

Still, whenever is possible we want to avoid to compute $\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$

**Lemma 2 (Bounding the pricing problem per arc)**

*Given a node $j \in J$ such that*

$$\bar{u}_i - \bar{v}_j > c_{ij} \qquad \text{and let } \bar{c}_{ij} = c_{ij} - \bar{u}_i - \bar{v}_j$$

*then, for every other node $h \in J \setminus \{j\}$ such that*

$$||y_h||^2 + v_h - \bar{c}_{ij} > 2\,||x_i||\;||y_h|| \tag{20}$$

*we can avoid to compute $\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$.*

# Pre-tests to skip and stop pricing subproblems 2/2

Still, whenever is possible we want to avoid to compute $\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$

## Lemma 2 (Bounding the pricing problem per arc)

*Given a node $j \in J$ such that*

$$\bar{u}_i - \bar{v}_j > c_{ij} \qquad \text{and let } \bar{c}_{ij} = c_{ij} - \bar{u}_i - \bar{v}_j$$

*then, for every other node $h \in J \setminus \{j\}$ such that*

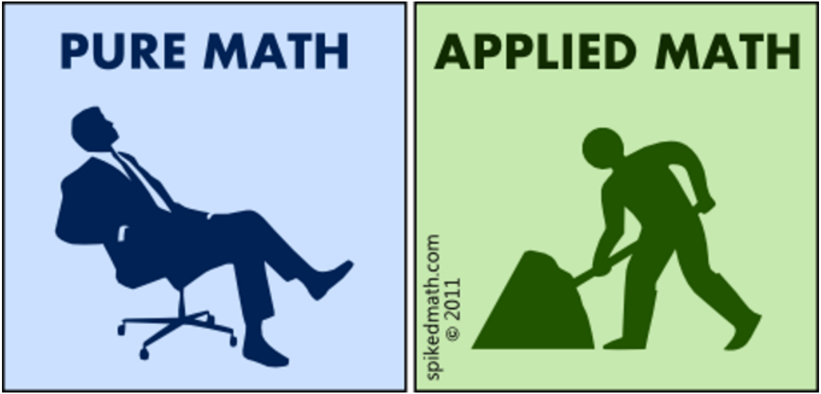
$$||y_h||^2 + v_h - \bar{c}_{ij} > 2\,||x_i||\,||y_h|| \tag{20}$$

*we can avoid to compute $\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$.*

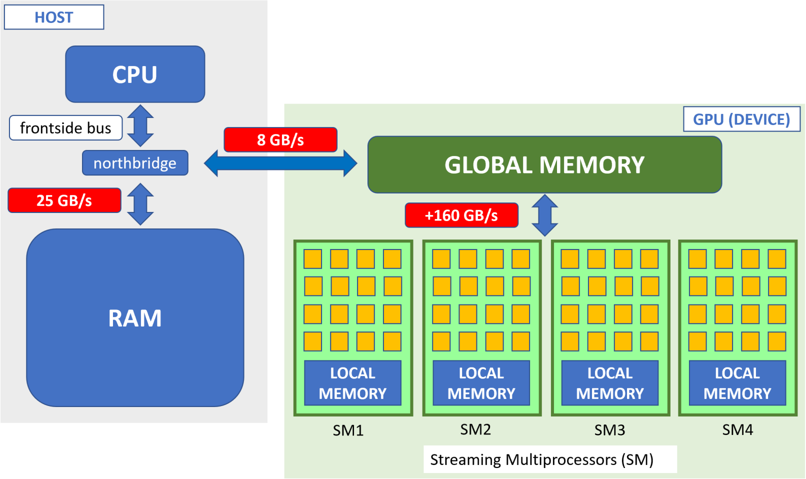
Where in the proof we exploit the cost structure:

$$\begin{aligned} c_{ij} &= ||x_i||^2 + ||y_j||^2 - 2\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle && (21) \\ &\geq ||x_i||^2 + ||y_j||^2 - 2|\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle| && (22) \\ &\geq ||x_i||^2 + ||y_j||^2 - 2\,||x_i||\,||y_j||\,. && (23) \end{aligned}$$

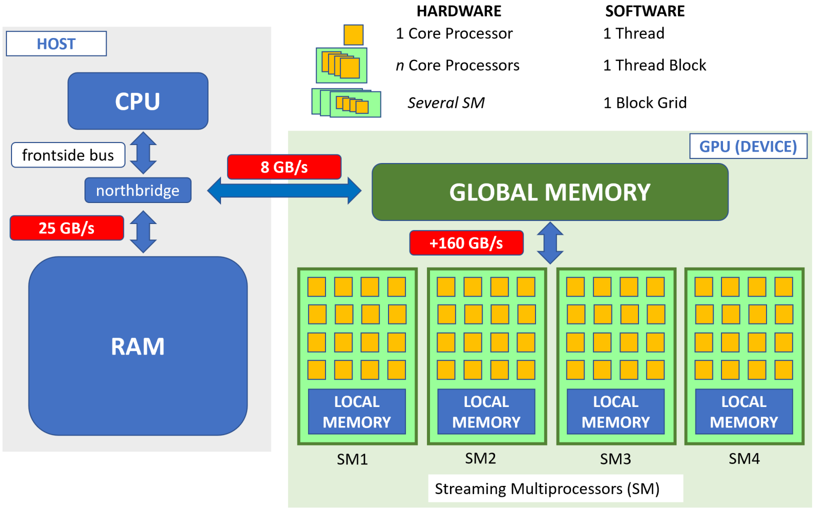
# From Theory to Practice



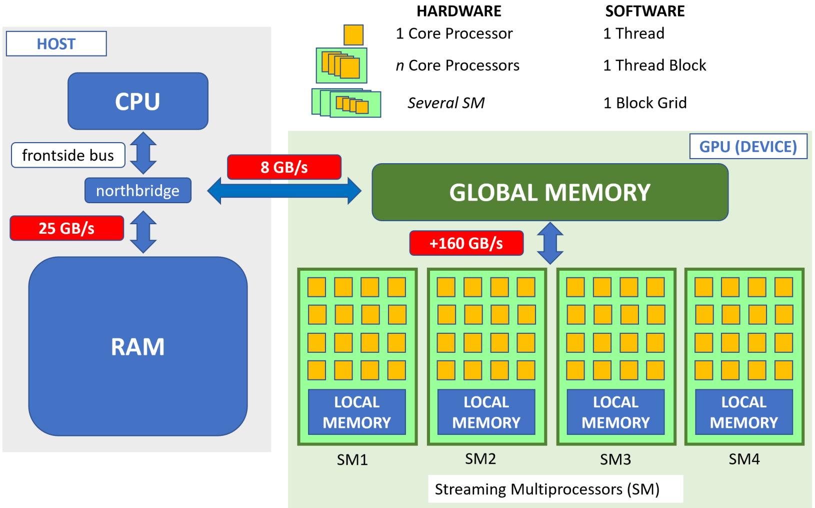
# Memory Bandwidth Bottlenecks



# Threading Hierarchy

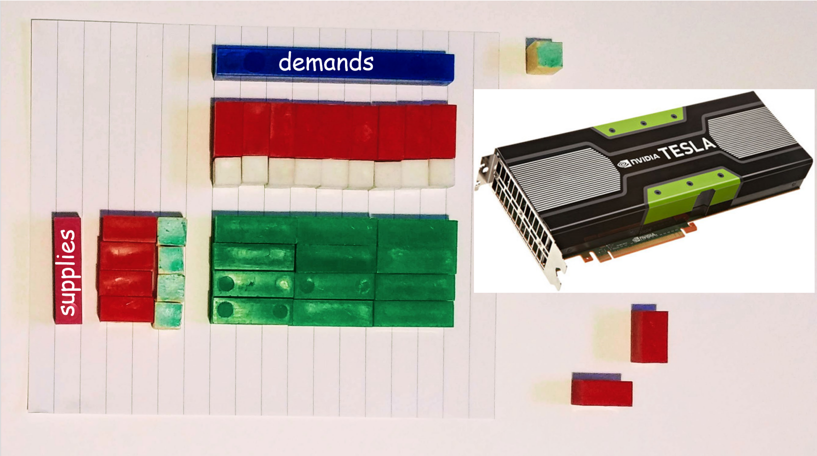


# Threading Hierarchy

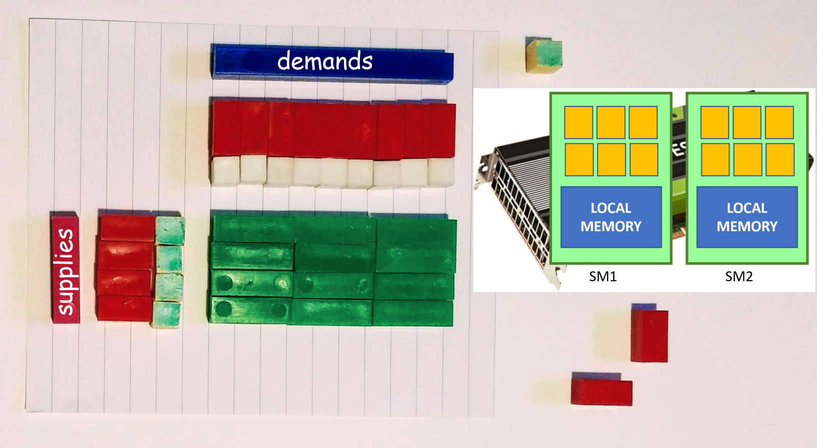



NVIDIA Quadro P6000 has 60 SM with 64 cores each: 3840 cores in total

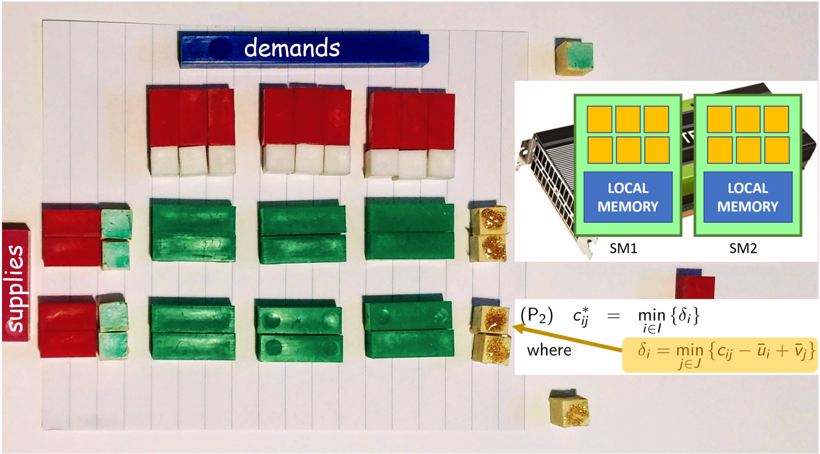
# GPU-based Implementation: Simplified Model



# GPU-based Implementation: Simplified Model



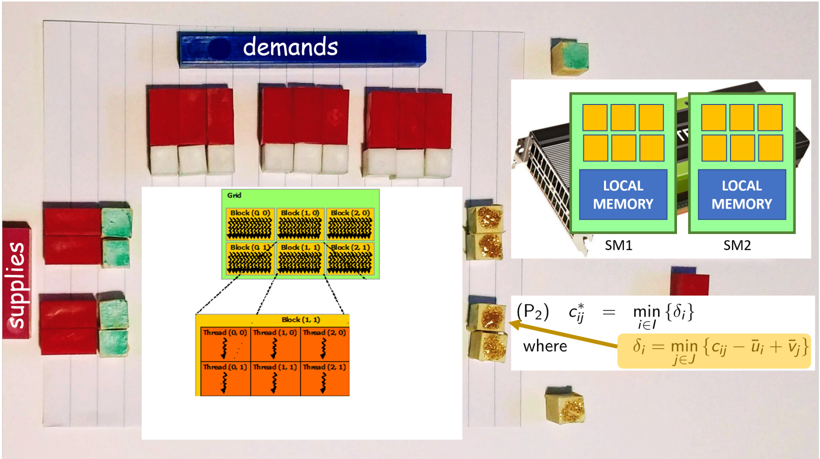
# GPU-based Implementation: Simplified Model




$$(\mathrm{P}_2) \quad c^*_{ij} = \min_{i \in I} \{\delta_i\}$$

where $\delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\}$

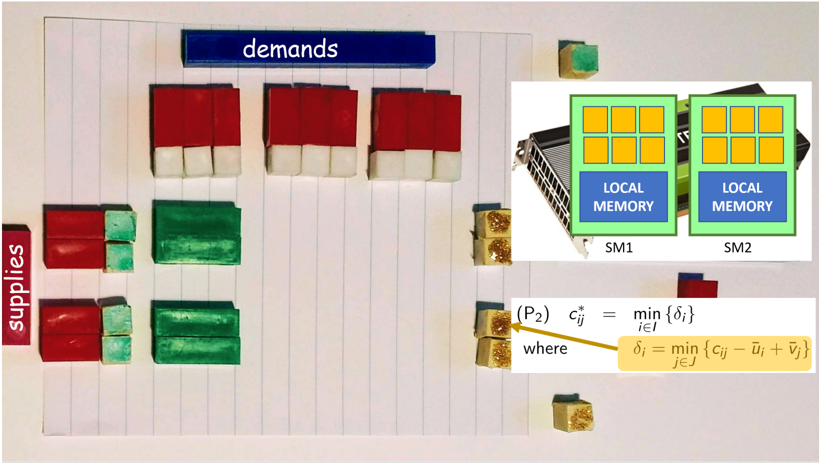
# GPU-based Implementation: Simplified Model




Each single GPU thread computes: $\tilde{v}_j - 2\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$ with $j \in \bar{J}$

Each thread block computes: $\tilde{\delta}_i = \min_{j \in \bar{J}} \left\{ \tilde{v}_j - 2 \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle \right\}$, with $i \in \bar{I}$

# GPU-based Implementation: Simplified Model



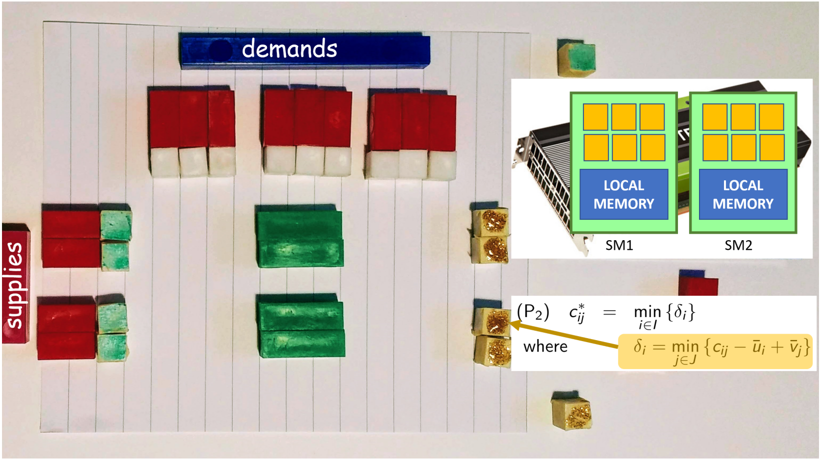

$$(\mathrm{P}_2) \quad c^*_{ij} = \min_{i \in I} \{\delta_i\}$$

where $\delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\}$

Each single GPU thread computes: $\tilde{v}_j - 2\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$ with $j \in \bar{J}$

Each thread block computes: $\tilde{\delta}_i = \min_{j \in \bar{J}} \{\tilde{v}_j - 2\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle\}$, with $i \in \bar{I}$

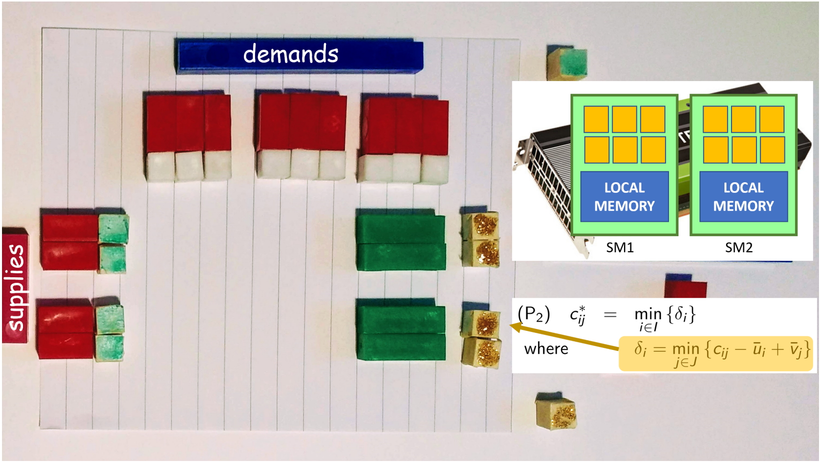
# GPU-based Implementation: Simplified Model




Each single GPU thread computes: $\tilde{v}_j - 2\langle \mathbf{x}_i, \mathbf{y}_j \rangle$ with $j \in \bar{J}$

Each thread block computes: $\tilde{\delta}_i = \min_{j \in \bar{J}} \left\{ \tilde{v}_j - 2 \langle \mathbf{x}_i, \mathbf{y}_j \rangle \right\}$, with $i \in \bar{I}$

# GPU-based Implementation: Simplified Model




$$(\mathrm{P}_2) \quad c_{ij}^* = \min_{i \in I} \{\delta_i\}$$

where $\delta_i = \min_{j \in J} \{c_{ij} - \bar{u}_i + \bar{v}_j\}$

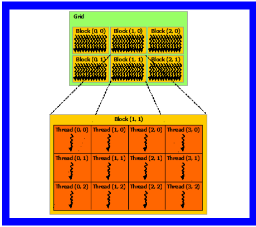
Each single GPU thread computes: $\tilde{v}_j - 2\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$ with $j \in \bar{J}$

Each thread block computes: $\tilde{\delta}_i = \min_{j \in \bar{J}} \left\{ \tilde{v}_j - 2 \left\langle \boldsymbol{x}_i, \boldsymbol{y}_j \right\rangle \right\}$, with $i \in \bar{I}$

# Pricing on the GPU: Threads, Blocks, and Grids

Each GPU block gets a subset of supplies $\bar{I} \subset I$ and demands $\bar{J} \subset J$

$$\tilde{\delta}_i = \min_{j \in \bar{J}} \left\{ \tilde{v}_j - 2 \sum_{h=1}^{k} x_{ih} \, y_{jh} \right\},$$




The GPU thread hierarchy is organized as follows:

- Check the pretests, and if passed:
- Each single GPU thread computes: $\tilde{v}_j - 2\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$
- Each thread (out of two) within a GPU block cooperates in finding the minimum over $\bar{J}$, using a **parallel reduction** algorithm (over the block-shared memory) [H+07].
- Inter block (grid) cooperation is achieved via **atomic** updates on the global GPU memory for computing for every $i \in I$ the optimal $\delta_i$.

In the end, we get in parallel the optimal $\delta_i$ for every $i \in I$.

# GPU-based Network Simplex for Dense Problems

**We solve a sequence of very sparse network problems.**
**We keep in memory only $O(|I| + |J|)$ arc variables.**

# GPU-based Network Simplex for Dense Problems

**We solve a sequence of very sparse network problems.**
**We keep in memory only $O(|I| + |J|)$ arc variables.**

- Copy from host to GPU: $\boldsymbol{x}_i, \boldsymbol{y}_j, ||\boldsymbol{y}_i||^2, ||\boldsymbol{y}_j||^2$
- Compute $\bar{E} \subset E$, and an initial basis tree

# GPU-based Network Simplex for Dense Problems

**We solve a sequence of very sparse network problems.**
**We keep in memory only $O(|I| + |J|)$ arc variables.**

- Copy from host to GPU: $\boldsymbol{x}_i, \boldsymbol{y}_j, ||\boldsymbol{y}_i||^2, ||\boldsymbol{y}_j||^2$
- Compute $\bar{E} \subset E$, and an initial basis tree
  1. Solve the corresponding sparse transportation problem using our sequential (incremental) Network Simplex algorithm
  2. Compute dual multipliers $\tilde{u}_i$ and $\tilde{v}_j$, and copy them on GPU

# GPU-based Network Simplex for Dense Problems

**We solve a sequence of very sparse network problems.**
**We keep in memory only $O(|I|+|J|)$ arc variables.**

- Copy from host to GPU: $\boldsymbol{x}_i, \boldsymbol{y}_j, ||\boldsymbol{y}_i||^2, ||\boldsymbol{y}_j||^2$
- Compute $\bar{E} \subset E$, and an initial basis tree

1. Solve the corresponding sparse transportation problem using our sequential (incremental) Network Simplex algorithm
2. Compute dual multipliers $\tilde{u}_i$ and $\tilde{v}_j$, and copy them on GPU
3. **Using the GPU: Compute $\delta_i$ for each supply.**
   If every $\delta_i \geq 0$, stop the algorithm

# GPU-based Network Simplex for Dense Problems

**We solve a sequence of very sparse network problems.**
**We keep in memory only $O(|I| + |J|)$ arc variables.**

- Copy from host to GPU: $\boldsymbol{x}_i, \boldsymbol{y}_j, ||\boldsymbol{y}_i||^2, ||\boldsymbol{y}_j||^2$
- Compute $\bar{E} \subset E$, and an initial basis tree

1. Solve the corresponding sparse transportation problem using our sequential (incremental) Network Simplex algorithm
2. Compute dual multipliers $\tilde{u}_i$ and $\tilde{v}_j$, and copy them on GPU
3. **Using the GPU: Compute $\delta_i$ for each supply.** If every $\delta_i \geq 0$, stop the algorithm
4. Whenever $\delta_i < 0$, copy from GPU to host the corresponding cost $c_{ij}$ and add arc $\{i, j\}$ to $\bar{E}$.
5. **Important**: Remove (aggressively) from $\bar{E}$ all the variables with a reduced costs greater than $\tau > 0$. Go back to (1).

# Multicore CPU Network Simplex for Dense Problems

**We solve a sequence of very sparse network problems.**
**We keep in memory only $O(|I| + |J|)$ arc variables.**

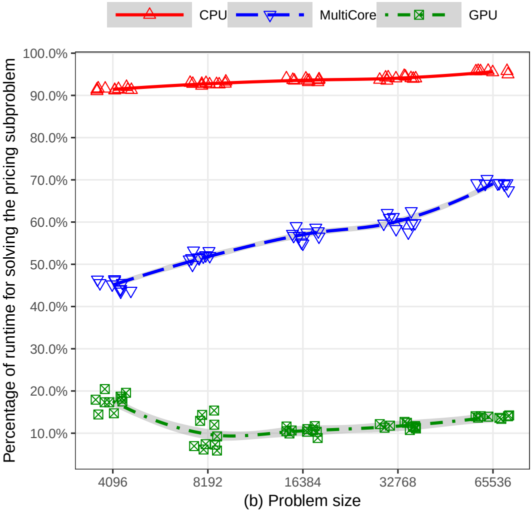
- ~~Copy from host to GPU: $\boldsymbol{x}_i, \boldsymbol{y}_j, ||\boldsymbol{y}_i||^2, ||\boldsymbol{y}_j||^2$~~
- Compute $\bar{E} \subset E$, and an initial basis tree

1. Solve the corresponding sparse transportation problem using our sequential (incremental) Network Simplex algorithm
2. Compute dual multipliers $\tilde{u}_i$ and $\tilde{v}_j$, ~~copy them on the GPU~~
3. **Using CPU cores: Compute $\delta_i$ for each supply.** If every $\delta_i \geq 0$, stop the algorithm
4. Whenever $\delta_i < 0$, ~~copy from GPU to host~~ the corresponding cost $c_{ij}$ and add arc $\{i, j\}$ to $\bar{E}$.
5. **Important**: Remove (aggressively) from $\bar{E}$ all the variables with a reduced costs greater than $\tau > 0$. Go back to (1).

# Implementation details: Code and Dataset

- All the algorithms coded in standard ANSI C++11
- First implementation using the Microsoft AMP C++ library. Current development using NVIDIA CUDA toolkit.
- Multicore CPU parallel algorithms use OpenMP 4.5.
- As benchmarks, with locations $\boldsymbol{x}_i, \boldsymbol{y}_j \in \mathbb{R}^2$, we use:
  1. Random assignment problems.
  2. **DOTmark** grey scale images [SSG17], a standard benchmark for computing Wasserstein distances.
- All results refer to a Dell workstation with an Intel Xeon CPU, 10 physical cores at 3.3GHz, 32GB of RAM, equipped with an NVIDIA Quadro P6000 GPU.

# Random Assignment - Pricing subproblems




Problem size refer to $|I| = |J|$.

# Random Assignment - Details for larger instances

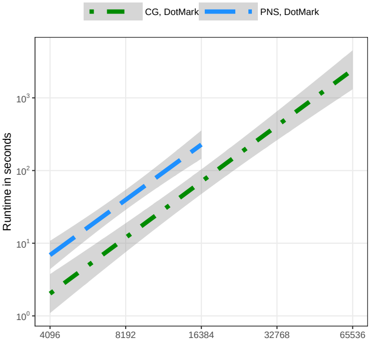
| Size | Method | CG Iter | Average Running time Master | Pricing | Total (stdev) | Vars % | RAM (MB) |
|---|---|---|---|---|---|---|---|
| 32 768 | CPU | 213.0 | 32.4 | 514.4 | 546.8 (61.8) | 0.35% | 69.3 |
| | MultiCore | 213.0 | 33.3 | 50.2 | 83.5 (9.5) | 0.35% | 69.8 |
| | GPU | 214.0 | 35.3 | 4.7 | 40.0 (4.5) | 0.35% | 57.5 |
| 65 536 | CPU | 506.0 | 209.3 | 4547.6 | 4756.9 (470.8) | 0.21% | 83.6 |
| | MultiCore | 504.1 | 203.6 | 454.4 | 658.0 (53.8) | 0.20% | 84.1 |
| | GPU | 497.1 | 220.1 | 35.4 | 255.6 (15.0) | 0.20% | 82.3 |

# Random Assignment - Details for larger instances

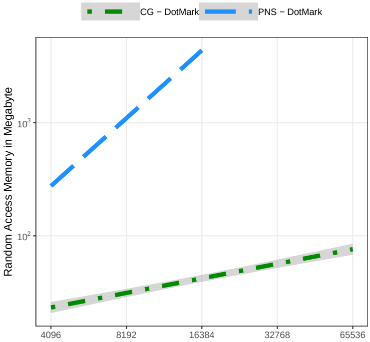
| Size | Method | CG Iter | Average Running time Master | Pricing | Total | (stdev) | Vars % | RAM (MB) |
|---|---|---|---|---|---|---|---|---|
| 32 768 | CPU | 213.0 | 32.4 | 514.4 | 546.8 | (61.8) | 0.35% | 69.3 |
| | MultiCore | 213.0 | 33.3 | 50.2 | 83.5 | (9.5) | 0.35% | 69.8 |
| | GPU | 214.0 | 35.3 | 4.7 | 40.0 | (4.5) | 0.35% | 57.5 |
| 65 536 | CPU | 506.0 | 209.3 | 4547.6 | 4756.9 | (470.8) | 0.21% | 83.6 |
| | MultiCore | 504.1 | 203.6 | 454.4 | 658.0 | (53.8) | 0.20% | 84.1 |
| | GPU | 497.1 | 220.1 | 35.4 | 255.6 | (15.0) | 0.20% | 82.3 |

# DOTmark grey scale images [SSG17] (45 inst. per size)

Comparison with the Parallel Network Simplex (PNS) [BVDPPH11], which stores the cost coefficient matrix on the RAM memory.




(a) Problem size




(b) Problem size

# Conclusions

1. We have implemented an incremental two staged GPU-based Network Simplex (source code coming soon on Github)

# Conclusions

1. We have implemented an incremental two staged GPU-based Network Simplex (source code coming soon on Github)
2. **Working with GPU is technically tricky**

# Conclusions

1. We have implemented an incremental two staged GPU-based Network Simplex (source code coming soon on Github)
2. **Working with GPU is technically tricky, but we can do it!**

# Conclusions

1. We have implemented an incremental two staged GPU-based Network Simplex (source code coming soon on Github)
2. **Working with GPU is technically tricky, but we can do it!**
3. Even when memory is not an issue, our approach is faster than storing the full matrix in memory (as in [BVDPPH11])

# Conclusions

1. We have implemented an incremental two staged GPU-based Network Simplex (source code coming soon on Github)
2. **Working with GPU is technically tricky, but we can do it!**
3. Even when memory is not an issue, our approach is faster than storing the full matrix in memory (as in [BVDPPH11])
4. We are currently working on a new single-cell RNA classification problem, where points $\boldsymbol{x}_i, \boldsymbol{y}_j \in \mathbb{R}^{200}$

# Thanks to the sponsor: NVIDIA

# Thanks to the sponsor: NVIDIA


We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro P6000 GPU used for this research.

# Questions?

Thanks!

Gennaro Auricchio, Federico Bassetti, Stefano Gualandi, and Marco Veneroni.
Computing Kantorovich-Wasserstein distances on d-dimensional histograms using (d+1)-partite graphs.
*Advances in Neural Information Processing Systems*, 2018.

Federico Bassetti, Stefano Gualandi, and Marco Veneroni.
On the computation of Kantorovich-Wasserstein distances between 2D-histograms by uncapacitated minimum cost flows.
*arXiv preprint arXiv:1804.00445*, 2018.

Richard S Barr and Betty L Hickman.
Parallel simplex for large pure network problems: Computational testing and sources of speedup.
*Operations Research*, 42(1):65–80, 1994.

Nicolas Bonneel, Michiel Van De Panne, Sylvain Paris, and Wolfgang Heidrich.
Displacement interpolation using Lagrangian mass transport.
*ACM Transactions on Graphics (TOG)*, 30(6):158, 2011.

William H Cunningham.

A network simplex method.
*Mathematical Programming*, 11(1):105–116, 1976.

Marco Cuturi.
Sinkhoirn distances: Lightspeed computation of optimal transport.
In *Advances in Neural Information Processing Systems*, pages 2292–2300, 2013.

Uriel Frisch, Sabino Matarrese, Roya Mohayaee, and Andrei Sobolevski.
A reconstruction of the initial conditions of the universe by optimal mass transportation.
*Nature*, 417(6886):260, 2002.

Mark Harris et al.
Optimizing parallel reduction in cuda.
*Nvidia developer technology*, 2(4):70, 2007.

JAJ Hall.
Towards a practical parallelisation of the simplex method.
*Computational Management Science*, 7(2):139–170, 2010.

Péter Kovács.
Minimum-cost flow algorithms: an experimental evaluation.

*Optim. Methods Softw.*, 30(1):94–127, 2015.

Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger.
From word embeddings to document distances.
In *International Conference on Machine Learning*, pages 957–966, 2015.

Gabriel Peyré, Marco Cuturi, et al.
Computational optimal transport.
*Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

Ofir Pele and Michael Werman.
Fast and robust Earth Mover's Distances.
In *Computer vision, 2009 IEEE 12th international conference on*, pages 460–467. IEEE, 2009.

Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas.
The Earth Mover's Distance as a metric for image retrieval.
*International Journal of Computer Vision*, 40(2):99–121, 2000.

Jörn Schrieber, Dominic Schuhmacher, and Carsten Gottschlich.
Dotmark–a benchmark for discrete optimal transport.
*IEEE Access*, 5:271–282, 2017.