A Novel Heterogeneous Algorithm for Multiplying Scale-Free Sparse Matrices

Kiran Raj Ramamoorthy, Dip Sankar Banerjee, Kannan Srinathan and Kishore Kothapalli.

C-STAR, IIIT Hyderabad.

Outline

- **Inspiration** :: Heterogeneous Platform & Challenges
- **Introduction** :: Sparse Matrix-Matrix Multiplication (SPMM)
- **Earlier Work** :: Row-Row (K. Matam et. al)
- **Our Approach** :: HH-CPU
- **Implementation** :: Notes
- **Results** :: Datasets (SNAP, Synthetic …), Experiments & Discussion
- **Other Approaches** :: Work Queue & its variations
- **Conclusion** :: Future Work & References

Heterogeneous Platform

CPU

GPU

Send Code

Send Data

Send Results

Heterogeneous Platform

CPU

GPU

Send Code

Send Data

Data Transfer

Data Transfer

Data Transfer

Data Transfer

Send Results

Challenges

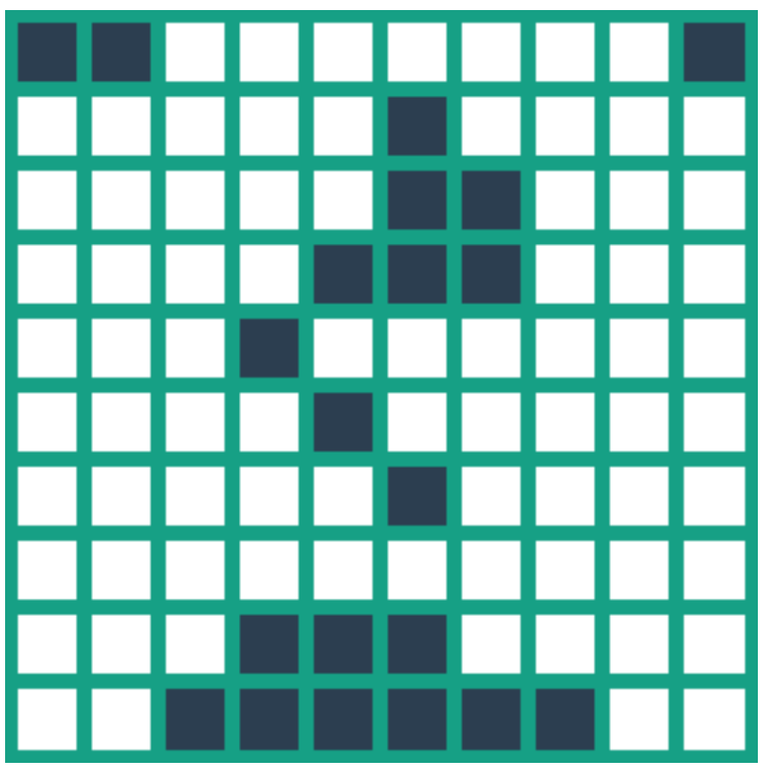
- Which portion of input is processed by which device ?
- Static Partitioning input is a good solution to obtain high performance on heterogeneous platforms.
- However, compute capability of each entity is different & performance of device is dependent on nature of input.
- Simple/Static partitioning is not optimal.
- Is it possible to come up with partitioning techniques for heterogenous platforms and applications ?

Our Goal

- To propose a novel heterogeneous algorithm for sparse matrix-matrix multiplication that,
  - not only, balances load across heterogeneous devices in computing platform.
  - but also, assigns "right" work to the "right" processor.

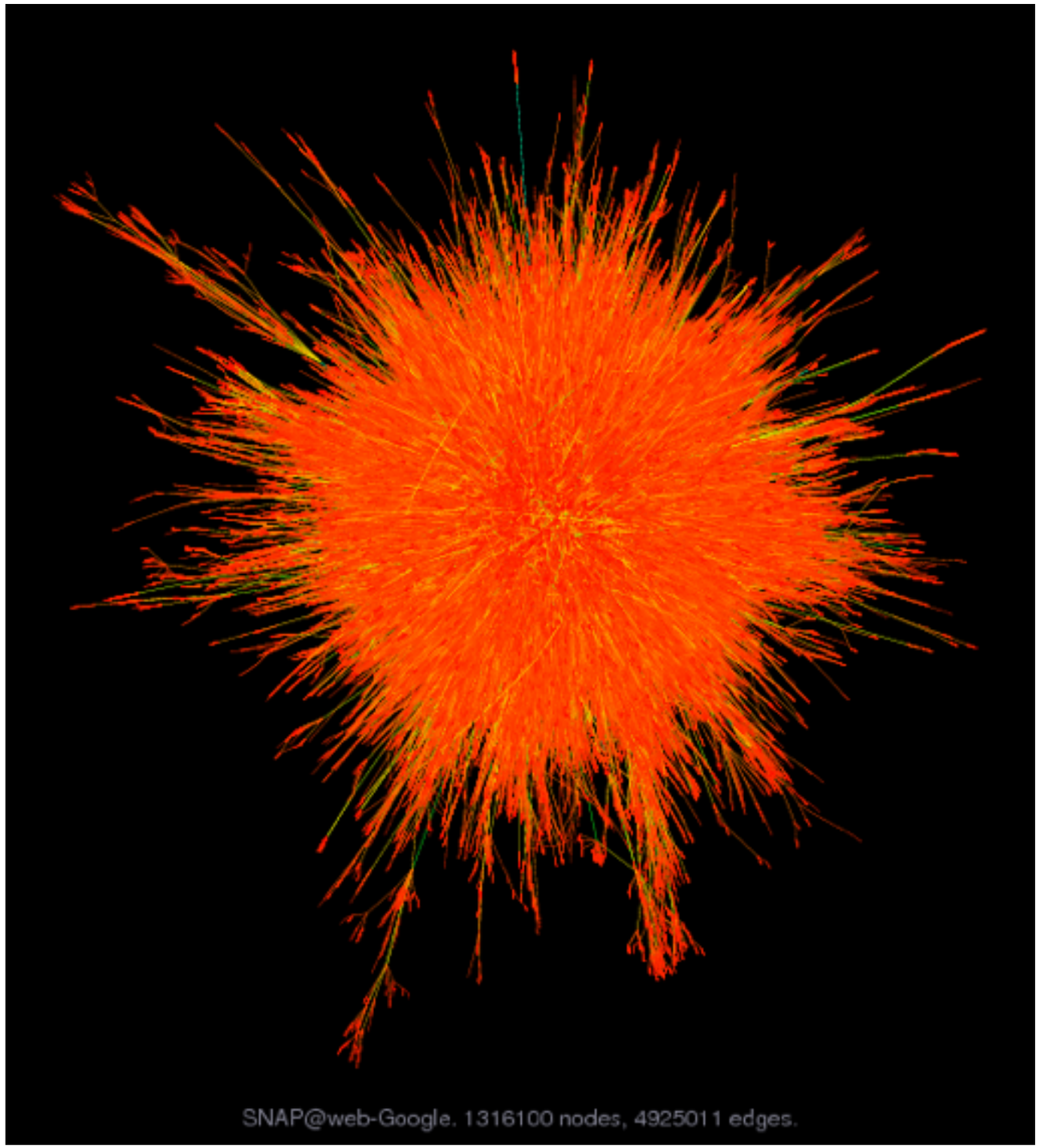
Sparse Matrix

- Matrix in which most of the elements are zero.
- i.e. nnz = k * n
- Example

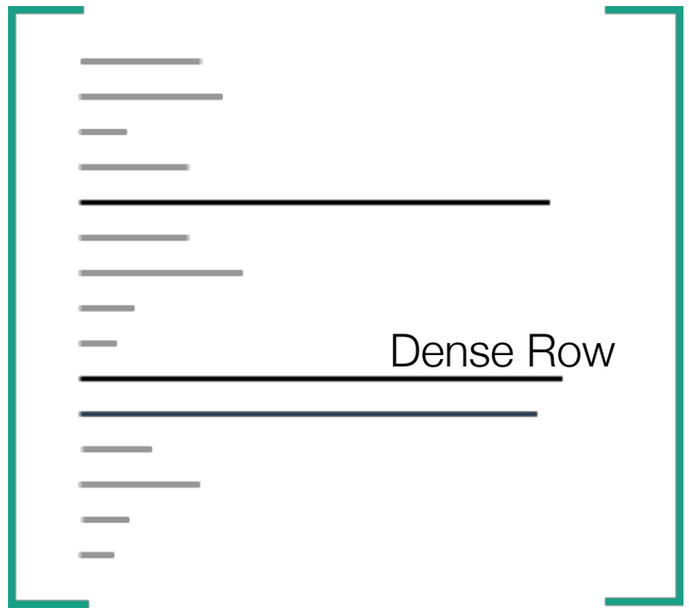


Real-World Matrices

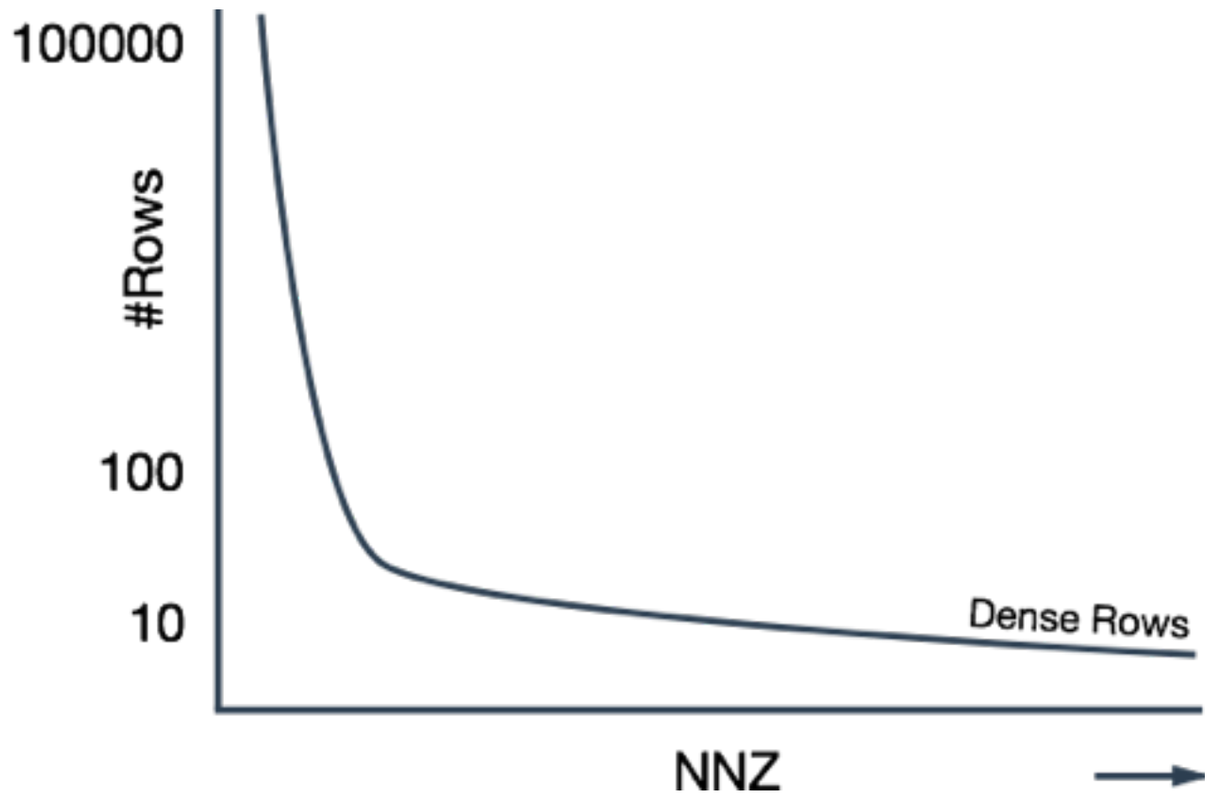
Usually datasets in Data Mining, Social Network Analysis & Communication Networks are very large.



Nature of Real-world Matrices



These graphs are highly irregular & scale-free with a power-law degree distribution.



Sparse Matrix-Matrix Multiplication

- Compute C = A x B, where A, B are two sparse matrices.
- Why is it hard in a heterogeneous setting ?
  - Sparse nature of matrix makes it hard for programmers to exploit CPU's cache hierarchy (tiling) to achieve performance.
  - Irregular computation implies thread load imbalance & hence not suitable for GPUs.

Row-Row Formulation

- K. Matam et. al, proved row-row formulation of matrix multiplication out performs usual row-column formulation for SPMM in GPUs.

$$C(i,:) = \sum_{j \in I_i(A)} A(i,j) * B(j,:)$$

$$A = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 4 \end{bmatrix} \end{array} \quad B = \begin{array}{c} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \end{array} \begin{bmatrix} 2 & 3 & 4 \\ 8 & 0 & 0 \\ 0 & 0 & 6 \\ 0 & 7 & 0 \end{bmatrix} \quad A \times B = \begin{bmatrix} 16 & 0 & 6 \\ 0 & 7 & 6 \\ 2 & 3 & 10 \\ 4 & 34 & 8 \end{bmatrix}$$

C(1, :) =

Row-Row Formulation

Example

$$A = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 4 \end{bmatrix} \end{array} \quad B = \begin{array}{c} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \end{array} \begin{bmatrix} 2 & 3 & 4 \\ 8 & 0 & 0 \\ 0 & 0 & 6 \\ 0 & 7 & 0 \end{bmatrix} \quad A \times B = \begin{bmatrix} 16 & 0 & 6 \\ 0 & 7 & 6 \\ 2 & 3 & 10 \\ 4 & 34 & 8 \end{bmatrix}$$

C(1, :) = **2** *

Row-Row Formulation

Example

$$A = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 4 \end{bmatrix} \end{array} \quad B = \begin{array}{c} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \end{array} \begin{bmatrix} 2 & 3 & 4 \\ 8 & 0 & 0 \\ 0 & 0 & 6 \\ 0 & 7 & 0 \end{bmatrix} \quad A \times B = \begin{bmatrix} 16 & 0 & 6 \\ 0 & 7 & 6 \\ 2 & 3 & 10 \\ 4 & 34 & 8 \end{bmatrix}$$

C(1, :) = **2** * [8 0 0]

Row-Row Formulation

Example

$$A = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 4 \end{bmatrix} \end{array} \quad B = \begin{array}{c} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \end{array} \begin{bmatrix} 2 & 3 & 4 \\ 8 & 0 & 0 \\ 0 & 0 & 6 \\ 0 & 7 & 0 \end{bmatrix} \quad A \times B = \begin{bmatrix} 16 & 0 & 6 \\ 0 & 7 & 6 \\ 2 & 3 & 10 \\ 4 & 34 & 8 \end{bmatrix}$$

C(1, :) = **2** * [8 0 0] + **1** *

Row-Row Formulation

Example

$$A = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 4 \end{bmatrix} \end{array} \quad B = \begin{array}{c} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \end{array} \begin{bmatrix} 2 & 3 & 4 \\ 8 & 0 & 0 \\ 0 & 0 & 6 \\ 0 & 7 & 0 \end{bmatrix} \quad A \times B = \begin{bmatrix} 16 & 0 & 6 \\ 0 & 7 & 6 \\ 2 & 3 & 10 \\ 4 & 34 & 8 \end{bmatrix}$$

$C(1, :) = \mathbf{2} * [8 \;\; 0 \;\; 0] + \mathbf{1} * [0 \;\; 0 \;\; 6]$

Row-Row Formulation

Example

$$A = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 4 \end{bmatrix} \end{array} \quad B = \begin{array}{c} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \end{array} \begin{bmatrix} 2 & 3 & 4 \\ 8 & 0 & 0 \\ 0 & 0 & 6 \\ 0 & 7 & 0 \end{bmatrix} \quad A \times B = \begin{bmatrix} 16 & 0 & 6 \\ 0 & 7 & 6 \\ 2 & 3 & 10 \\ 4 & 34 & 8 \end{bmatrix}$$

$$C(1, :) = \mathbf{2} * [8 \;\; 0 \;\; 0] + \mathbf{1} * [0 \;\; 0 \;\; 6] = [16 \;\; 0 \;\; 6]$$

Row-Row Formulation

Example

$$A = \begin{matrix} & 0 & 1 & 2 & 3 \\ & \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 4 \end{bmatrix} \end{matrix} \quad B = \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} 2 & 3 & 4 \\ 8 & 0 & 0 \\ 0 & 0 & 6 \\ 0 & 7 & 0 \end{bmatrix} \quad A \times B = \begin{bmatrix} 16 & 0 & 6 \\ 0 & 7 & 6 \\ 2 & 3 & 10 \\ 4 & 34 & 8 \end{bmatrix}$$

$C(1, :) = \mathbf{2} * [8\ \ 0\ \ 0] + \mathbf{1} * [0\ \ 0\ \ 6] = [16\ \ 0\ \ 6]$

$C(2, :) = \mathbf{1} * [0\ \ 0\ \ 6] + \mathbf{1} * [0\ \ 7\ \ 0] = [0\ \ 7\ \ 6]$

Row-Row Formulation

Example

$$A = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 4 \end{bmatrix} \end{array} \quad B = \begin{array}{c} \begin{matrix} \\ 0 \\ 1 \\ 2 \\ 3 \end{matrix} \\ \begin{bmatrix} 2 & 3 & 4 \\ 8 & 0 & 0 \\ 0 & 0 & 6 \\ 0 & 7 & 0 \end{bmatrix} \end{array} \quad A \times B = \begin{bmatrix} 16 & 0 & 6 \\ 0 & 7 & 6 \\ 2 & 3 & 10 \\ 4 & 34 & 8 \end{bmatrix}$$

$C(1, :) = \mathbf{2} * [8\ \ 0\ \ 0] + \mathbf{1} * [0\ \ 0\ \ 6] = [16\ \ 0\ \ 6]$

$C(2, :) = \mathbf{1} * [0\ \ 0\ \ 6] + \mathbf{1} * [0\ \ 7\ \ 0] = [0\ \ 7\ \ 6]$

$C(3, :) = \mathbf{1} * [2\ \ 3\ \ 4] + \mathbf{1} * [0\ \ 0\ \ 6] = [2\ \ 3\ \ 10]$

Row-Row Formulation

Example

$$A = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 4 \end{bmatrix} \end{array} \quad B = \begin{array}{c} \begin{matrix} \\ 0 \\ 1 \\ 2 \\ 3 \end{matrix} \\ \begin{bmatrix} 2 & 3 & 4 \\ 8 & 0 & 0 \\ 0 & 0 & 6 \\ 0 & 7 & 0 \end{bmatrix} \end{array} \quad A \times B = \begin{bmatrix} 16 & 0 & 6 \\ 0 & 7 & 6 \\ 2 & 3 & 10 \\ 4 & 34 & 8 \end{bmatrix}$$

$C(1, :) = \mathbf{2} * [8\ \ 0\ \ 0] + \mathbf{1} * [0\ \ 0\ \ 6] = [16\ \ 0\ \ 6]$

$C(2, :) = \mathbf{1} * [0\ \ 0\ \ 6] + \mathbf{1} * [0\ \ 7\ \ 0] = [0\ \ 7\ \ 6]$

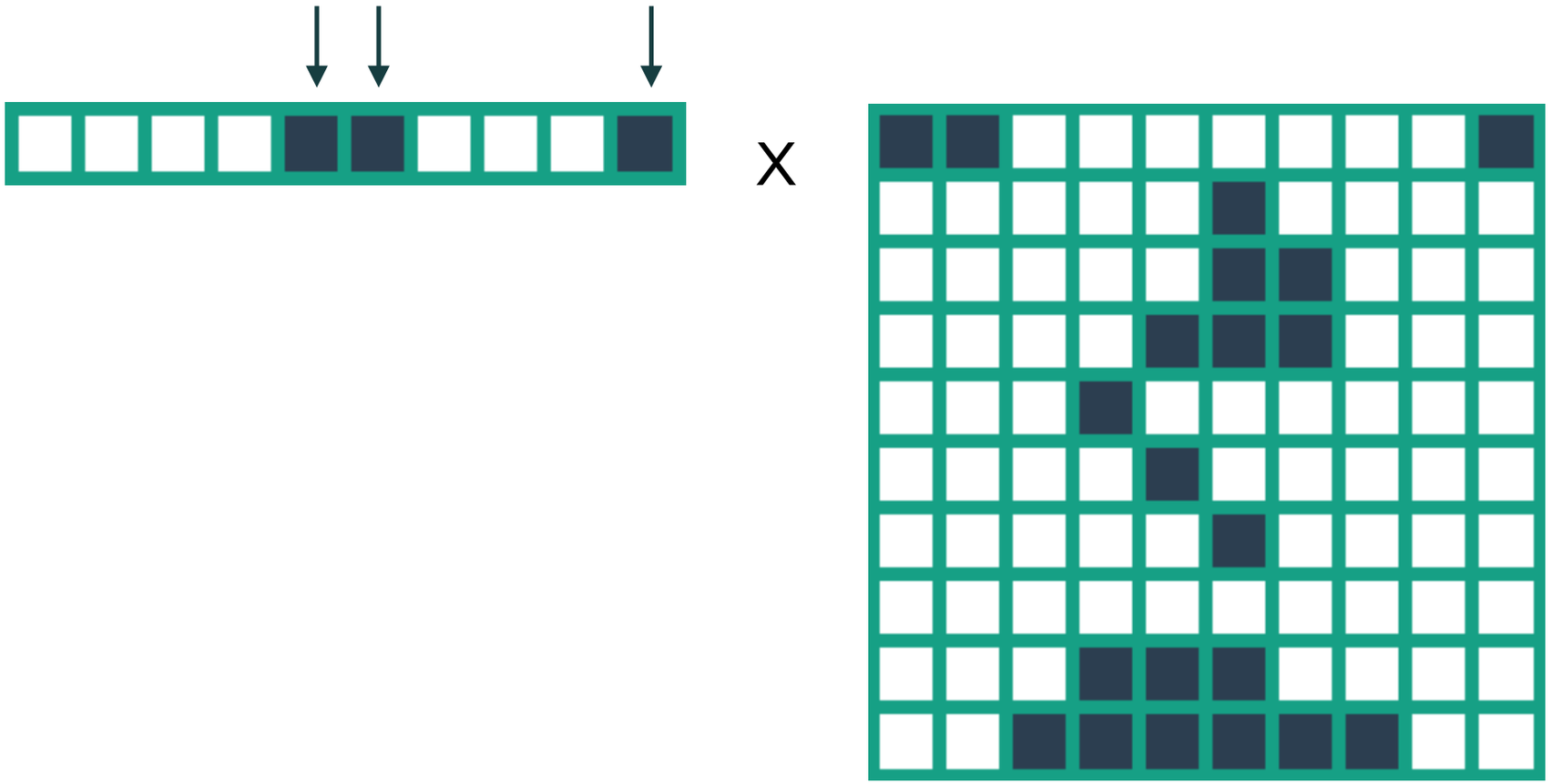
$C(3, :) = \mathbf{1} * [2\ \ 3\ \ 4] + \mathbf{1} * [0\ \ 0\ \ 6] = [2\ \ 3\ \ 10]$

$C(4, :) = \mathbf{2} * [2\ \ 3\ \ 4] + \mathbf{4} * [0\ \ 7\ \ 0] = [4\ \ 34\ \ 8]$

Row-Row Formulation

Example

Thread Load Imbalance



HH-CPU

- Classify each row of sparse matrix into high dense and low dense. Now we can write SPMM as,

$$C = A \times B$$

$$\Rightarrow C = (A_H + A_L) \times (B_H + B_L)$$

$$\Rightarrow C = A_H \times B_H + A_L \times B_L + A_H \times B_L + A_L \times B_H$$

- Each multiplication above has certain properties that helps us to map it to a device that performs better.

Example

$$A = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 4 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 6 & 12 & 7 & 7 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

Example

$$A = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 4 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 6 & 12 & 7 & 7 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

$$A_H = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 0 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B_H = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 0 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad A_H \times B_H = \begin{bmatrix} 3 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 \\ 6 & 10 & 7 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Example

$$A = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 4 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 6 & 12 & 7 & 7 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

$A_H \times B_H$

$$\begin{bmatrix} 3 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 \\ 6 & 10 & 7 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix} +$$

Example

$$A = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 4 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 6 & 12 & 7 & 7 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

$A_H \times B_H$

$$\begin{bmatrix} 3 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 \\ 6 & 10 & 7 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix} +$$

$$A_L = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad B_L = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad A_L \times B_L = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

Example

$$A = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \qquad B = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \qquad C = \begin{bmatrix} 3 & 4 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 6 & 12 & 7 & 7 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

$A_H \times B_H$ $A_L \times B_L$

$$\begin{bmatrix} 3 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 \\ 6 & 10 & 7 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix} +$$

Example

$$A = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 4 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 6 & 12 & 7 & 7 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

$A_H \times B_H$ $A_L \times B_L$

$$\begin{bmatrix} 3 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 \\ 6 & 10 & 7 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix} +$$

$$A_H = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 0 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B_L = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad A_H \times B_L = \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 5 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Example

$$A = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 4 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 6 & 12 & 7 & 7 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

$A_H \times B_H$ $A_L \times B_L$ $A_H \times B_L$

$$\begin{bmatrix} 3 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 \\ 6 & 10 & 7 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 5 \\ 0 & 0 & 0 & 0 \end{bmatrix} +$$

Example

$$A = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 4 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 6 & 12 & 7 & 7 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

$A_H \times B_H$ $A_L \times B_L$ $A_H \times B_L$

$$\begin{bmatrix} 3 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 \\ 6 & 10 & 7 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 5 \\ 0 & 0 & 0 & 0 \end{bmatrix} +$$

$$A_L = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad B_H = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 0 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad A_L \times B_H = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Example

$$A = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 4 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 6 & 12 & 7 & 7 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

$A_H \times B_H$ $A_L \times B_L$ $A_H \times B_L$ $A_L \times B_H$

$$\begin{bmatrix} 3 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 \\ 6 & 10 & 7 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 5 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} =$$

Example

$$A = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad B = \begin{bmatrix} \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{3} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 0 & 0 & 0 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 4 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 6 & 12 & 7 & 7 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

$A_H \times B_H$ $A_L \times B_L$ $A_H \times B_L$ $A_L \times B_H$

$$\begin{bmatrix} 3 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 \\ 6 & 10 & 7 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 5 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 4 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 6 & 12 & 7 & 7 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

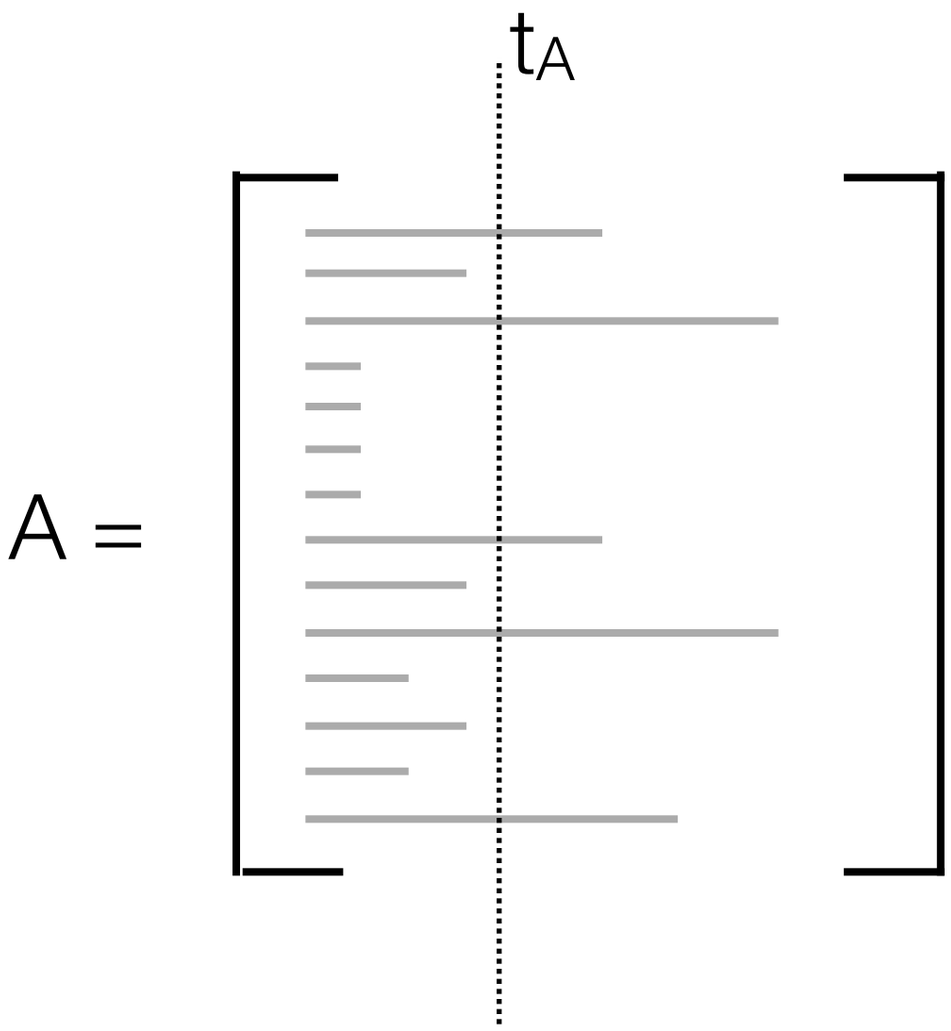
Phase I

- ***CPU, GPU ::*** *Identify thresholds t_A, t_B and the matrices A_H, A_L, B_H, B_L.*

A =

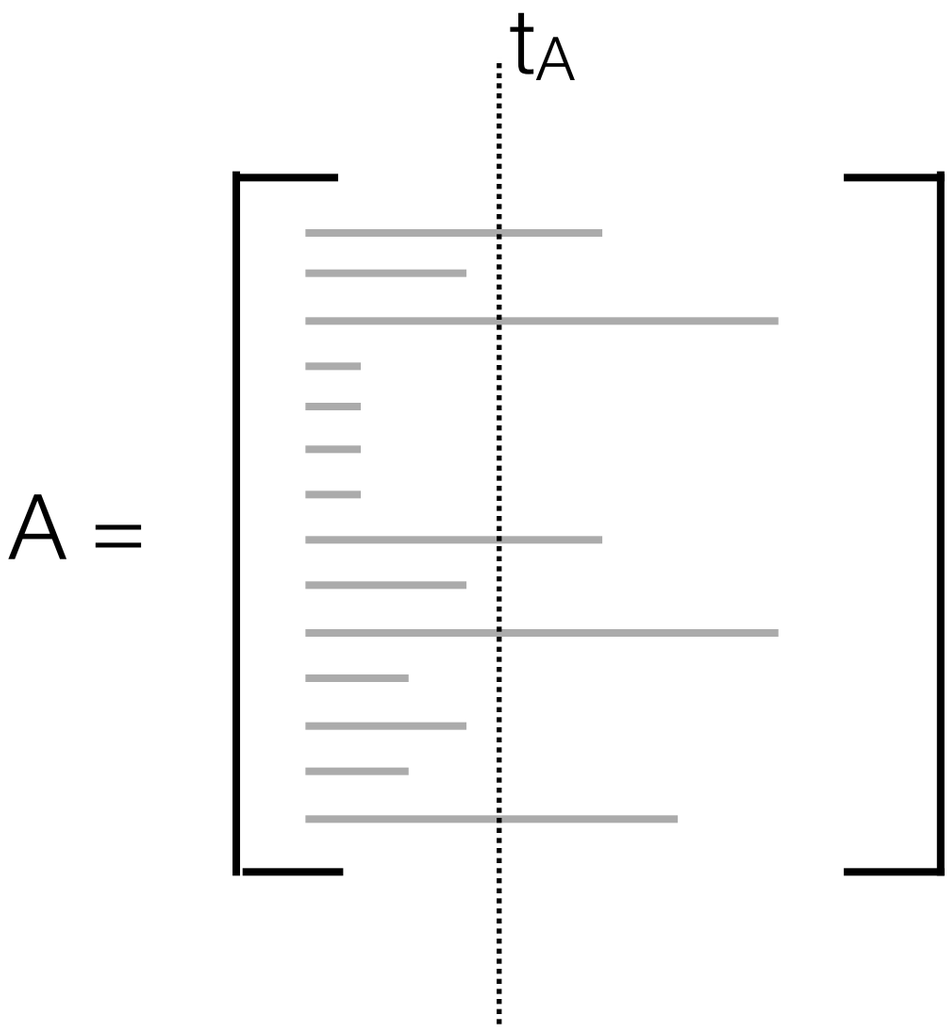
Phase I

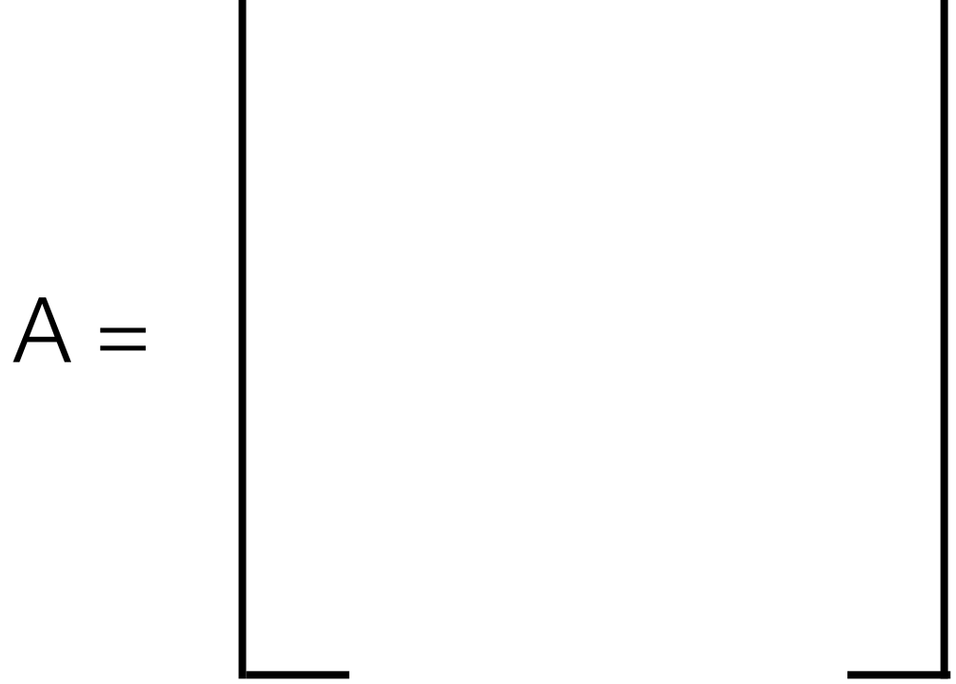
- ***CPU, GPU ::*** *Identify thresholds t_A, t_B and the matrices A_H, A_L, B_H, B_L.*



Phase I

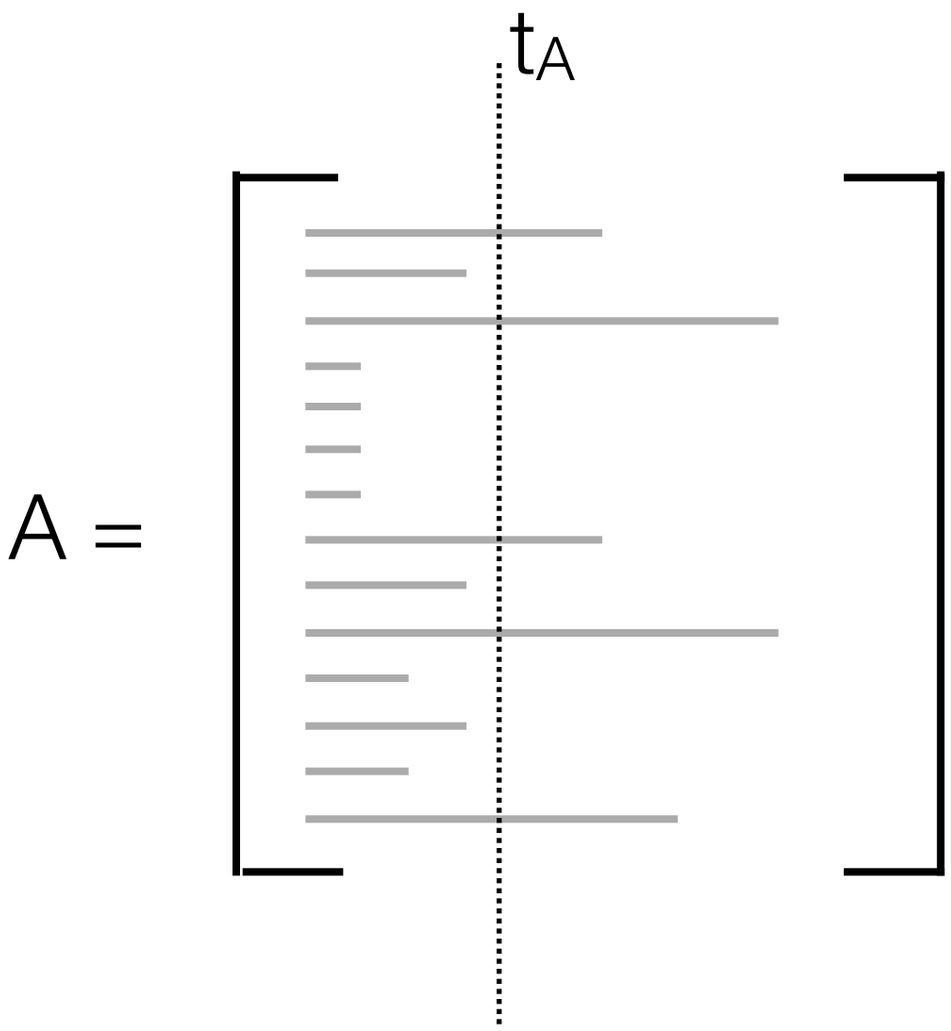
- ***CPU, GPU ::*** *Identify thresholds t_A, t_B and the matrices A_H, A_L, B_H, B_L.*

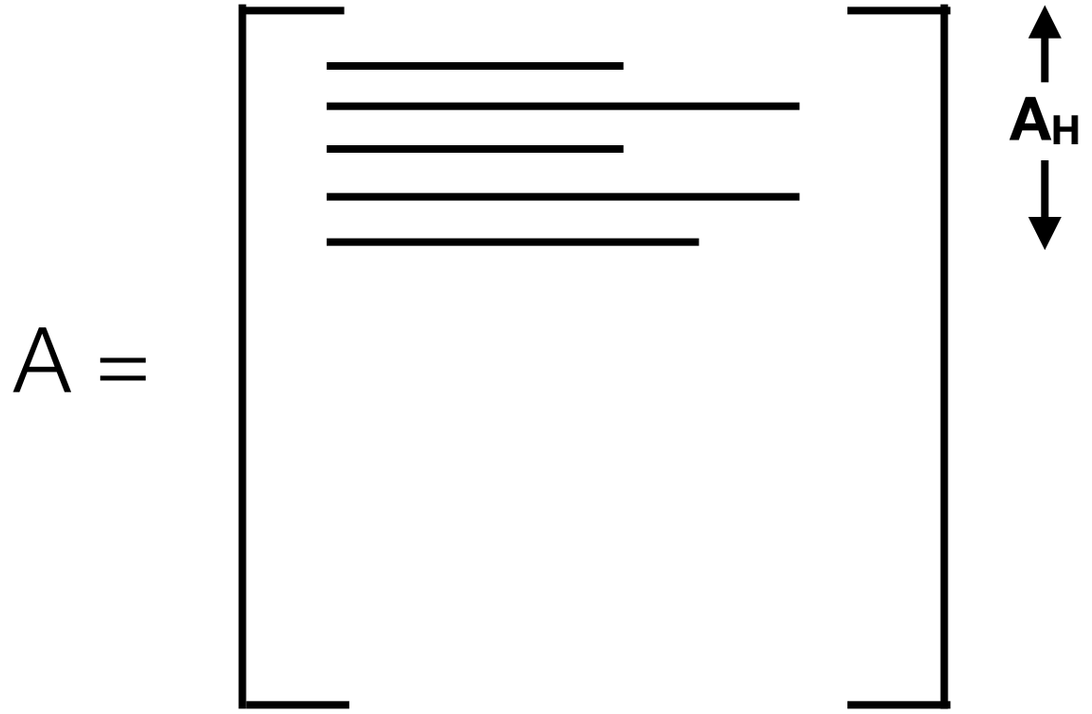




Phase I

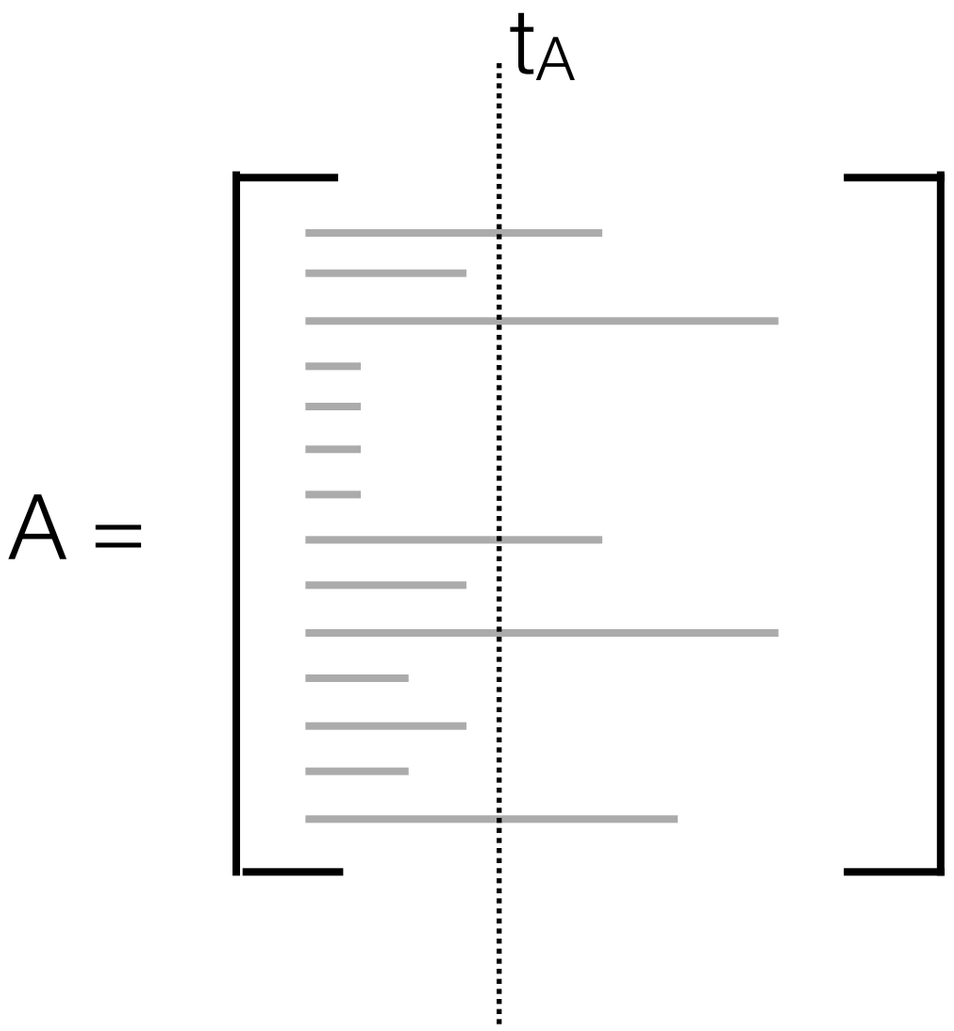
- ***CPU, GPU ::*** *Identify thresholds t_A, t_B and the matrices A_H, A_L, B_H, B_L.*





Phase I

- ***CPU, GPU ::*** *Identify thresholds t_A, t_B and the matrices A_H, A_L, B_H, B_L.*



Phase II

- *In parallel,*

  CPU :: *Compute* $A_H * B_H$.

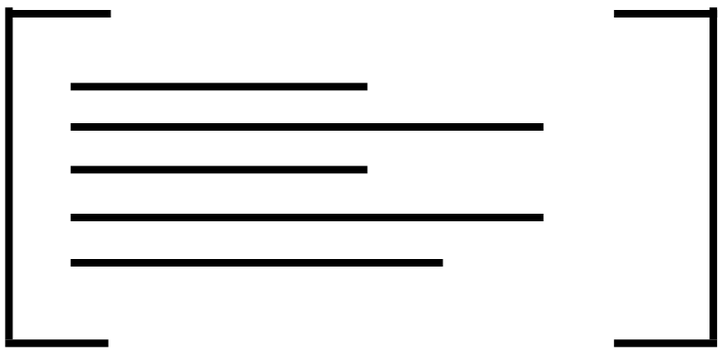
  GPU :: *Compute* $A_L * B_L$.

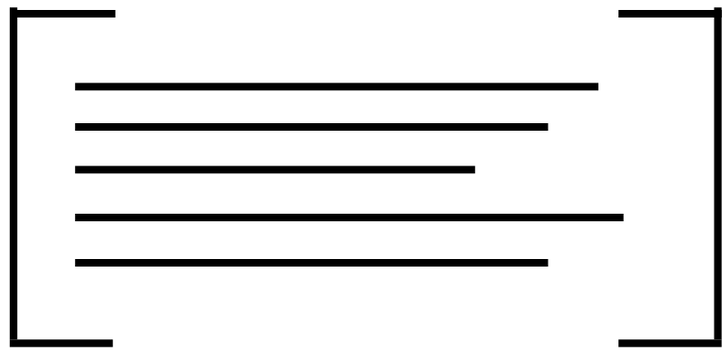
Phase II

- *In parallel,*

  CPU :: *Compute* $A_H * B_H$.

  GPU :: *Compute* $A_L * B_L$.

$A_H =$ 

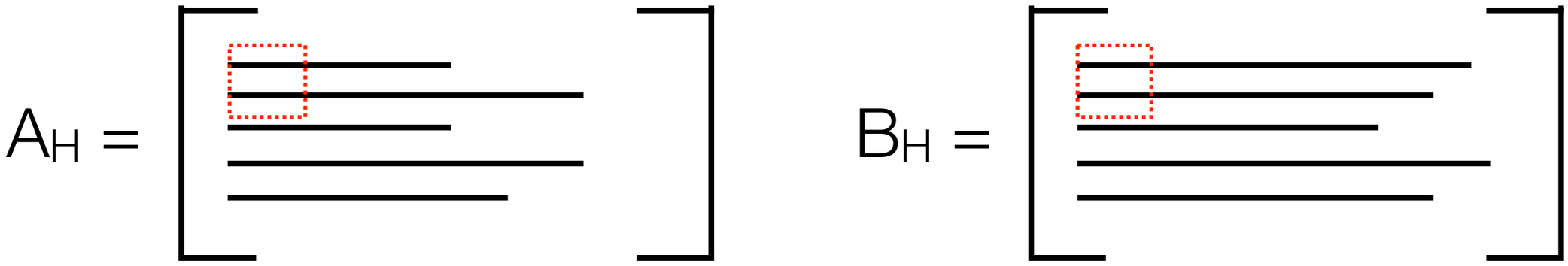
$B_H =$ 

Phase II

- *In parallel,*

  CPU :: *Compute* $A_H * B_H$.

  GPU :: *Compute* $A_L * B_L$.

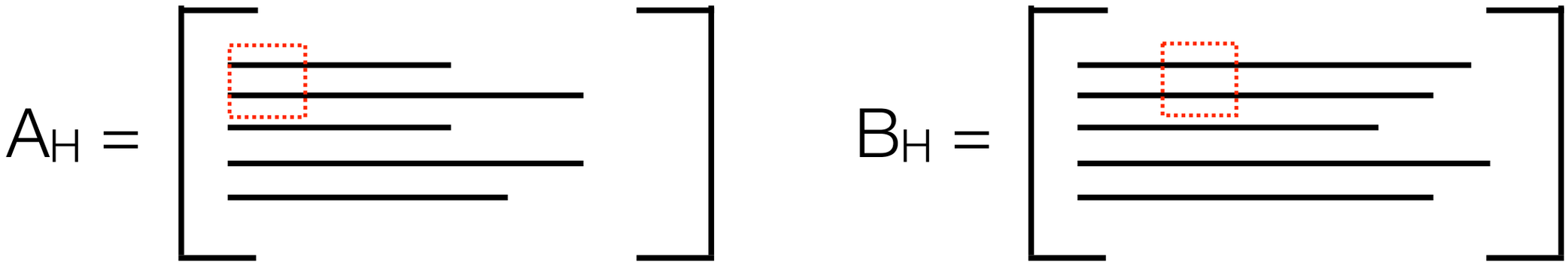


Phase II

- *In parallel,*

  CPU :: *Compute* $A_H * B_H$.

  GPU :: *Compute* $A_L * B_L$.

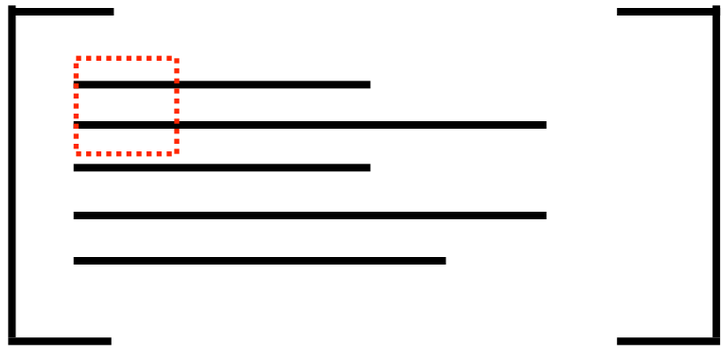


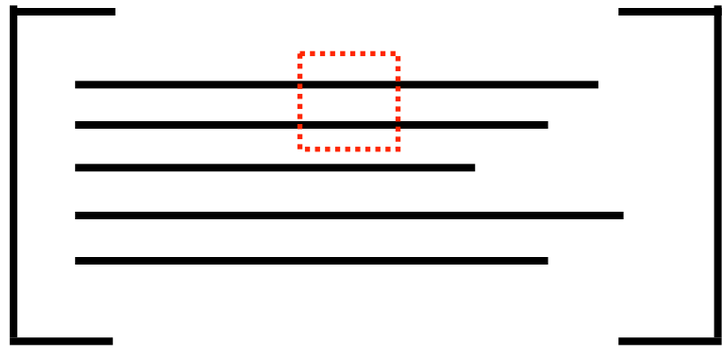
Phase II

- *In parallel,*

  CPU :: *Compute* $A_H * B_H$.

  GPU :: *Compute* $A_L * B_L$.

$A_H =$ 

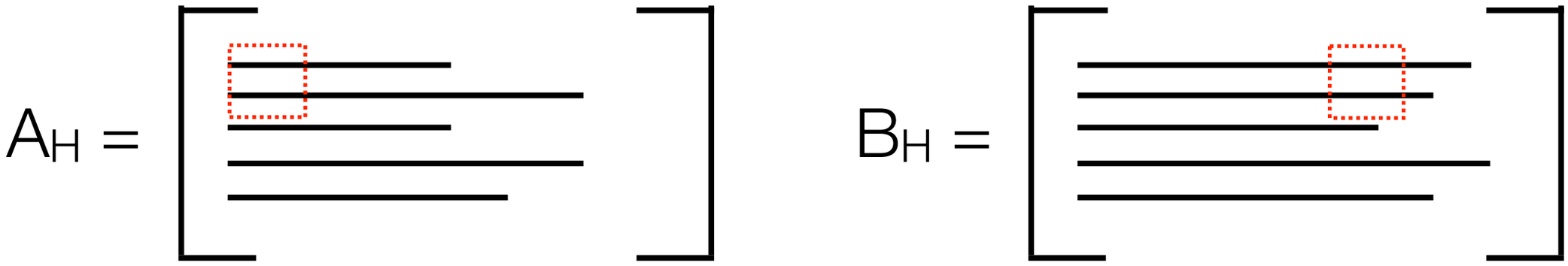
$B_H =$ 

Phase II

- *In parallel,*

  CPU :: *Compute* $A_H * B_H$.

  GPU :: *Compute* $A_L * B_L$.

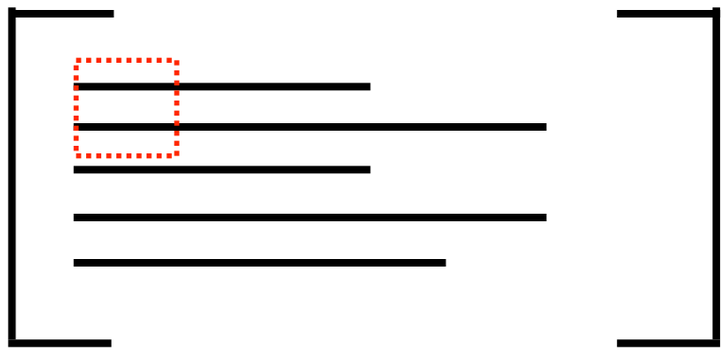


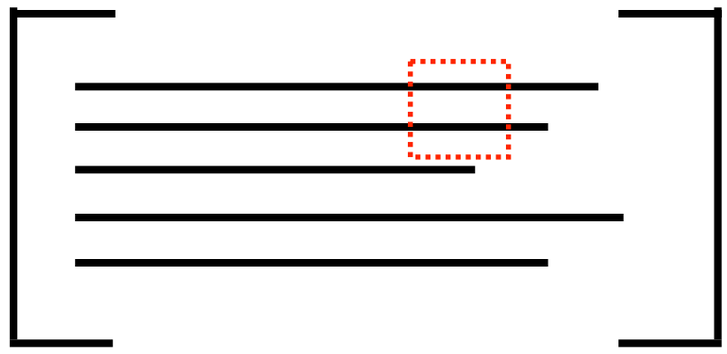
Phase II

- *In parallel,*

  CPU :: *Compute* $A_H * B_H$.

  GPU :: *Compute* $A_L * B_L$.

$A_H =$ 

$B_H =$ 

$A_L =$ 

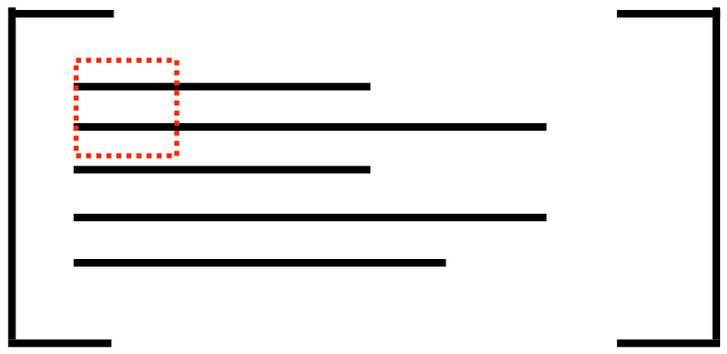
$B_L =$ 

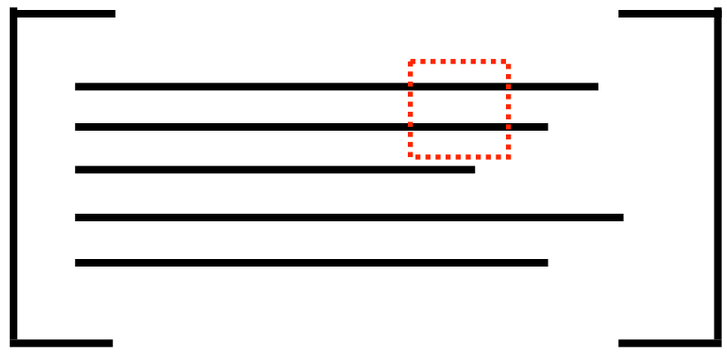
Phase II

- *In parallel,*

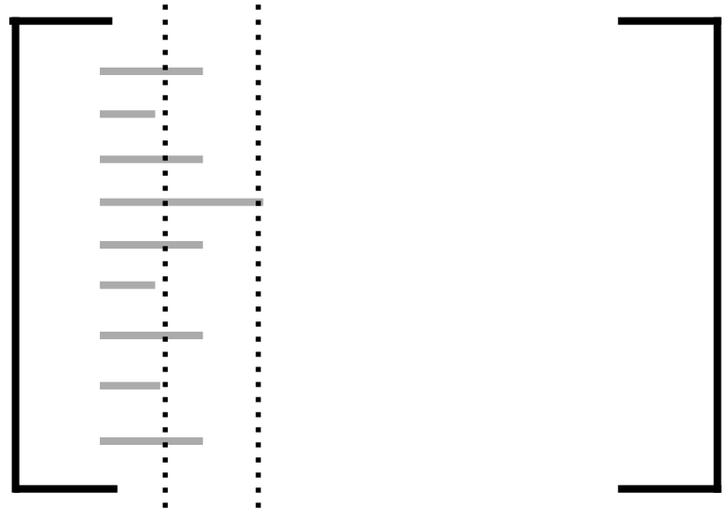
  CPU :: *Compute* $A_H * B_H$.

  GPU :: *Compute* $A_L * B_L$.

$A_H =$ 

$B_H =$ 

$A_L =$ 

$B_L =$ 

Phase III

- *In parallel,*

  CPU :: *Compute* $A_L * B_H$. *[WorkQueue Mode]*

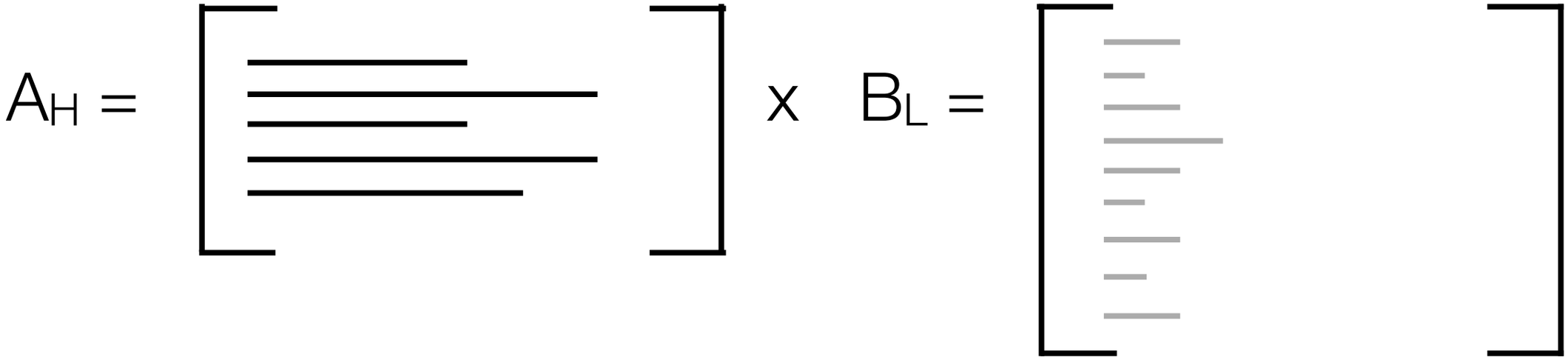
  GPU :: *Compute* $A_H * B_L$.

Phase III

- *In parallel,*

  CPU :: *Compute $A_L * B_H$. [WorkQueue Mode]*

  GPU :: *Compute $A_H * B_L$.*

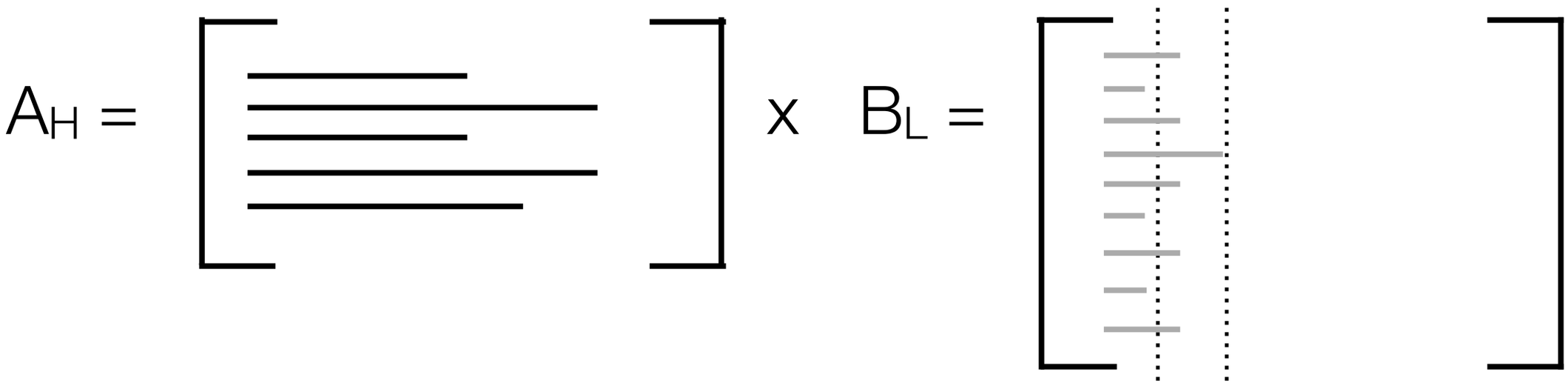


Phase III

- *In parallel,*

  CPU :: *Compute* $A_L * B_H$. *[WorkQueue Mode]*

  GPU :: *Compute* $A_H * B_L$.

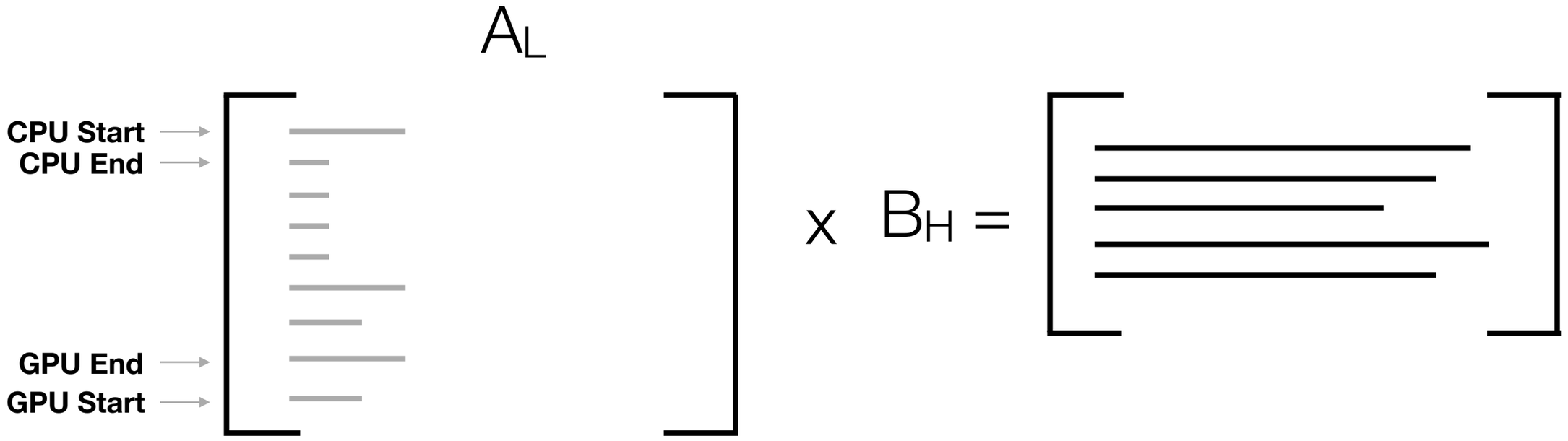


Phase III :: Contd

- *In parallel,*

  CPU :: *Compute $A_L * B_H$. [WorkQueue Mode]*

  GPU :: *Compute $A_H * B_L$.*

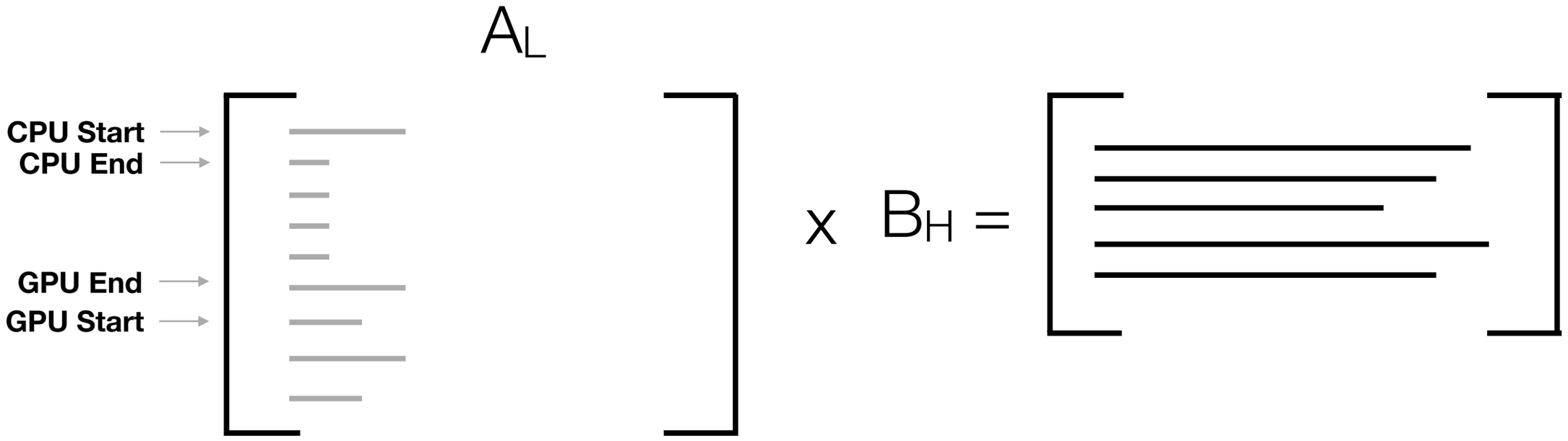


Phase III :: Contd

- *In parallel,*

  CPU :: *Compute $A_L * B_H$. [WorkQueue Mode]*

  GPU :: *Compute $A_H * B_L$.*

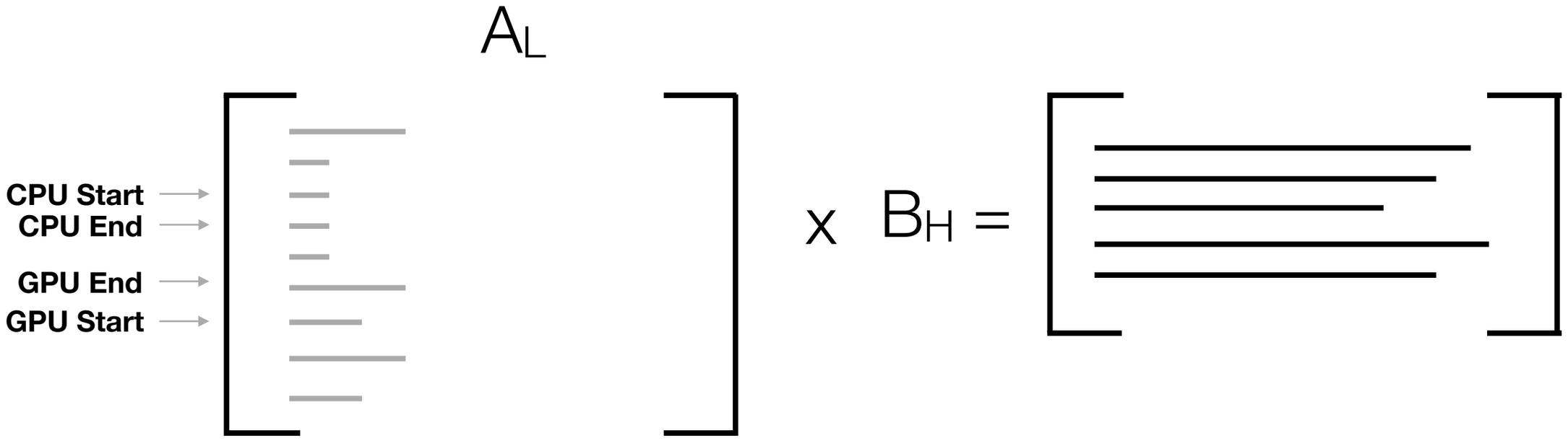


Phase III :: Contd

- *In parallel,*

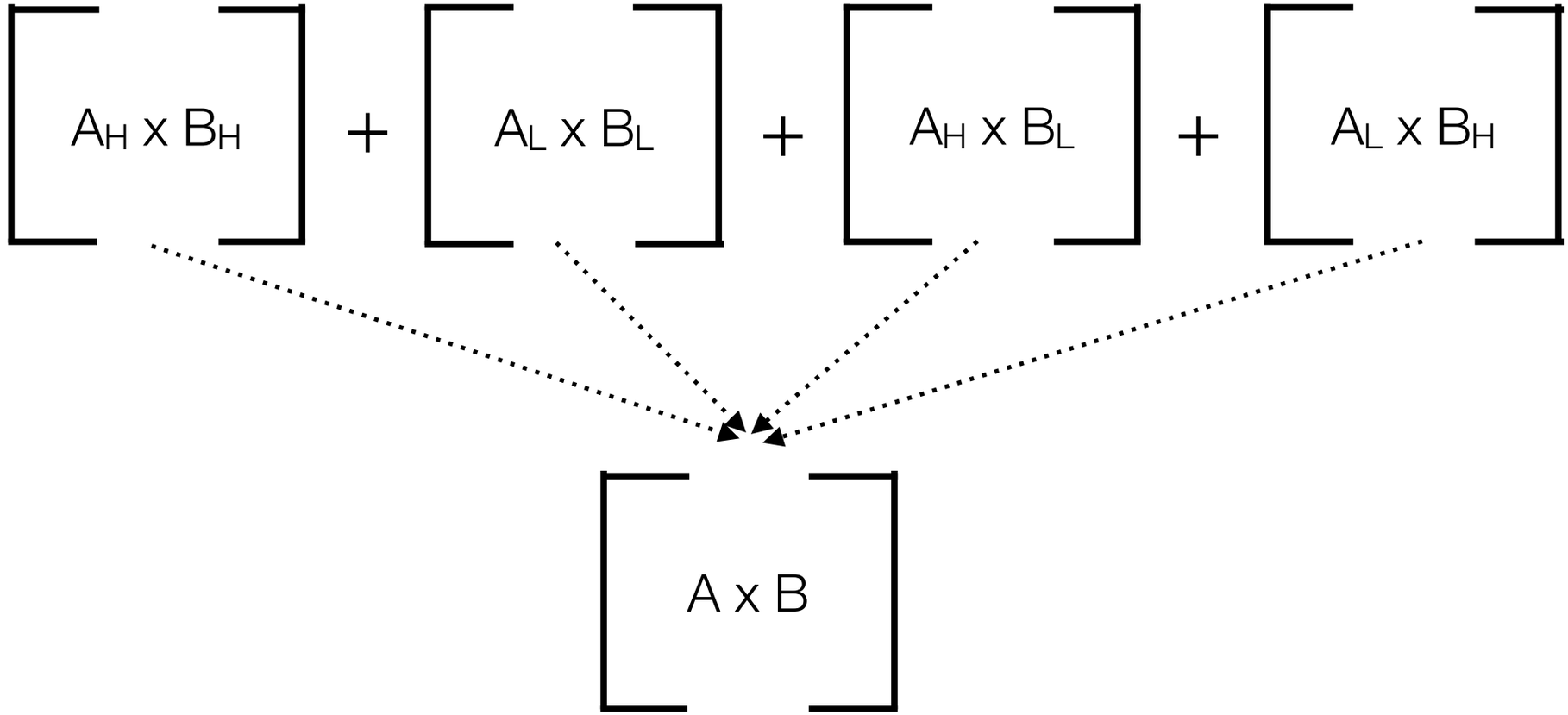
  CPU :: *Compute $A_L * B_H$. [WorkQueue Mode]*

  GPU :: *Compute $A_H * B_L$.*

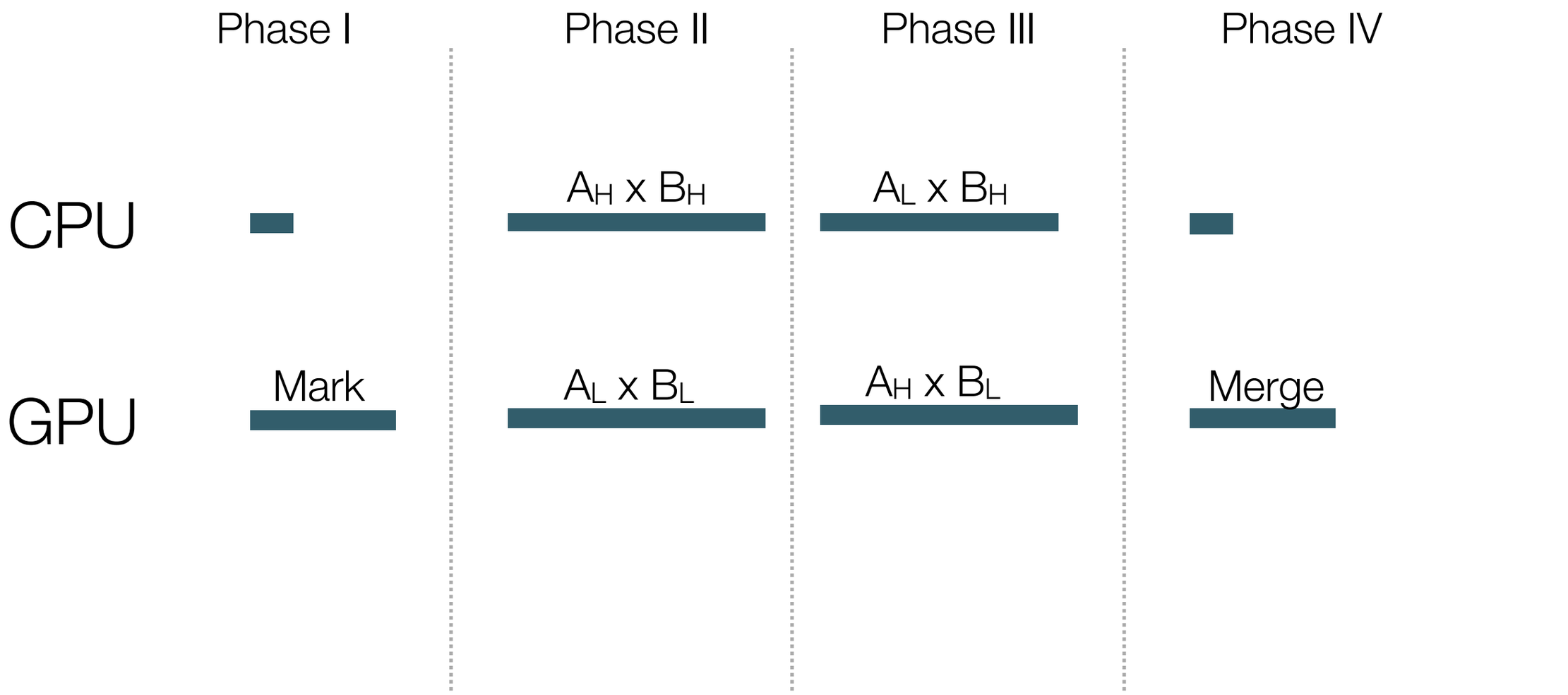


Phase IV

- ***CPU, GPU ::*** *Combine results of Phases II & III.*
- ***GPU to CPU ::*** *Transfer the partial results from GPU to CPU.*



Timeline Diagram



Implementation Details

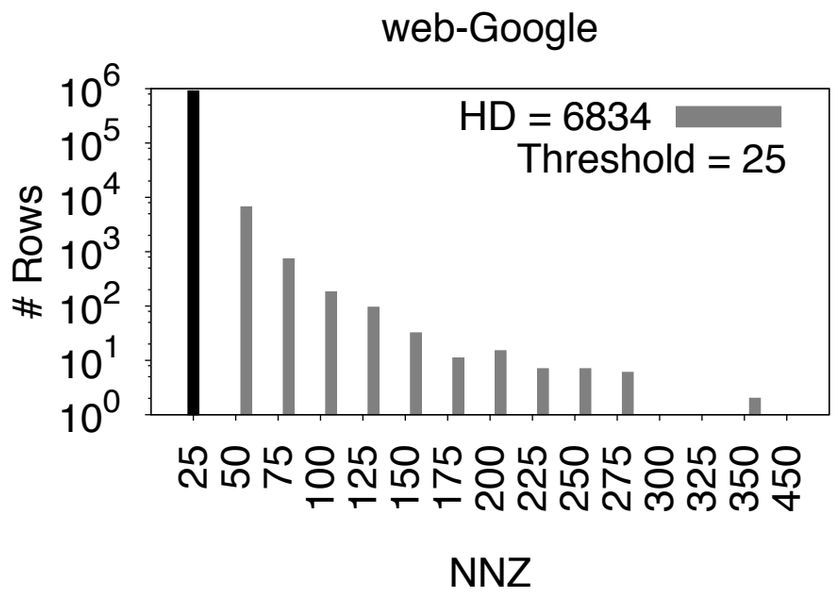
- Sparse matrices A, B are stored in CSR format.
- We multiply A x A instead of A x B due to dataset unavailability. However we show results for A x B in synthetically generated data for experiments.
- We consider only CPU & GPU for simplicity in the heterogeneous system.
- Phase 1 :: Thresholds (t_A, t_B) are empirically obtained.
- Phase 2, 3 :: We use modified version of Row_Row_SPMM developed by K. Matam et. al for SPMM of partial matrices.
- Phase 3 :: Work units of CPU & GPU in work queue model is empirically determined a-priori.
- Phase 4 :: Standard primitives like Mark, Scan & Merge are used.

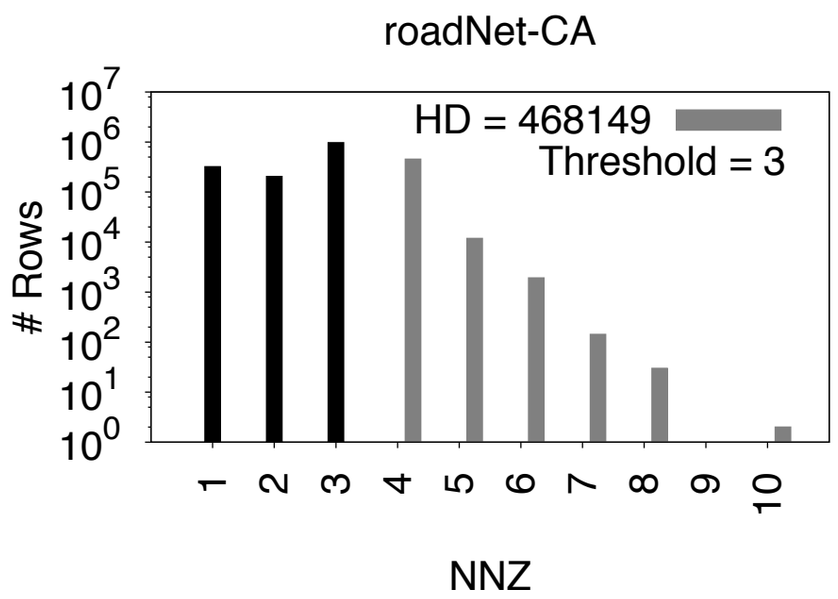
Experimental Setup

- CPU :: Intel i7 980
- GPU :: Nvidia Tesla K20c (Kepler)
- CPU - GPU Link :: PCI Express version 2.0 link that supports data transfer bandwidth of 8 GB/s.
- CUDA API Version 4.1 for Programming

Dataset (Scale-free & α)

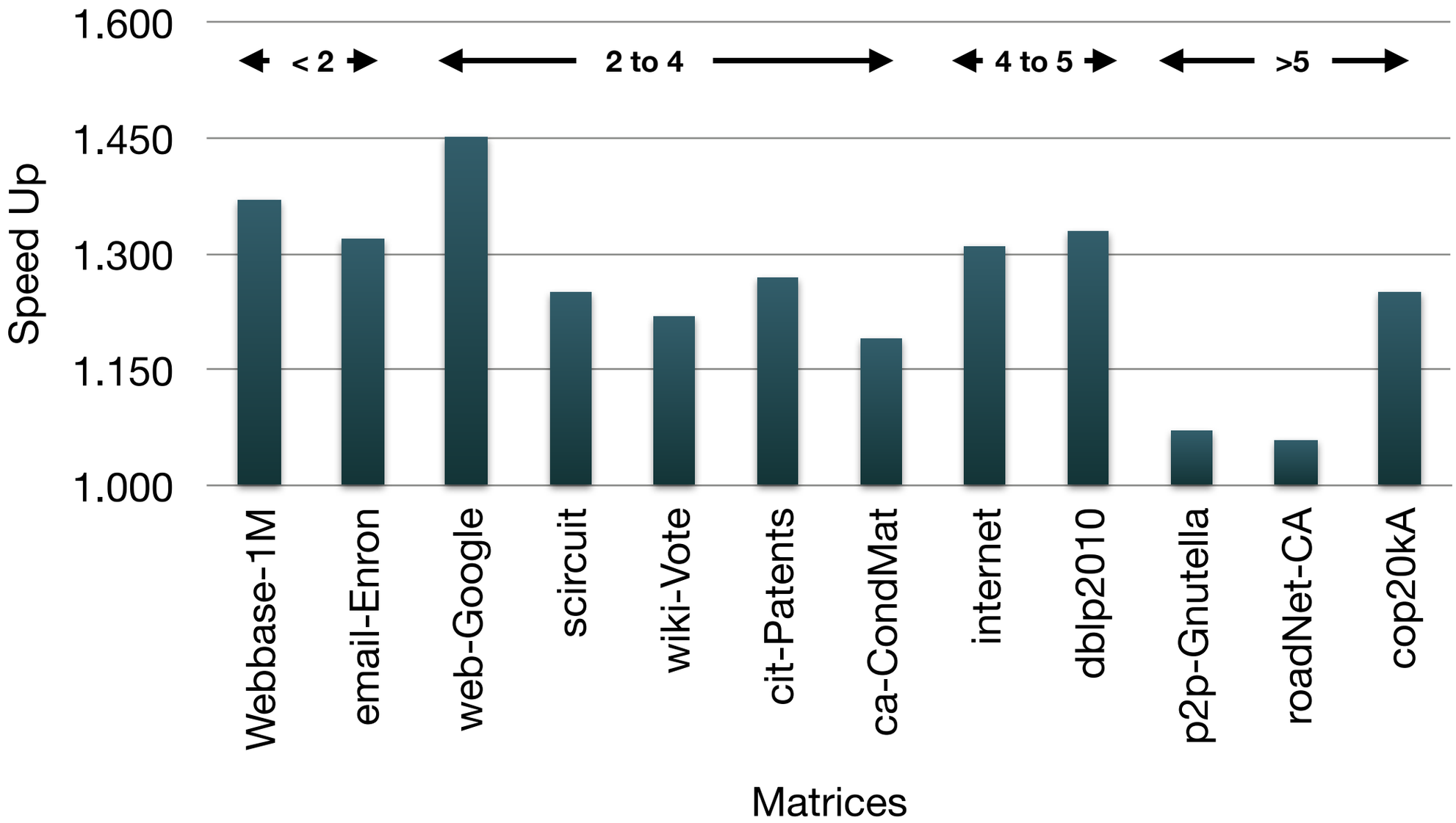
Matrix	Rows	NNZ	α
scircuit	1,70,998	9,58,936	3.55
Webbase-1M	10,00,005	31,05,536	2.1
cop20kA	1,21,192	26,24,331	143.8
web-Google	**9,16,428**	**51,05,039**	**3.75**
p2p-Gnutella	62,586	1,47,892	48.9
ca-CondMat	23,133	1,86,936	3.58
roadNet-CA	**19,71,281**	**55,33,214**	**133.80**
internet	1,24,651	2,07,214	4.63
dblp2010	3,26,186	16,15,400	5.79
email-Enron	36,692	3,67,662	2.1
wiki-Vote	8,297	1,03,689	3.88
cit-Patents	37,74,768	1,65,18,948	3.90



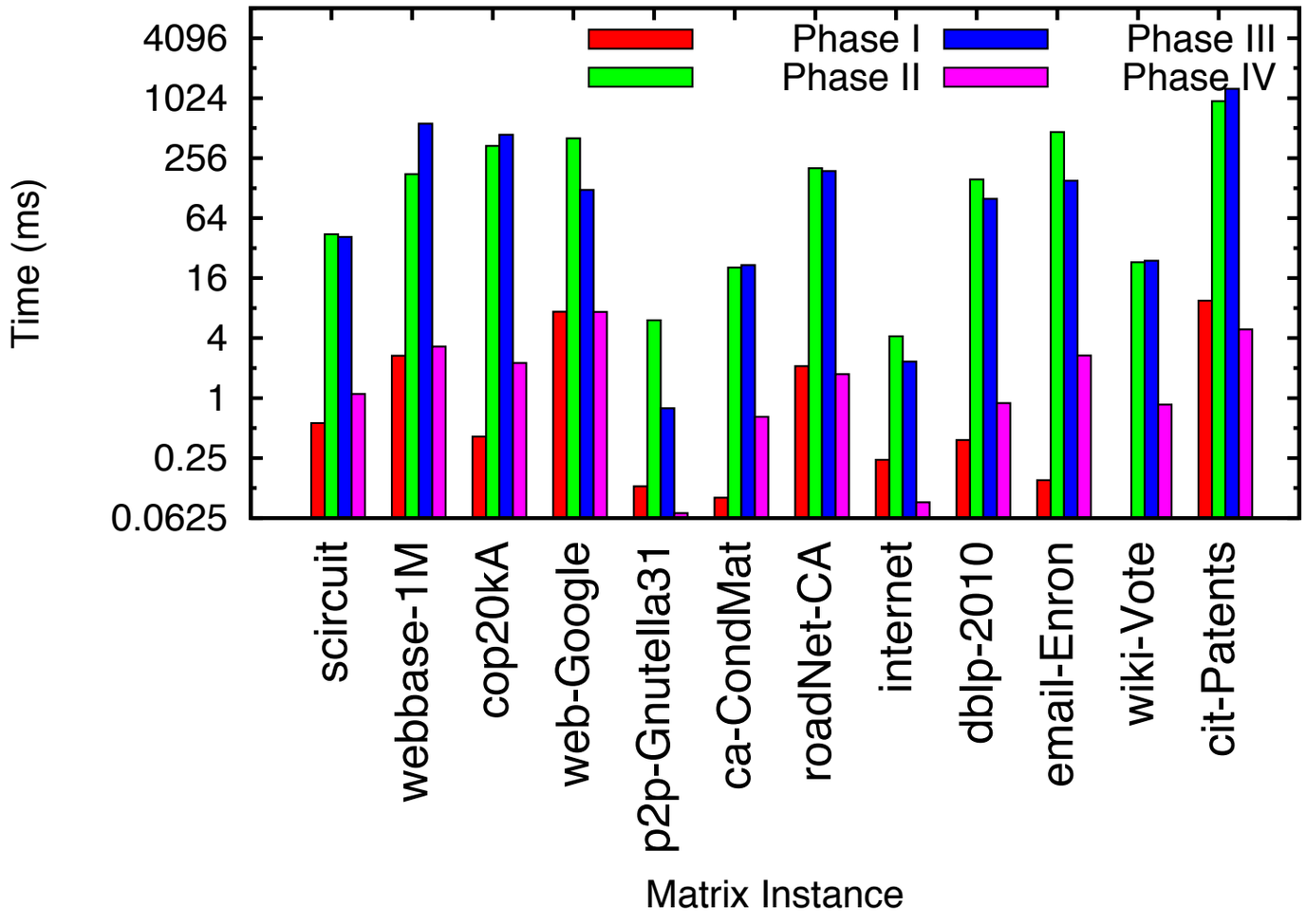


Results :: Overall Improvement

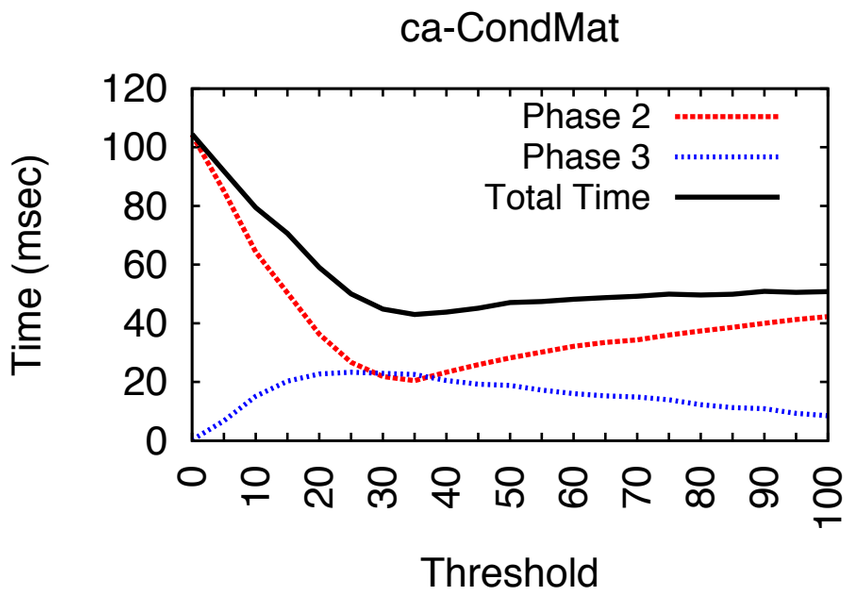
Average: 25% Faster

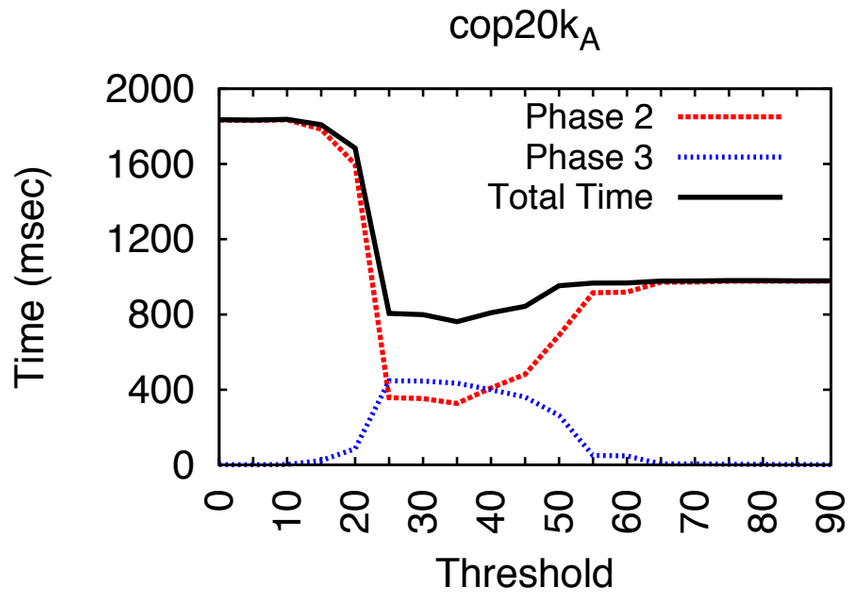


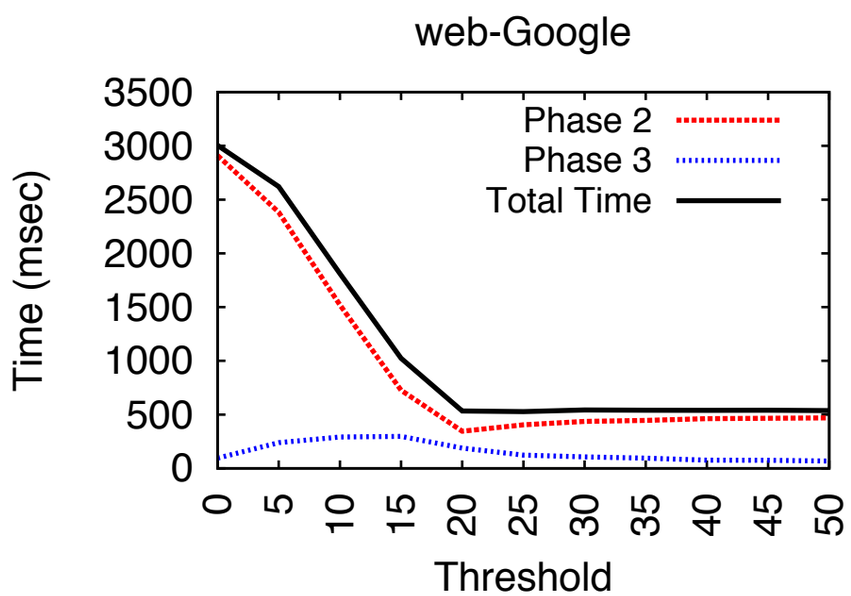
Results :: Profiling

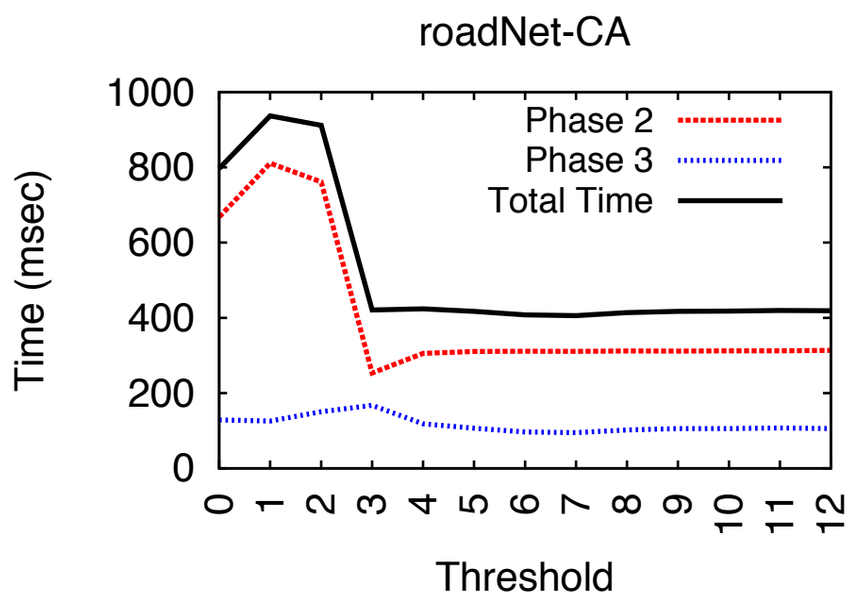


Results :: Trade-Off

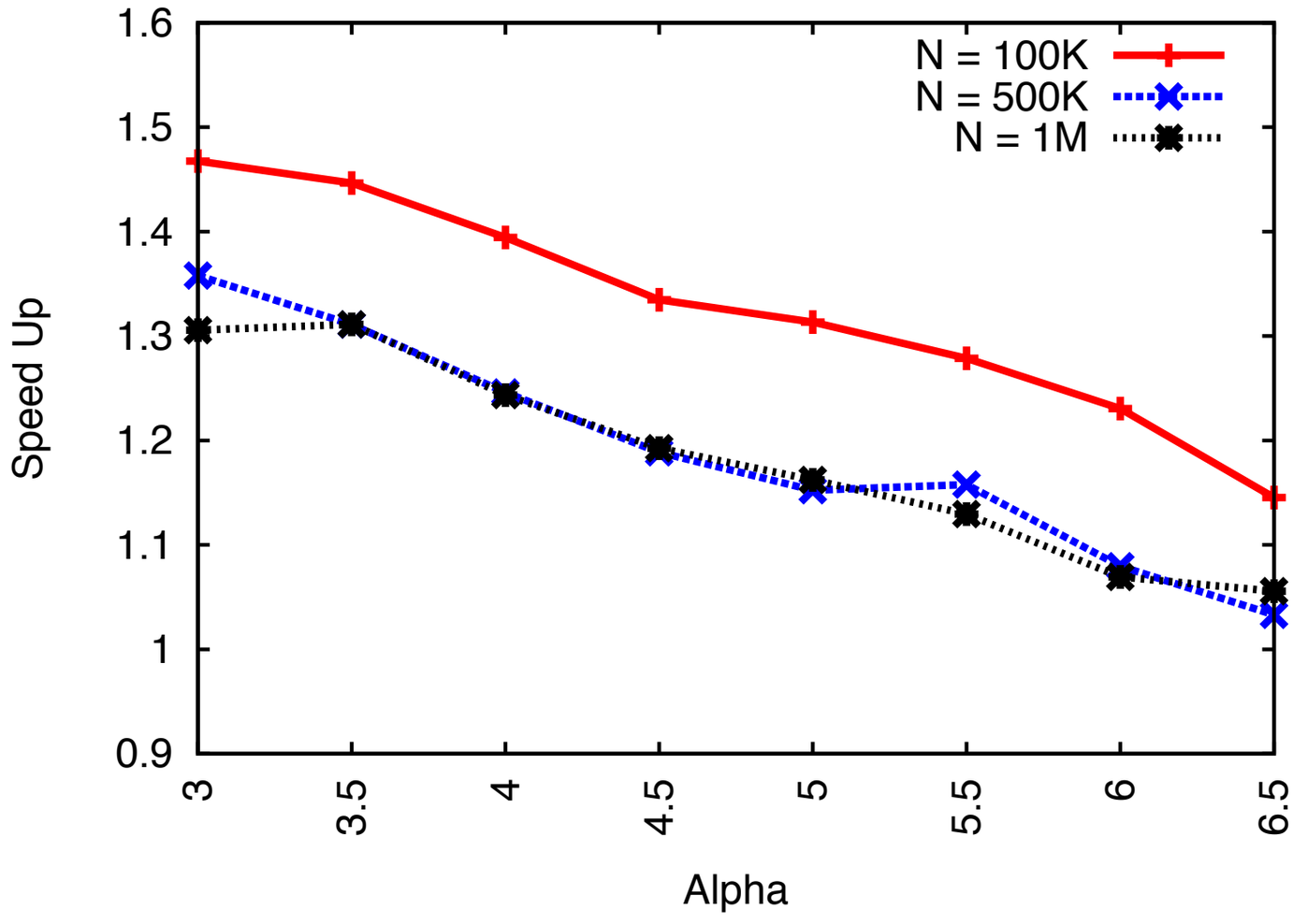






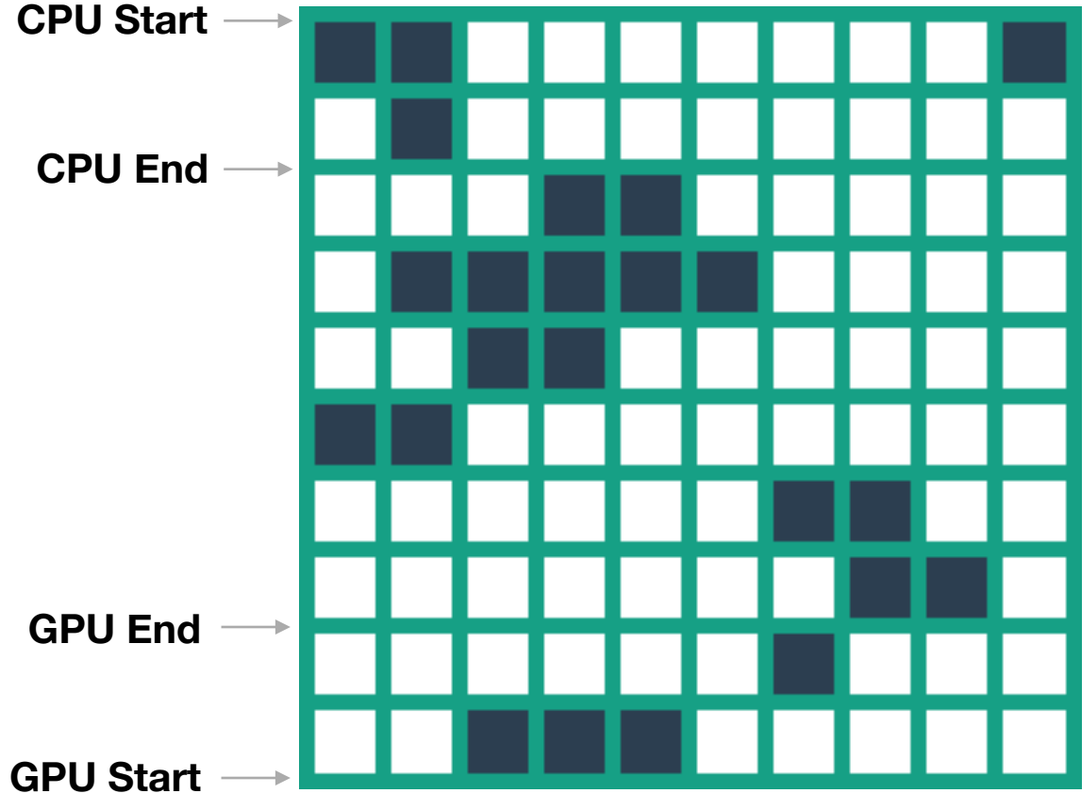


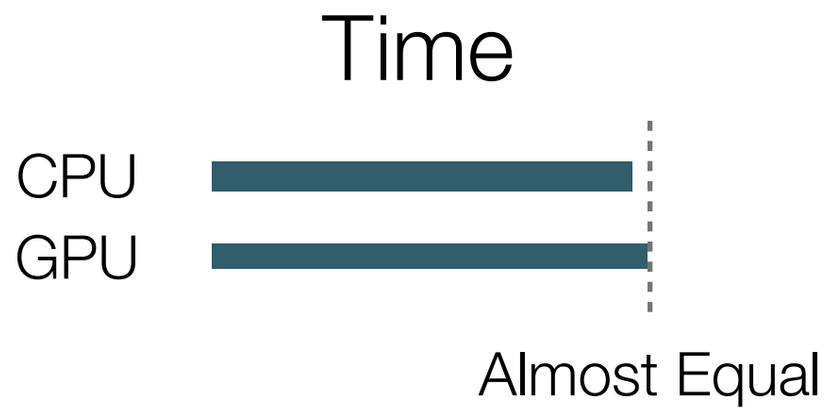
Experiments with Synthetic Datasets



Work Queue

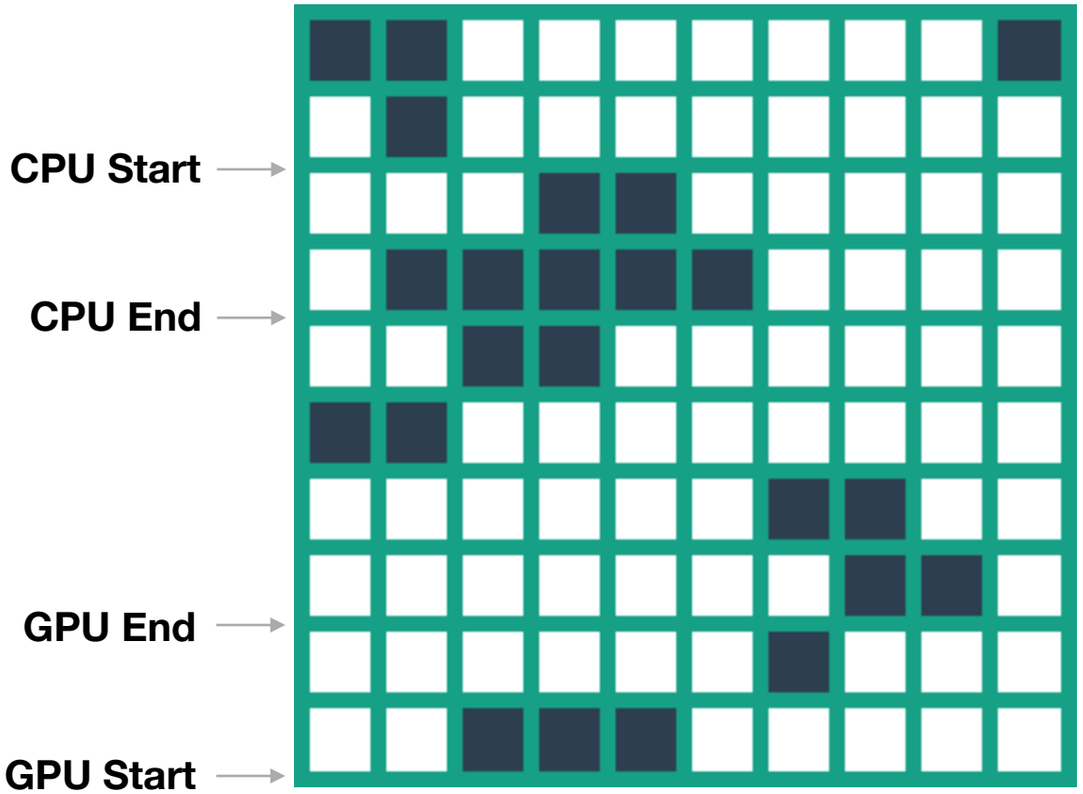
- Why not apply work queue completely ?
- CPU, GPU works with bunch of rows (work-units) until all rows are finished.
- Now, always amount of time taken by CPU & GPU is (almost) equal.
- Is it optimal ?
- No, since the rows processed by CPU & GPU are random, it might not be suited for the device and hence not optimal.

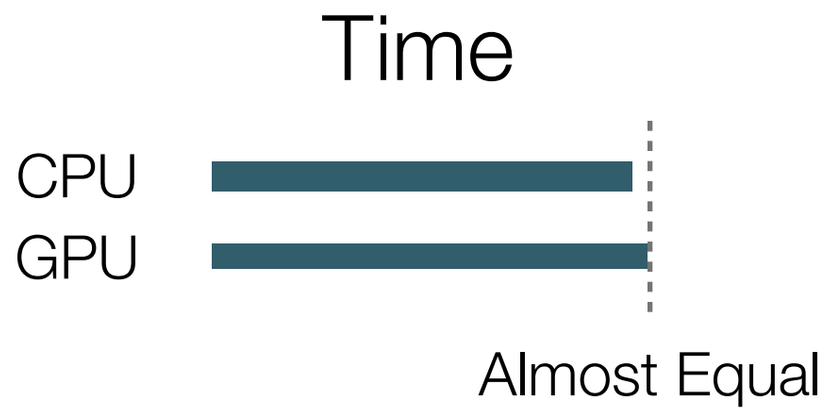




Work Queue

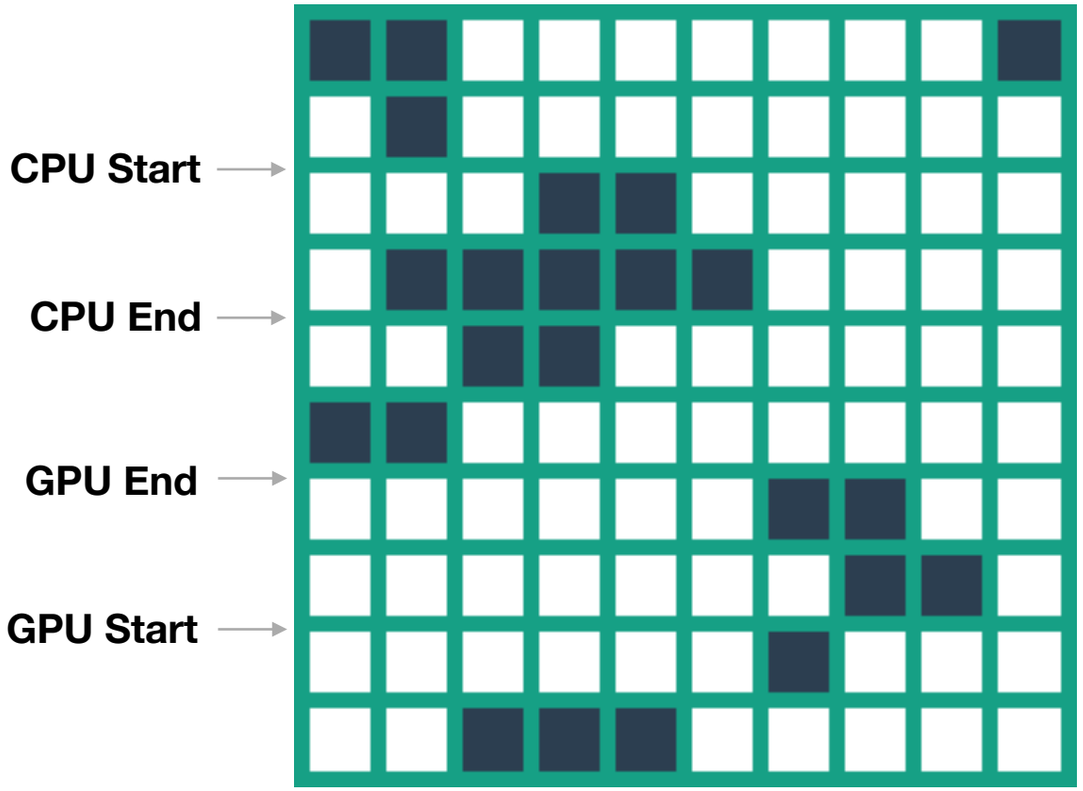
- Why not apply work queue completely ?
- CPU, GPU works with bunch of rows (work-units) until all rows are finished.
- Now, always amount of time taken by CPU & GPU is (almost) equal.
- Is it optimal ?
- No, since the rows processed by CPU & GPU are random, it might not be suited for the device and hence not optimal.

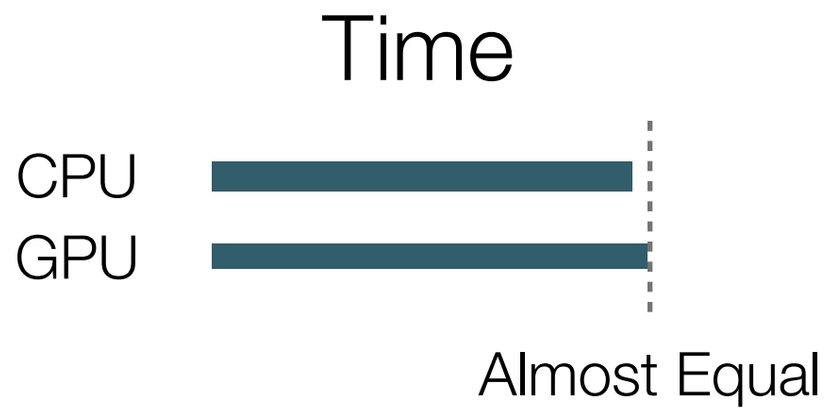




Work Queue

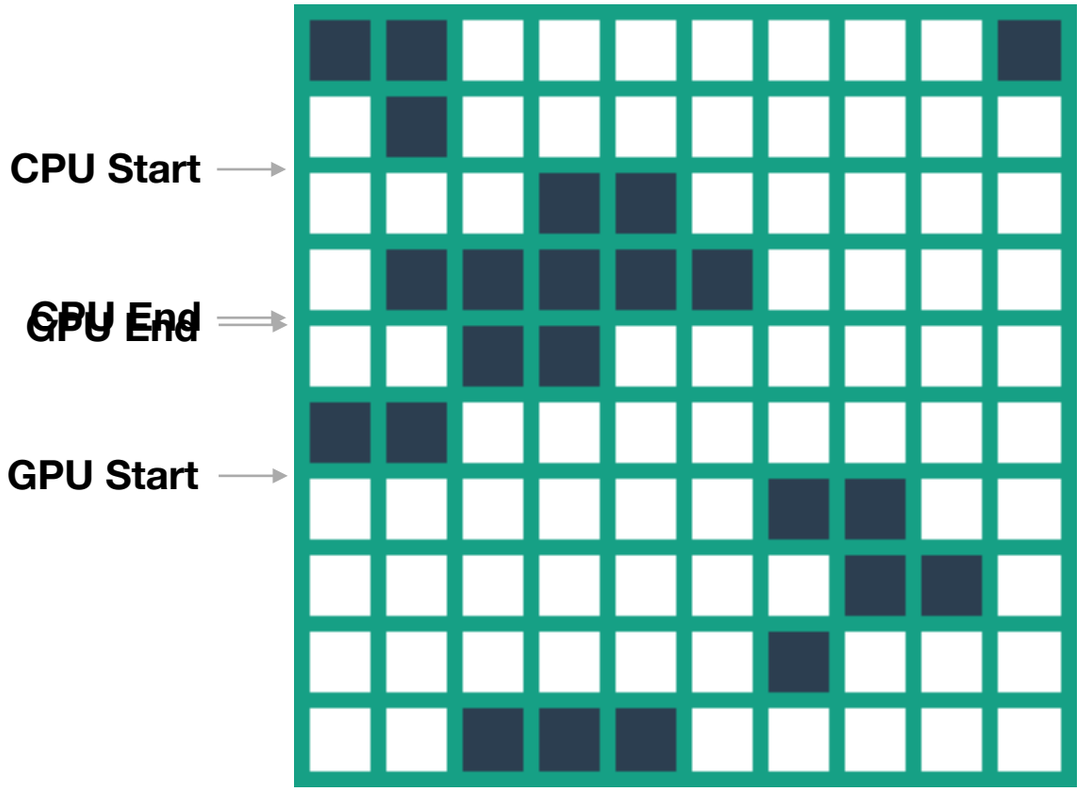
- Why not apply work queue completely ?
- CPU, GPU works with bunch of rows (work-units) until all rows are finished.
- Now, always amount of time taken by CPU & GPU is (almost) equal.
- Is it optimal ?
- No, since the rows processed by CPU & GPU are random, it might not be suited for the device and hence not optimal.

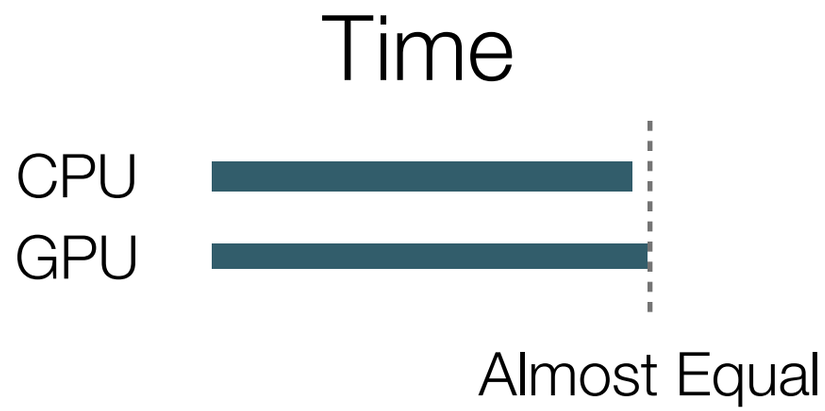




Work Queue

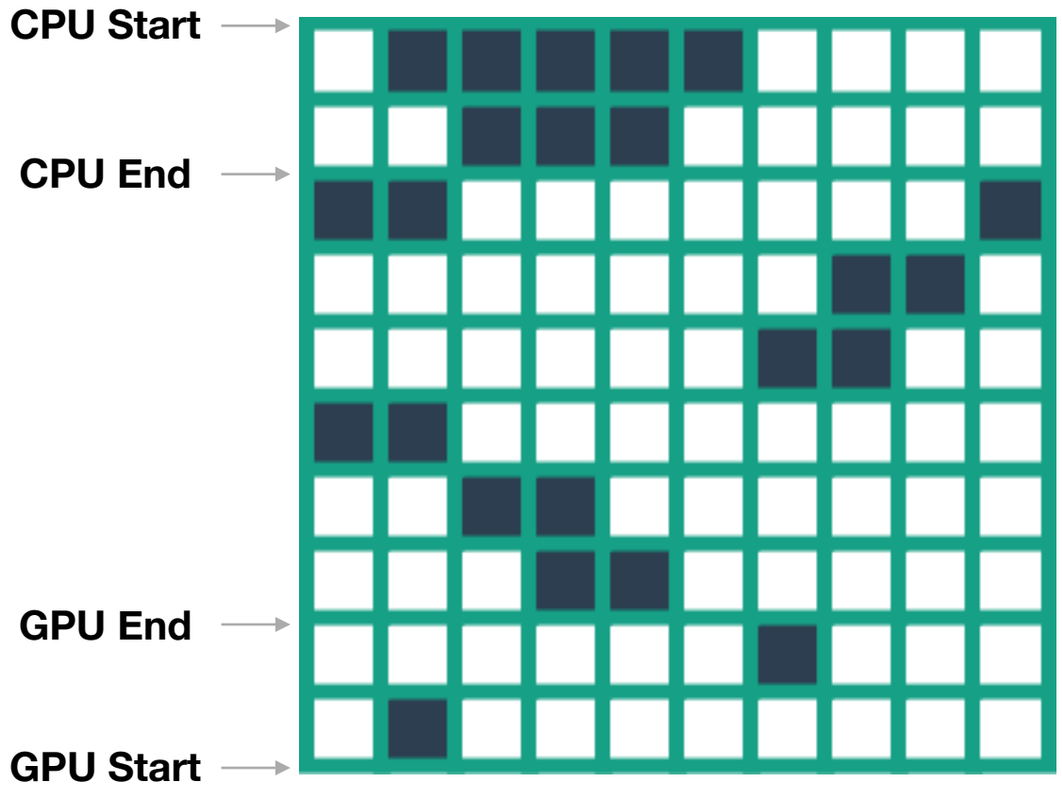
- Why not apply work queue completely ?
- CPU, GPU works with bunch of rows (work-units) until all rows are finished.
- Now, always amount of time taken by CPU & GPU is (almost) equal.
- Is it optimal ?
- No, since the rows processed by CPU & GPU are random, it might not be suited for the device and hence not optimal.





Sorted Work Queue

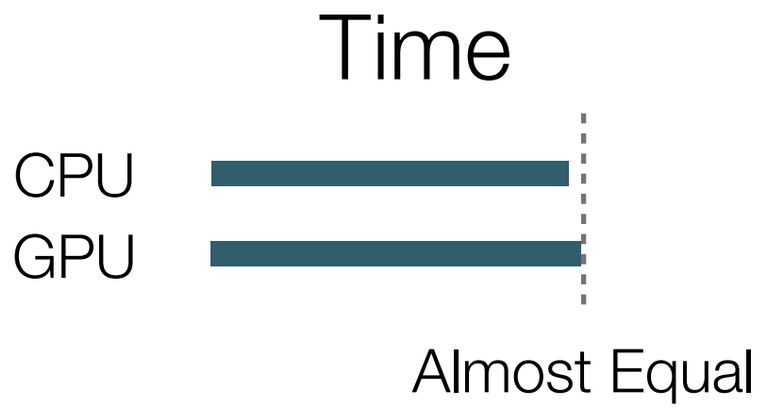
- Sort the rows such that nnz decreases. CPUs are better suited for top portion of the matrix as they are dense and can exploit cache hierarchy while bottom portion is suited for GPUs as input is almost regular.



- Again amount of time taken by CPU & GPU is (almost) equal.

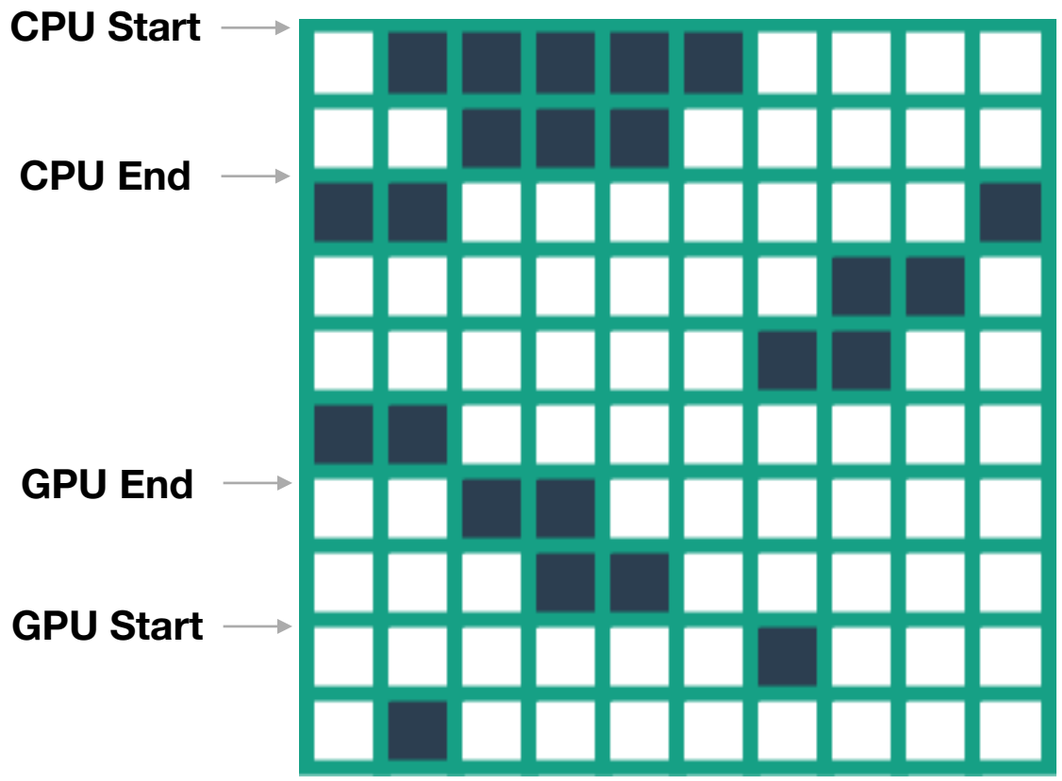
- Is it optimal ?

- No, since amount of computation done by each thread is primarily dependant on the non-zeros in the "B" matrix. This partition technique still leads to thread divergence inside a warp / block.



Sorted Work Queue

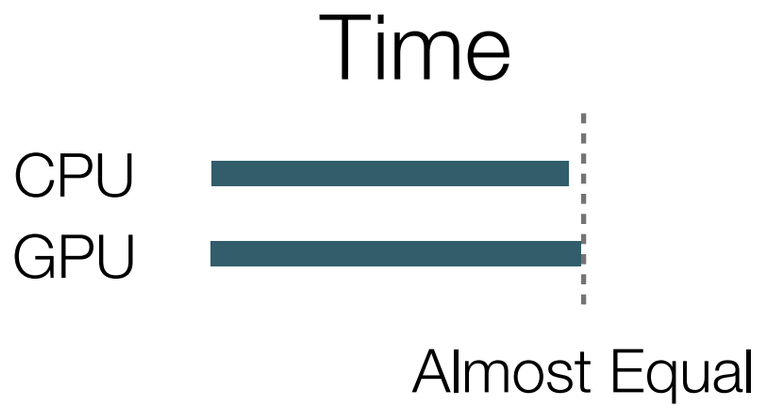
- Sort the rows such that nnz decreases. CPUs are better suited for top portion of the matrix as they are dense and can exploit cache hierarchy while bottom portion is suited for GPUs as input is almost regular.



- Again amount of time taken by CPU & GPU is (almost) equal.

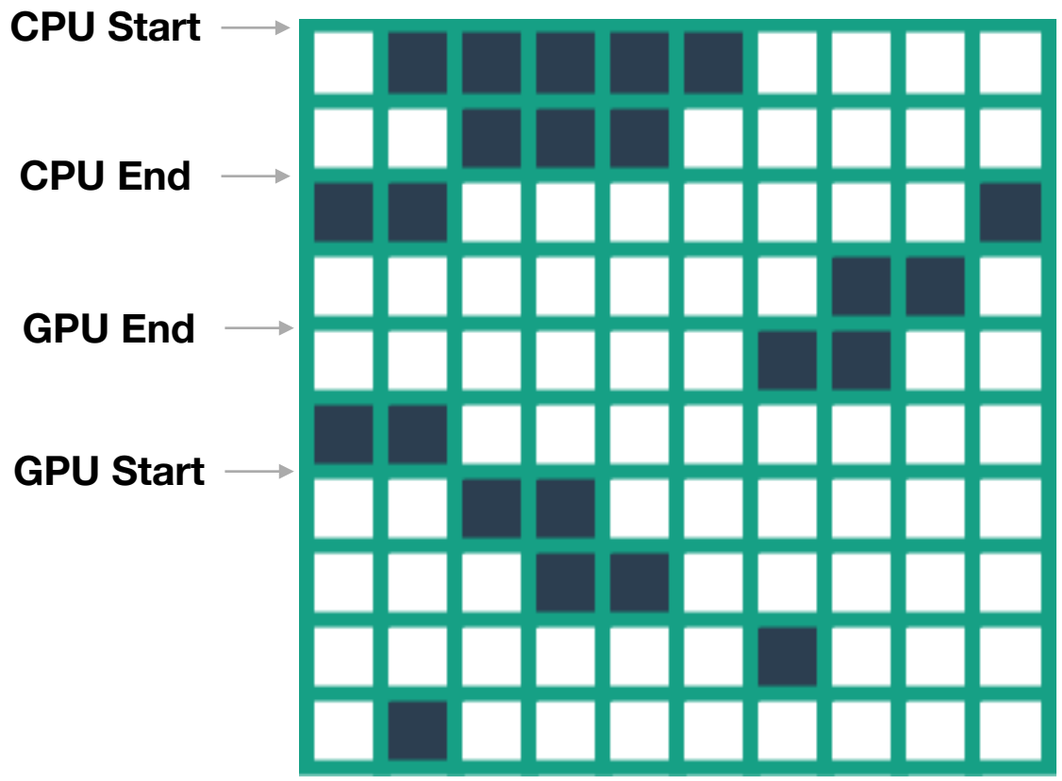
- Is it optimal ?

- No, since amount of computation done by each thread is primarily dependant on the non-zeros in the "B" matrix. This partition technique still leads to thread divergence inside a warp / block.



Sorted Work Queue

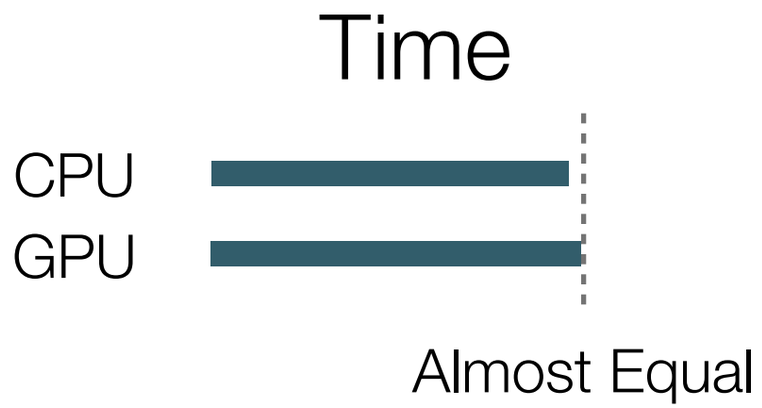
- Sort the rows such that nnz decreases. CPUs are better suited for top portion of the matrix as they are dense and can exploit cache hierarchy while bottom portion is suited for GPUs as input is almost regular.



- Again amount of time taken by CPU & GPU is (almost) equal.

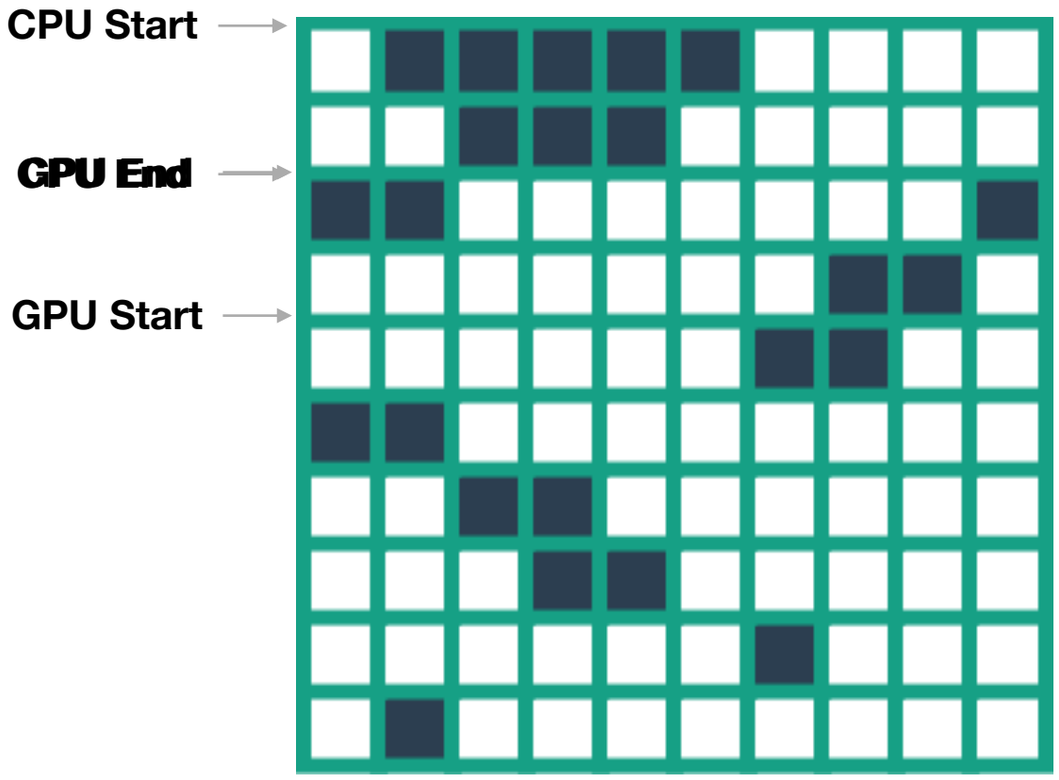
- Is it optimal ?

- No, since amount of computation done by each thread is primarily dependant on the non-zeros in the "B" matrix. This partition technique still leads to thread divergence inside a warp / block.



Sorted Work Queue

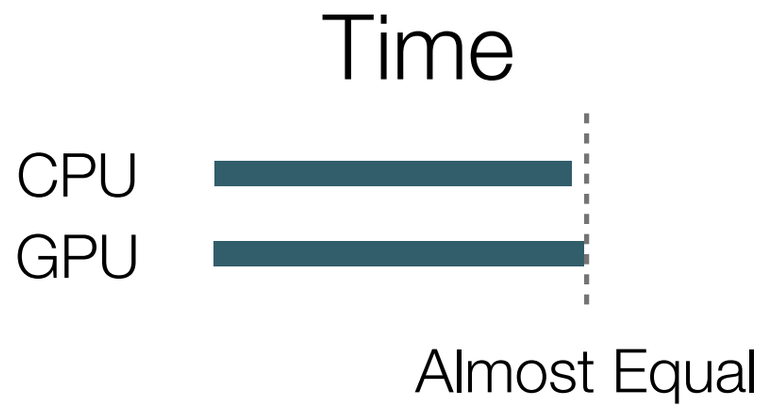
- Sort the rows such that nnz decreases. CPUs are better suited for top portion of the matrix as they are dense and can exploit cache hierarchy while bottom portion is suited for GPUs as input is almost regular.



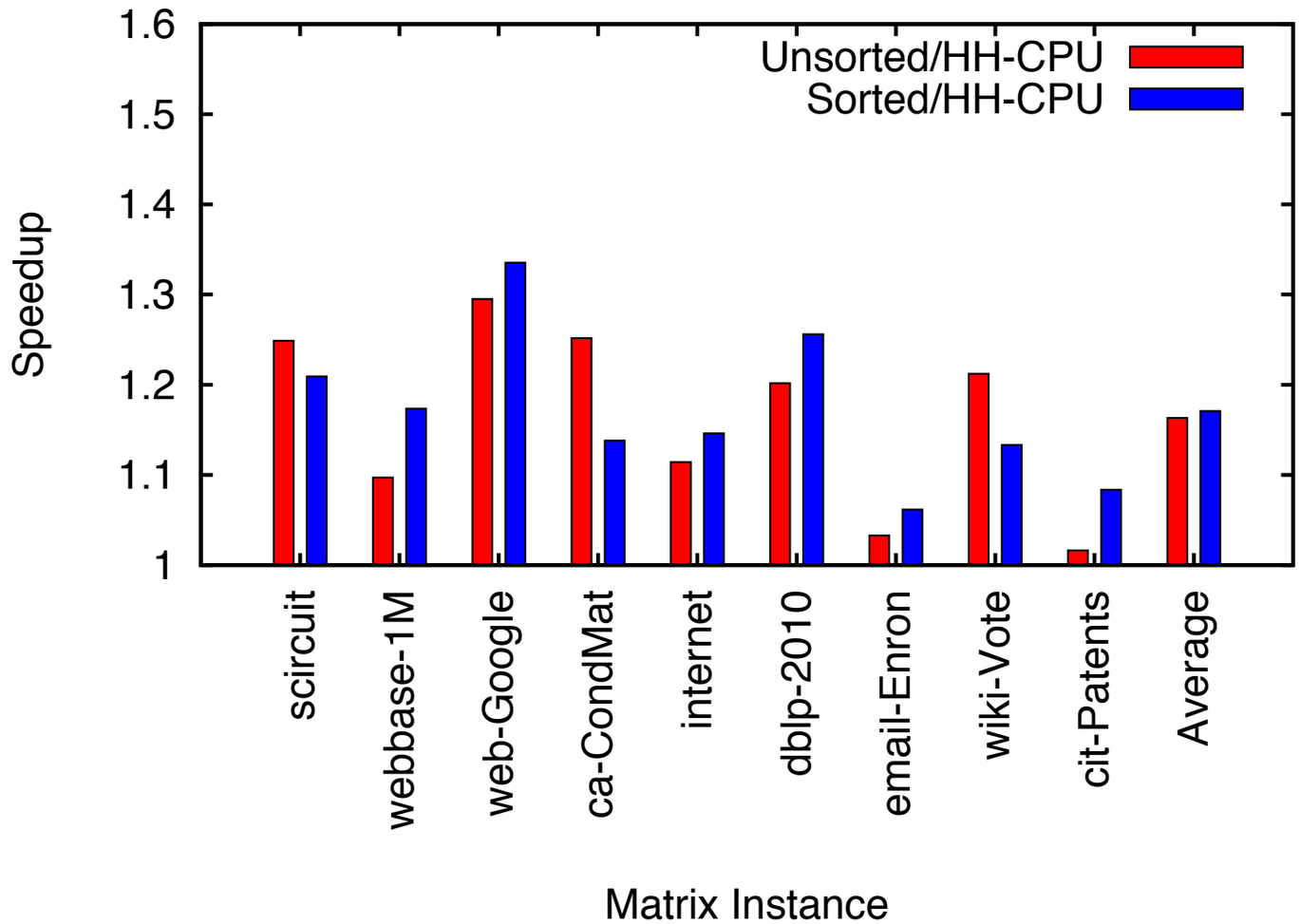
- Again amount of time taken by CPU & GPU is (almost) equal.

- Is it optimal ?

- No, since amount of computation done by each thread is primarily dependant on the non-zeros in the "B" matrix. This partition technique still leads to thread divergence inside a warp / block.



Work Queue Vs HH-CPU



Future Work

- Study analytical techniques to identify the threshold in Phase I of Algorithm HH-CPU

- Similar algorithm can be designed for CSRMM, which multiplies a sparse matrix A with a dense matrix B.

References

- D. A. Bader and K. Madduri. GTgrpah: A suite of synthetic graph generators. Available at https://sdm.lbl.gov/~kamesh/software/GTgraph/

- A. Buluc and J. R. Gilbert. Challenges and advances in parallel sparse matrix-matrix multiplication. In Proc. ICPP, pp 503–510, 2008.

- S. Indarapu, M. Maramreddy, and K. Kothapalli. Architecture- and Workload-aware algorithms for Spare Matrix- Vector Multiplication, Under submission, 2014.

- K. Matam, S. Indarapu, and K. Kothapalli. Sparse Matrix Matrix Multiplication on Modern Architectures, in Proc. of HiPC, 2012.

- NVIDIA cuSPARSE Library, https://developer.nvidia.com/cusparse

- Stanford Network Analysis Platform dataset , http://www.cise.ufl.edu/ research/ sparse/matrices/SNAP/

Thank You