Enhancing Performance of Tall-Skinny QR Factorization using FPGAs

Abid Rafique
Imperial College London

August 31, 2012

Our Claim

Common Misconception:
Dense linear algebra is always better on GPU.

We will show that it is true for square matrices but for **tall-skinny matrices**, **FPGA** is a better architecture.

Dense Linear Algebra - Tall-Skinny QR Factorization

- Stationary Video Background Subtraction

Dense Linear Algebra - Tall-Skinny QR Factorization

- Stationary Video Background Subtraction

- **Potential Real-time Applications (30 frames/sec)**
  - Video Surveillance
  - Traffic Monitoring

Dense Linear Algebra - Tall-Skinny QR Factorization

- Stationary Video Background Subtraction

- **Potential Real-time Applications (30 frames/sec)**
  - Video Surveillance
  - Traffic Monitoring

- **Robust Method**
  - Singular Value Decomposition (SVD)

Dense Linear Algebra - Tall-Skinny QR Factorization

- Tall-Skinny Matrix (M) - containing video frames as columns

$$\mathrm{M} = \begin{pmatrix} \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ f_1 & f_2 & \vdots & f_{99} & f_{100} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \vdots & \vdots \end{pmatrix}$$

$m \times n = 110{,}592 \times 100$

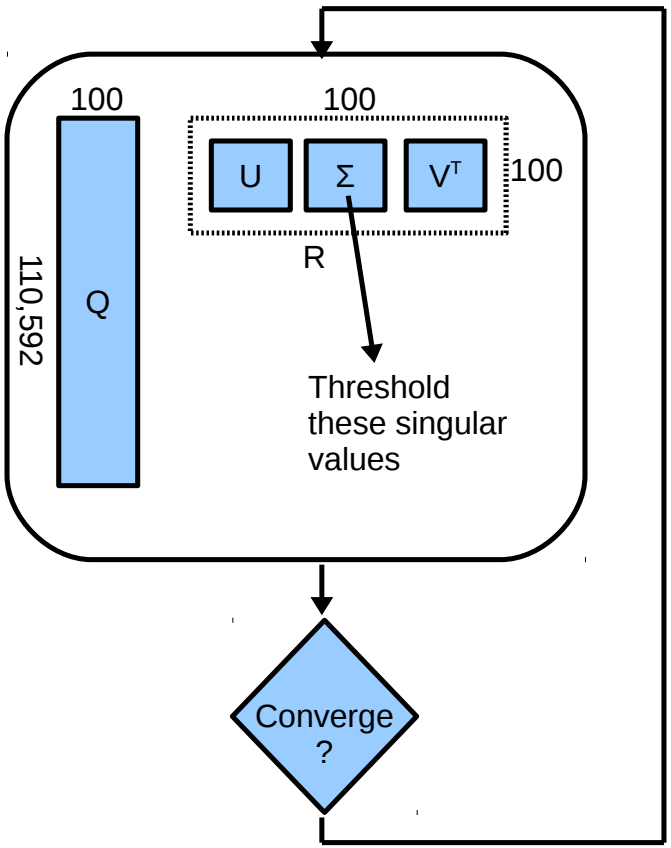
- SVD : $O(m^2 n + mn^2 + n^3)$
- QR : $O(mn^2 + n^3)$

Dense Linear Algebra - Tall-Skinny QR Factorization

- Tall-Skinny Matrix (M) - containing video frames as columns

$$M = \begin{pmatrix} \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ f_1 & f_2 & \vdots & f_{99} & f_{100} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \vdots & \vdots \end{pmatrix}$$

$m \times n = 110{,}592 \times 100$



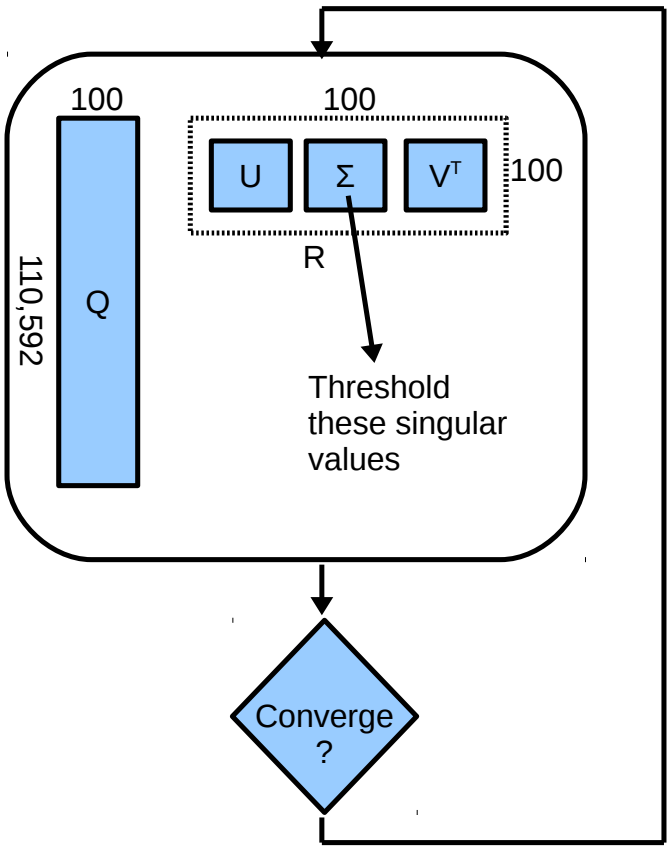
- SVD : $O(m^2n + mn^2 + n^3)$
- QR : $O(mn^2 + n^3)$

Dense Linear Algebra - Tall-Skinny QR Factorization

- Tall-Skinny Matrix (M) - containing video frames as columns

$$M = \begin{pmatrix} \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ f_1 & f_2 & \vdots & f_{99} & f_{100} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \vdots & \vdots \end{pmatrix}$$

$m \times n = 110{,}592 \times 100$



CPU-Only $\sim$ 10 min

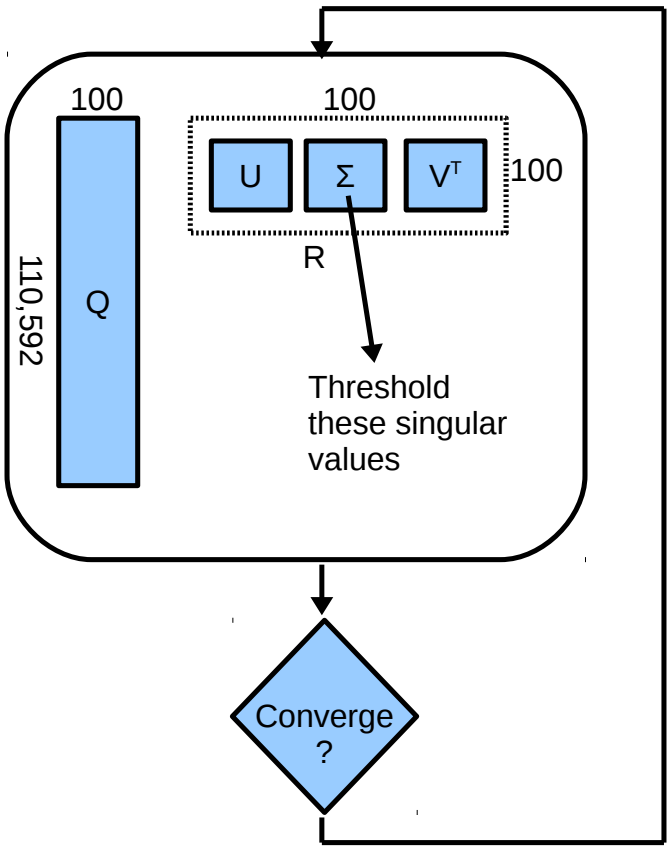
- SVD : $O(m^2n + mn^2 + n^3)$
- QR : $O(mn^2 + n^3)$

Dense Linear Algebra - Tall-Skinny QR Factorization

- Tall-Skinny Matrix (M) - containing video frames as columns

$$M = \begin{pmatrix} \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ f_1 & f_2 & \vdots & f_{99} & f_{100} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \vdots & \vdots \end{pmatrix}$$

$m \times n = 110{,}592 \times 100$



- SVD : $O(m^2n + mn^2 + n^3)$
- QR : $O(mn^2 + n^3)$

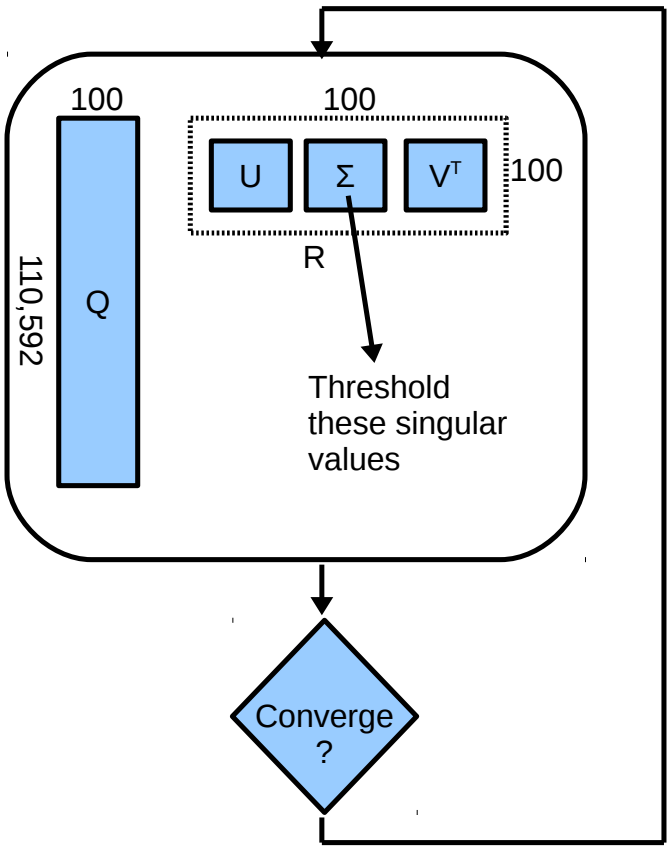
CPU-Only $\sim$ 10 min
GPU + CPU $\sim$ 17 sec

Dense Linear Algebra - Tall-Skinny QR Factorization

- Tall-Skinny Matrix (M) - containing video frames as columns

$$M = \begin{pmatrix} \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ f_1 & f_2 & \vdots & f_{99} & f_{100} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \cdots & \vdots & \vdots \end{pmatrix}$$

$m \times n = 110{,}592 \times 100$



- SVD : $O(m^2n + mn^2 + n^3)$
- QR : $O(mn^2 + n^3)$

CPU-Only $\sim$ 10 min

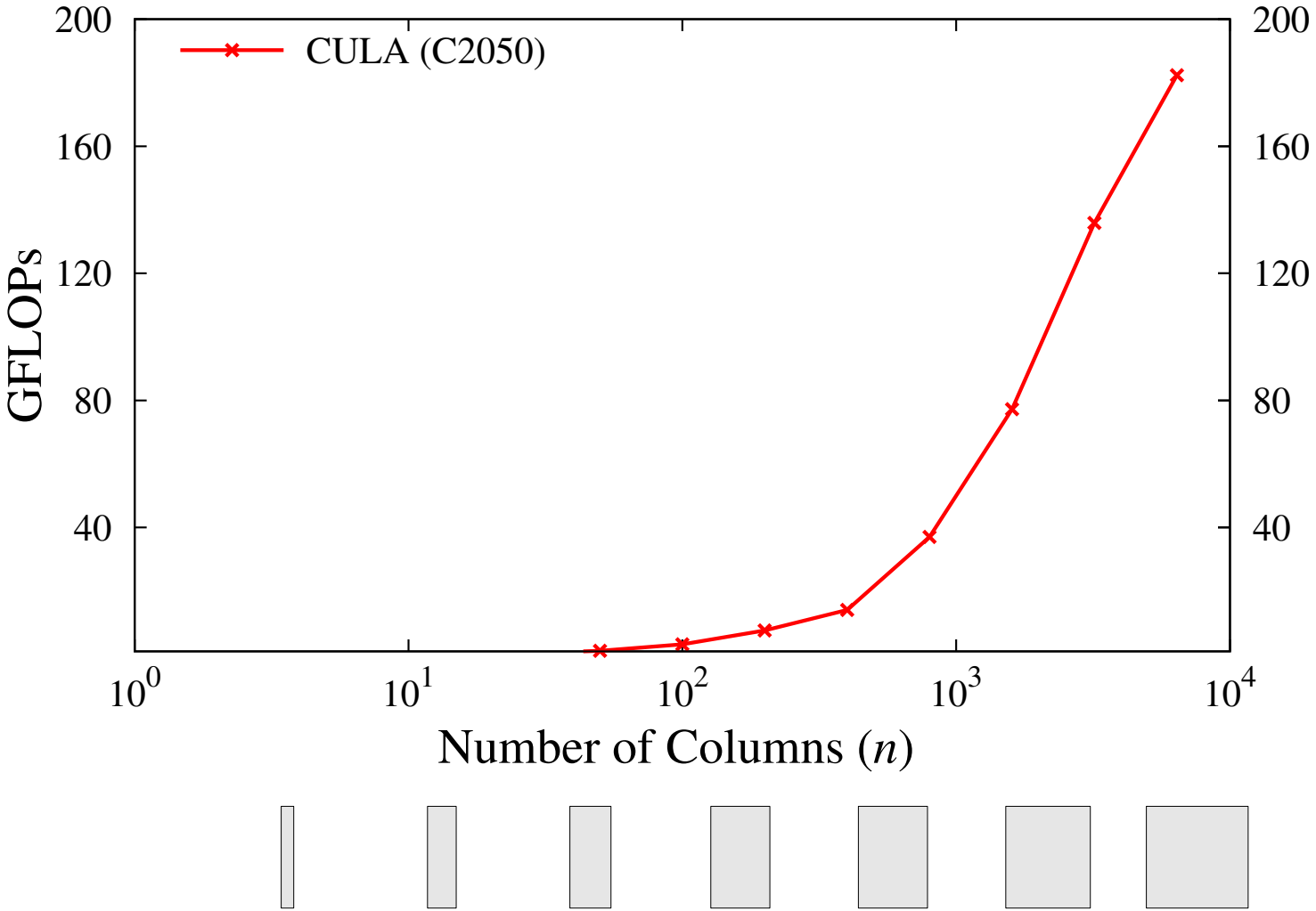
GPU + CPU $\sim$ 17 sec

FPGA + CPU$\sim$ 2-3 sec

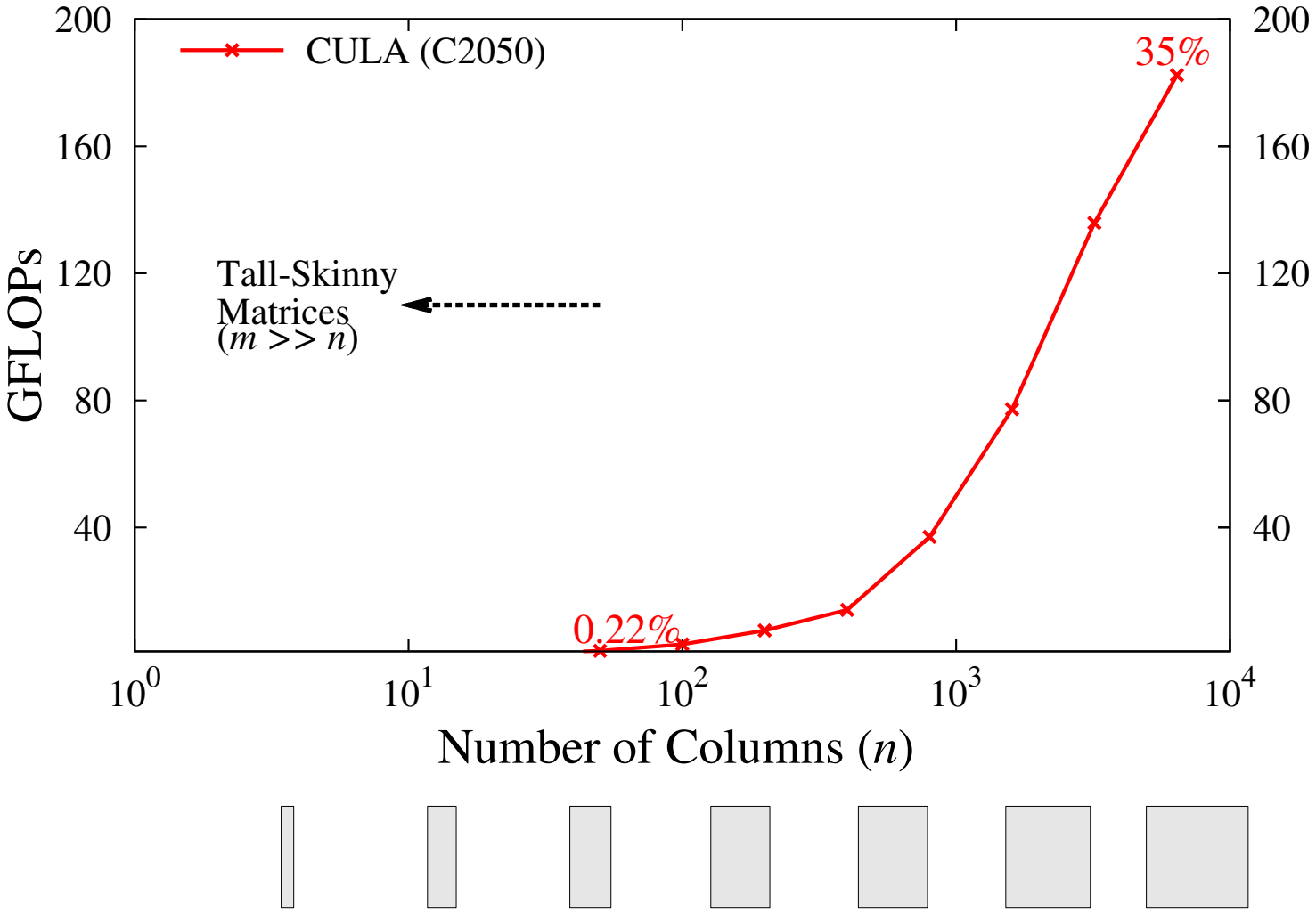
Some Extreme Aspect Ratios of Tall-Skinny Matrices ....

QR Application	**Aspect Ratio** $(\frac{m}{n})$
Background Subtraction	$\sim$ 1000
Least Squares	$\sim$ 1000
Iterative Solvers	$\sim$ 100,000

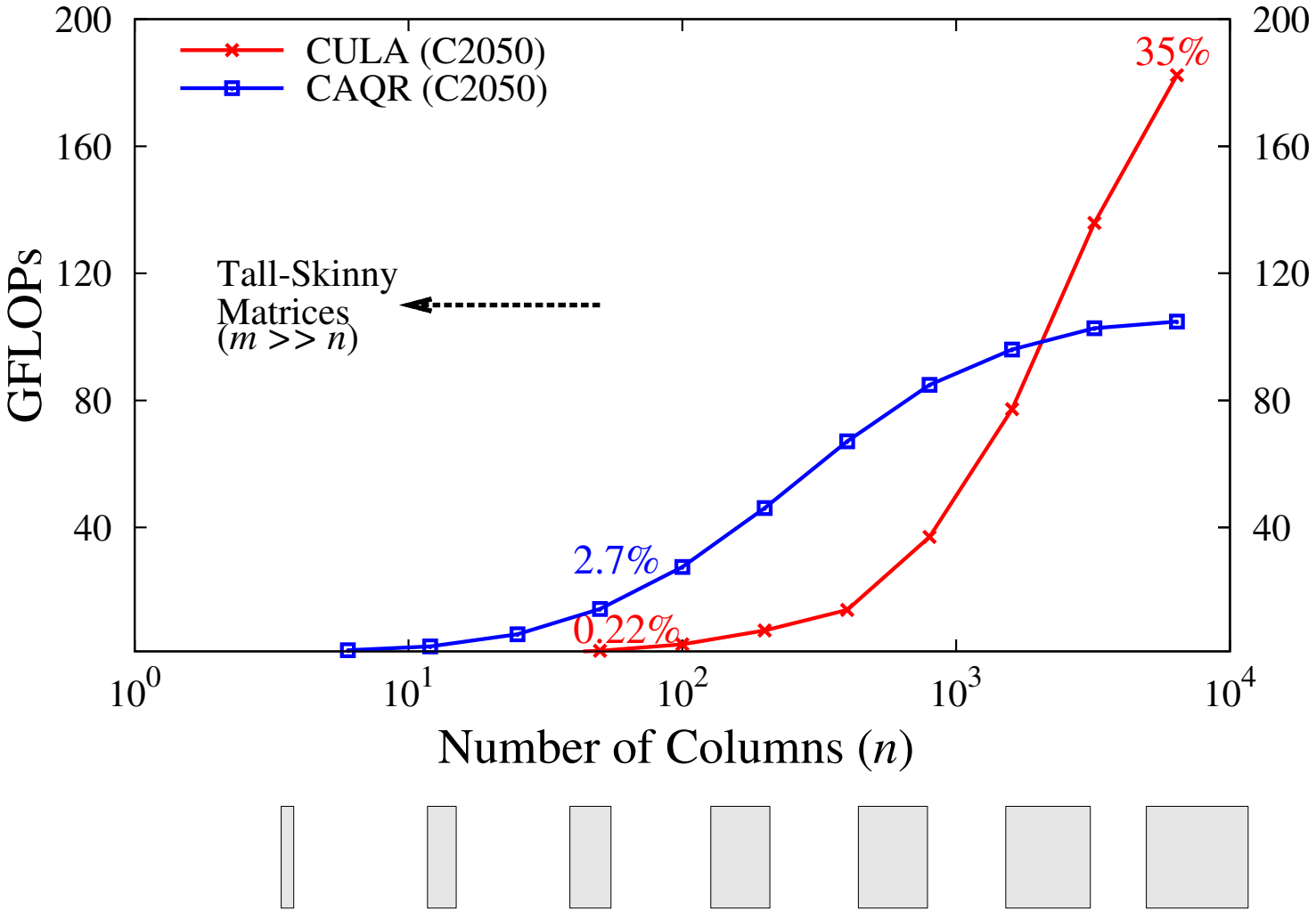
What we want to achieve?



What we want to achieve?



What we want to achieve?



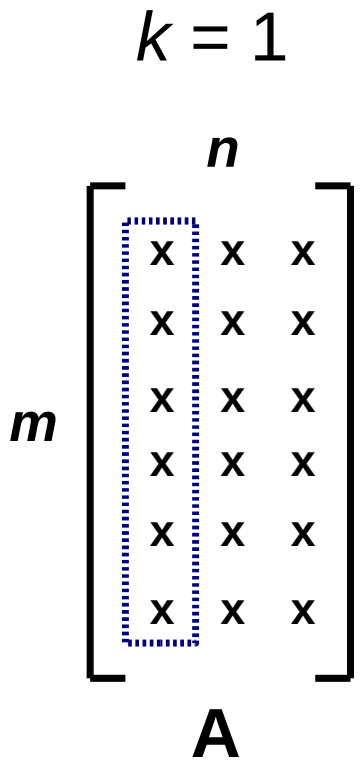
Outline

- What is QR Factorization?
- Why Tall-Skinny QR is challenging to parallelize?
- How Communication-Avoiding QR (CAQR) exposes parallelism?
- **Our Contribution**
  - **Why GPUs under-perform for CAQR?**
  - **How we can exploit architectural features of FPGAs to enhance performance?**
- Results

QR Factorization - Householder Reflections

Algorithm Householder QR

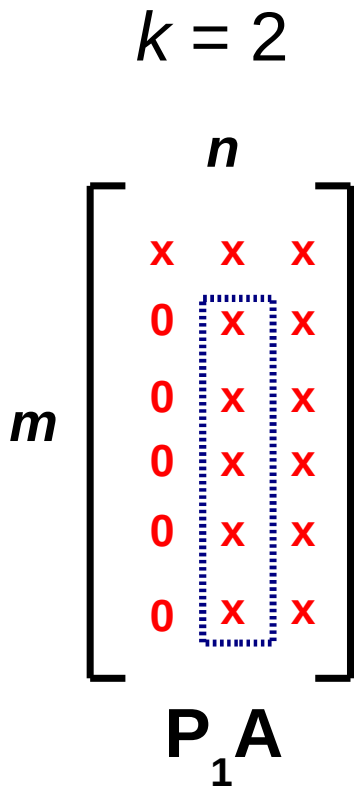
Require: A matrix $A \in \mathbb{R}^{m \times n}$
  for $k = 1$ to $n - 1$ **do**
    $x_k := A(k{:}m,\ k)$
    $[v_k, \tau_k] := \text{house}(x_k)$
    $P_k := I + \tau_k v_k v_k^T$
    $A(k{:}m,\ k{:}n) := P_k A(k{:}m,\ k{:}n)$
  end for
  return



QR Factorization - Householder Reflections

Algorithm Householder QR

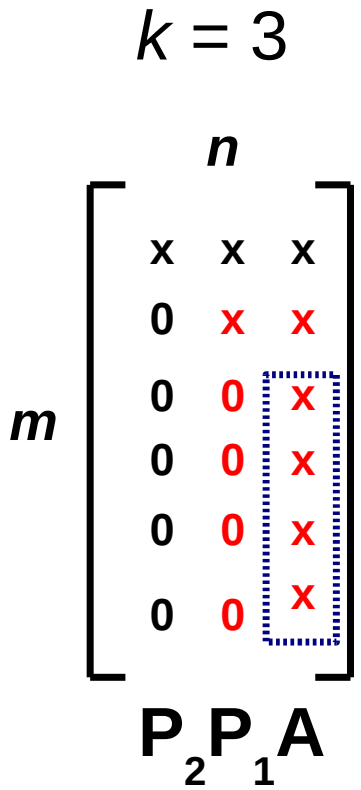
Require: A matrix $A \in \mathbb{R}^{m \times n}$
  for $k = 1$ to $n - 1$ **do**
    $x_k := A(k{:}m,\ k)$
    $[v_k, \tau_k] := \text{house}(x_k)$
    $P_k := I + \tau_k v_k v_k^T$
    $A(k{:}m,\ k{:}n) := P_k A(k{:}m,\ k{:}n)$
  end for
  return



QR Factorization - Householder Reflections

Algorithm Householder QR

Require: A matrix $A \in \mathbb{R}^{m \times n}$
  for $k = 1$ to $n - 1$ **do**
    $x_k := A(k{:}m,\ k)$
    $[v_k, \tau_k] := \text{house}(x_k)$
    $P_k := I + \tau_k v_k v_k^T$
    $A(k{:}m,\ k{:}n) := P_k A(k{:}m,\ k{:}n)$
  end for
  return



QR Factorization - Householder Reflections

Algorithm Householder QR

Require: A matrix $A \in \mathbb{R}^{m \times n}$
  for $k = 1$ to $n - 1$ **do**
    $x_k := A(k{:}m,\ k)$
    $[v_k, \tau_k] := \text{house}(x_k)$
    $P_k := I + \tau_k v_k v_k^T$
    $A(k{:}m,\ k{:}n) := P_k A(k{:}m,\ k{:}n)$
  end for
  return

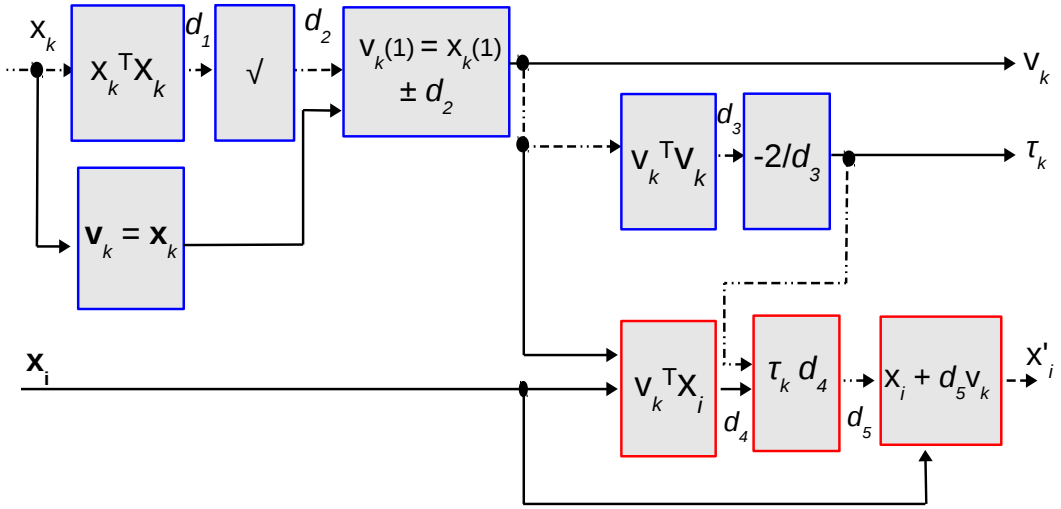
$k = 3$

$$R = P_{n-1} P_{n-2} \cdots P_1 A$$
$$Q = P_1^T P_2^T \cdots P_{n-2}^T P_{n-1}^T$$

Why Tall-Skinny Matrices are challenging to parallelize?

Algorithm Householder QR

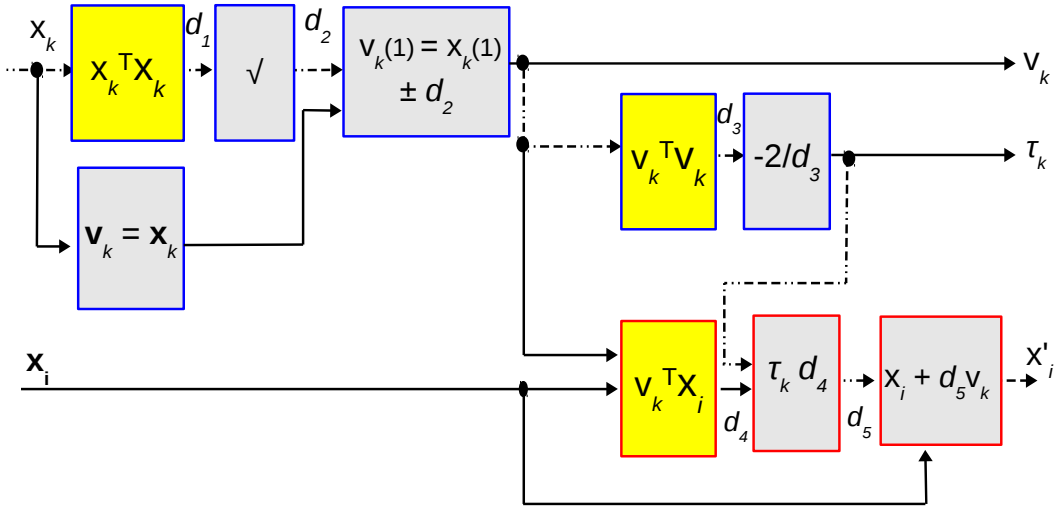
Require: A matrix $A \in \mathbb{R}^{m \times n}$
  for $k = 1$ to $n - 1$ **do**
    $x_k := A(k{:}m,\ k)$
    $[v_k, \tau_k] := \text{house}(x_k)$
    $P_k := I + \tau_k v_k v_k^T$
    $A(k{:}m,\ k{:}n) := P_k A(k{:}m,\ k{:}n)$
  end for
  return



Why Tall-Skinny Matrices are challenging to parallelize?

Algorithm Householder QR

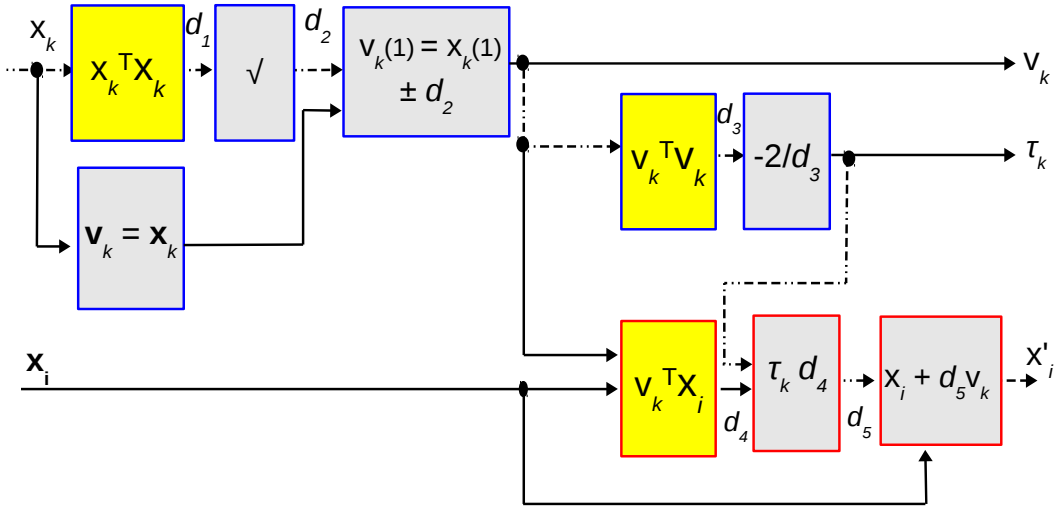
Require: A matrix $A \in \mathbb{R}^{m \times n}$
  for $k = 1$ to $n - 1$ **do**
    $x_k := A(k{:}m,\ k)$
    $[v_k, \tau_k] := \text{house}(x_k)$
    $P_k := I + \tau_k v_k v_k^T$
    $A(k{:}m,\ k{:}n) := P_k A(k{:}m,\ k{:}n)$
  end for
  return

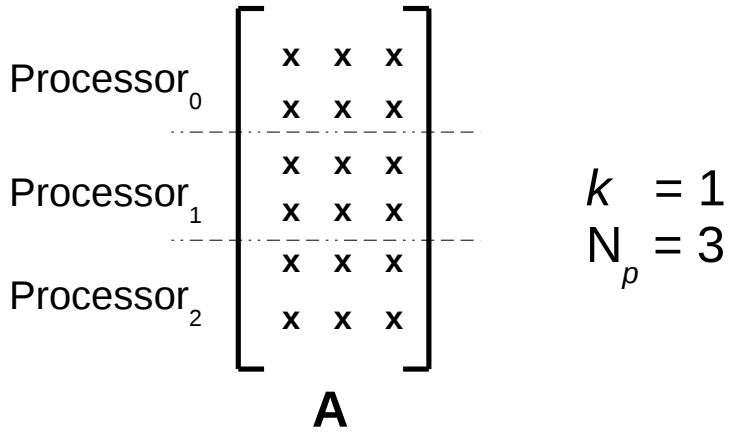


Why Tall-Skinny Matrices are challenging to parallelize?

Algorithm Householder QR

Require: A matrix $A \in \mathbb{R}^{m \times n}$
  for $k = 1$ to $n - 1$ **do**
    $x_k := A(k{:}m,\ k)$
    $[v_k, \tau_k] := \text{house}(x_k)$
    $P_k := I + \tau_k v_k v_k^T$
    $A(k{:}m,\ k{:}n) := P_k A(k{:}m,\ k{:}n)$
  end for
  return

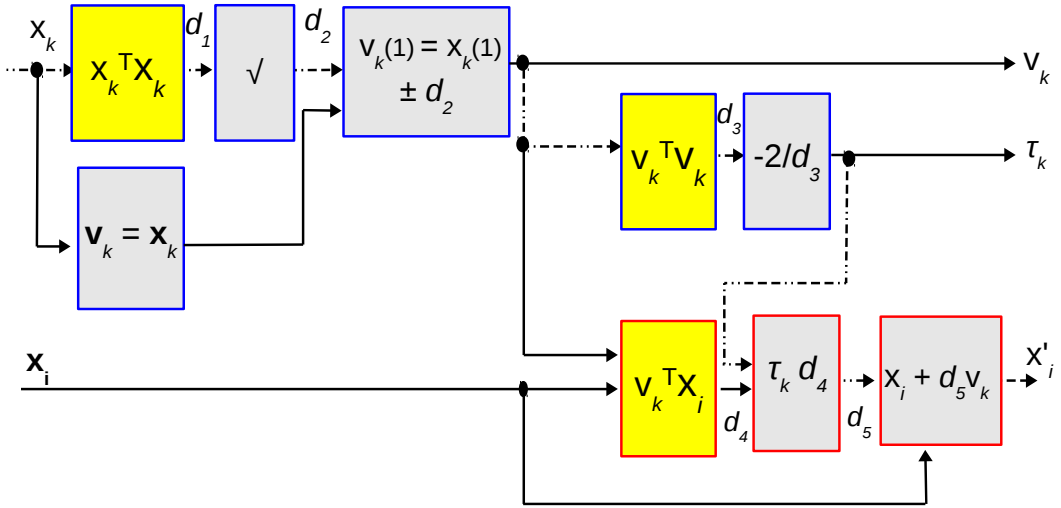




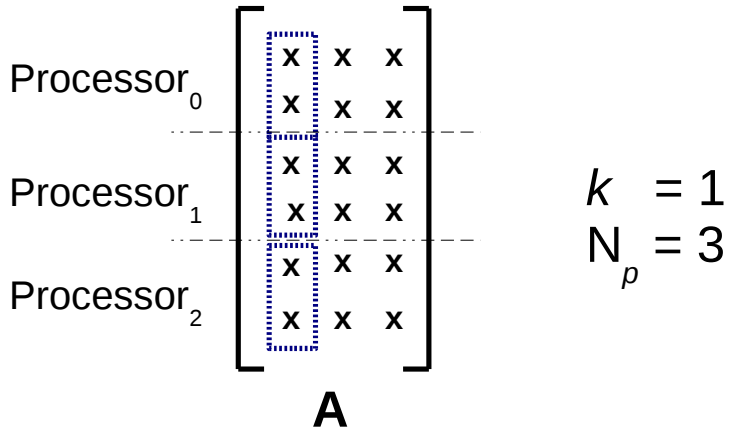
Why Tall-Skinny Matrices are challenging to parallelize?

Algorithm Householder QR

Require: A matrix $A \in \mathbb{R}^{m \times n}$
  for $k = 1$ to $n - 1$ **do**
    $x_k := A(k{:}m,\ k)$
    $[v_k, \tau_k] := \text{house}(x_k)$
    $P_k := I + \tau_k v_k v_k^T$
    $A(k{:}m,\ k{:}n) := P_k A(k{:}m,\ k{:}n)$
  end for
  return



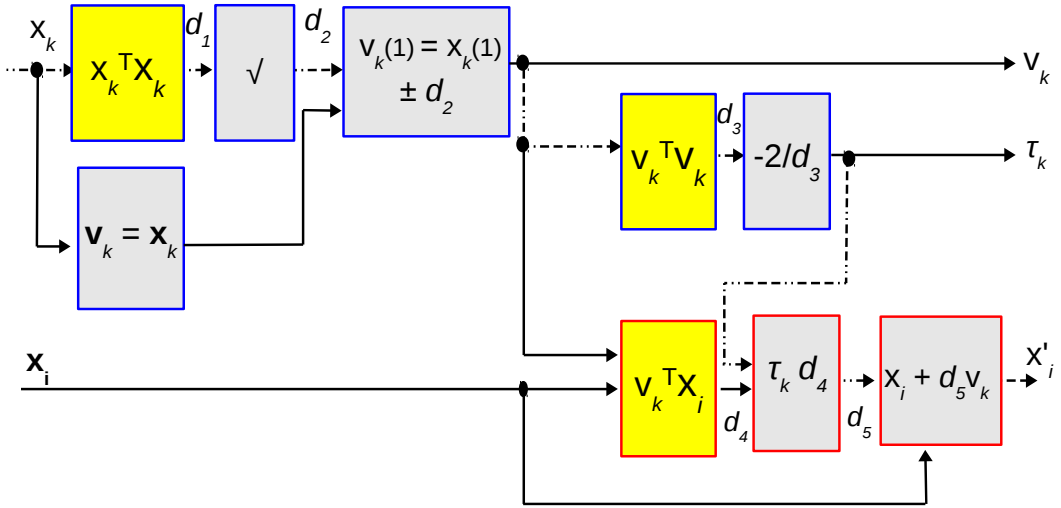
- $x_k^T x_k - \log_2 N_p$



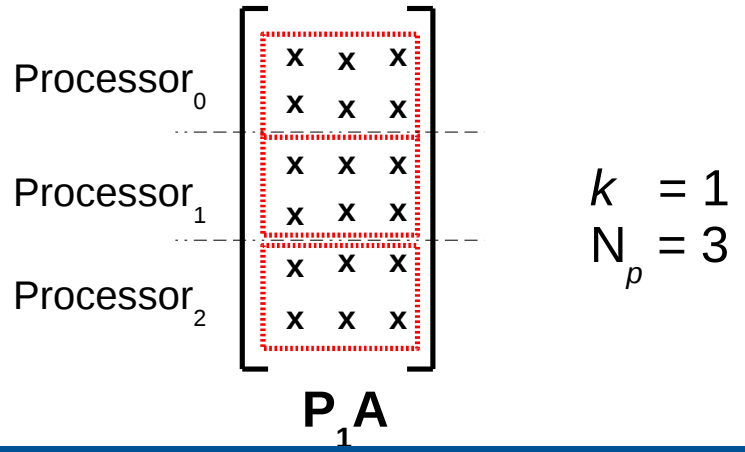
Why Tall-Skinny Matrices are challenging to parallelize?

Algorithm Householder QR

Require: A matrix $A \in \mathbb{R}^{m \times n}$
  for $k = 1$ to $n - 1$ **do**
    $x_k := A(k{:}m,\ k)$
    $[v_k, \tau_k] := \text{house}(x_k)$
    $P_k := I + \tau_k v_k v_k^T$
    $A(k{:}m,\ k{:}n) := P_k A(k{:}m,\ k{:}n)$
  end for
  return



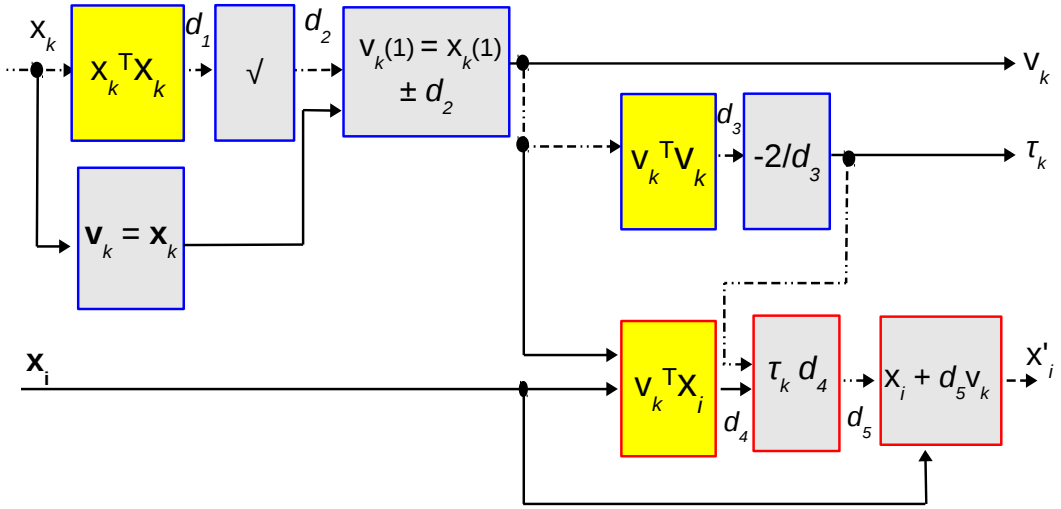
- $x_k^T x_k - \log_2 N_p$
- $v_k^T x_i - \log_2 N_p$



Why Tall-Skinny Matrices are challenging to parallelize?

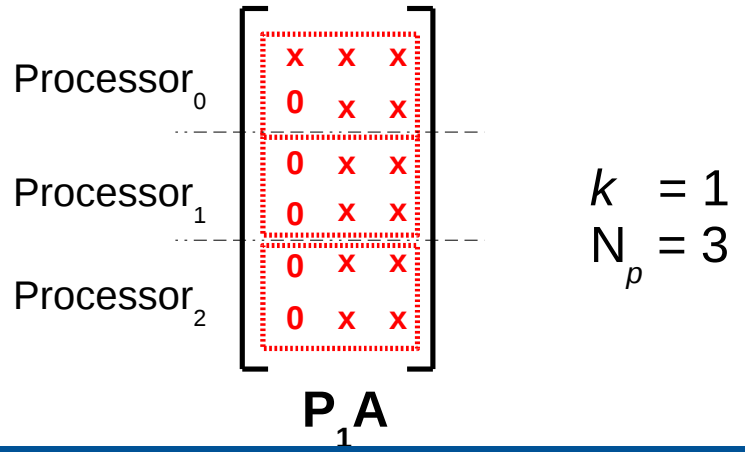
Algorithm Householder QR

Require: A matrix $A \in \mathbb{R}^{m \times n}$
  for $k = 1$ to $n - 1$ **do**
    $x_k := A(k{:}m,\ k)$
    $[v_k, \tau_k] := \text{house}(x_k)$
    $P_k := I + \tau_k v_k v_k^T$
    $A(k{:}m,\ k{:}n) := P_k A(k{:}m,\ k{:}n)$
  end for
  return



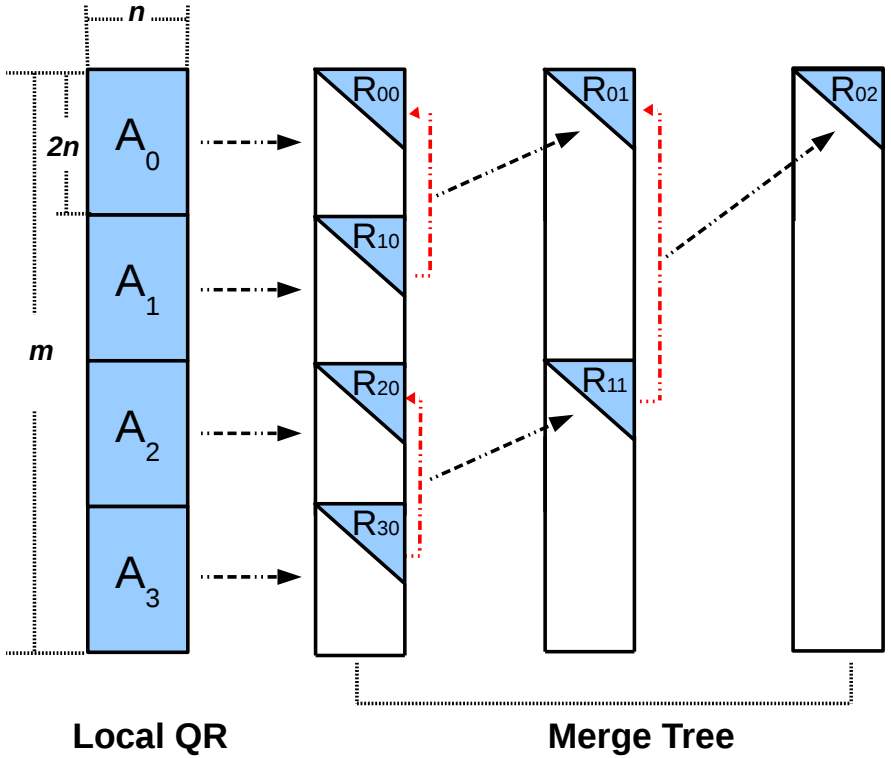
- $x_k^T x_k - \log_2 N_p$
- $v_k^T x_i - \log_2 N_p$

$O(n \log N_p)$ Sync. Points



Communication-Avoiding QR Factorization

- **Communication is costly compared to computation**
- Divide-and-Conquer Approach- Do as much local computations as possible and avoid communications
- Only R factors are communicated to the neighbours

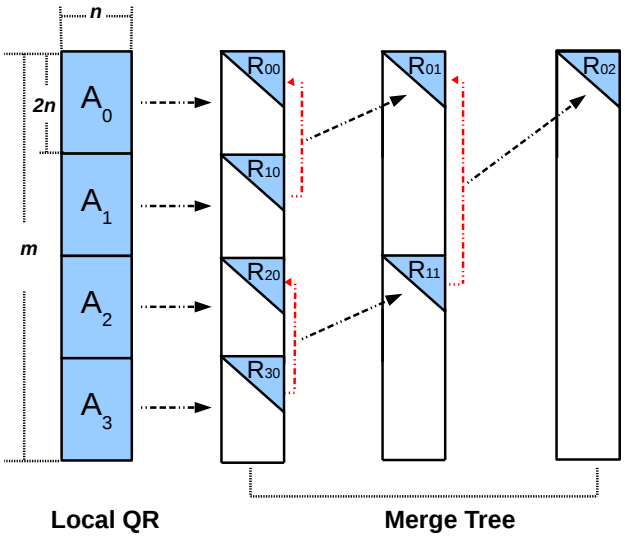


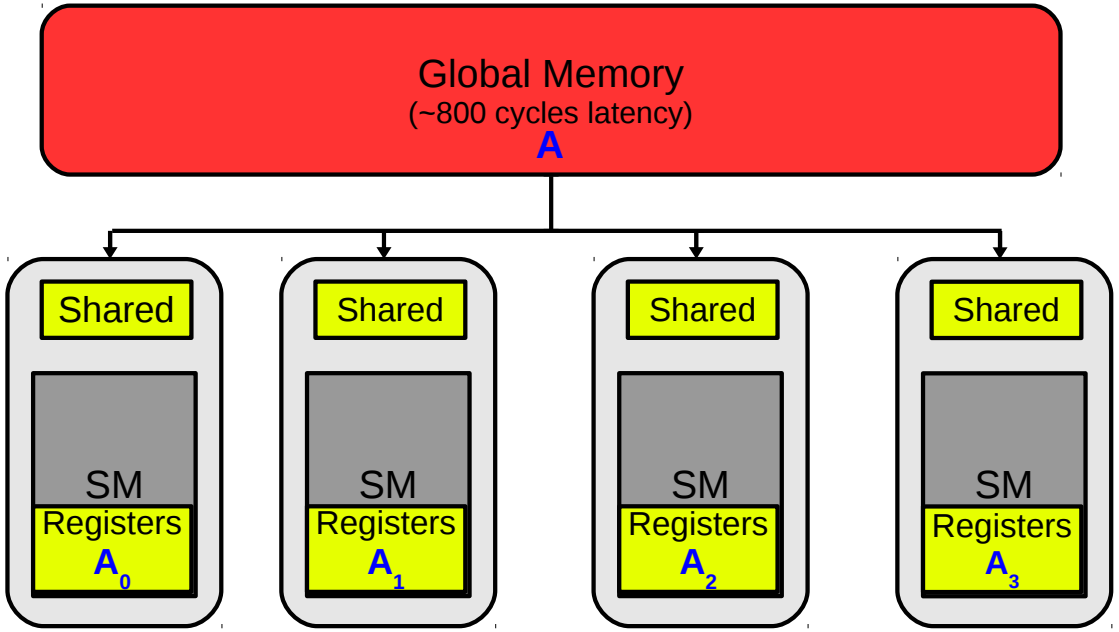
$O(\log \mathbf{N}_p)$
Sync. Points

Outline

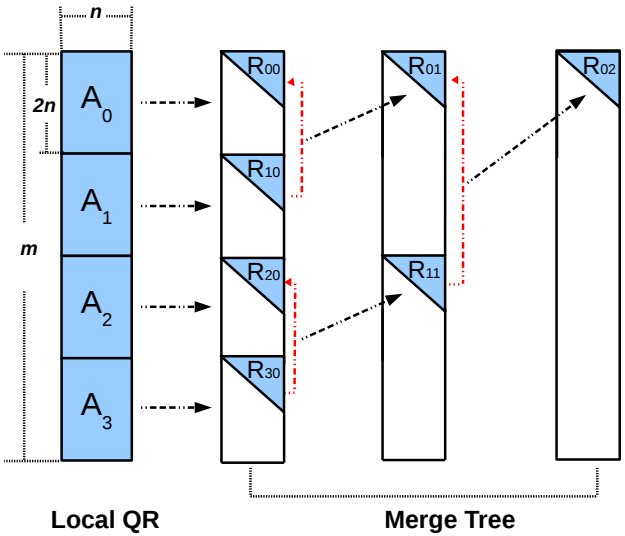
- What is QR Factorization?
- Why Tall-Skinny QR is challenging to parallelise?
- How Communication-Avoiding QR (CAQR) exposes parallelism?
- **Our Contribution**
  - **Why GPUs under-perform for CAQR?**
  - **How we can exploit architectural features of FPGAs to enhance performance?**
- Results

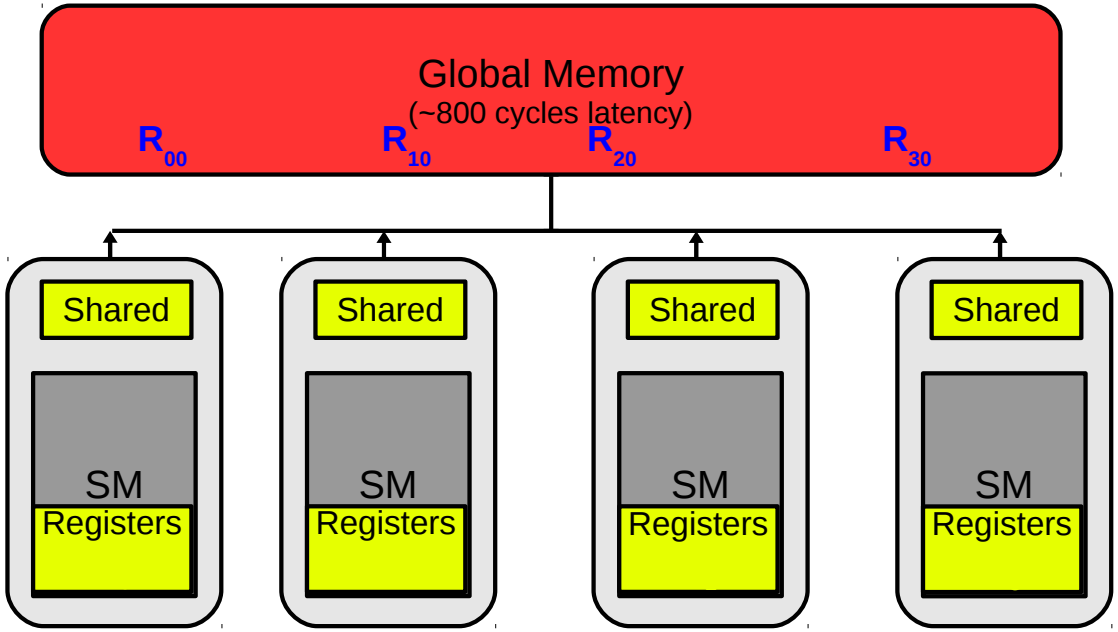
Why GPUs under-perform for CAQR?



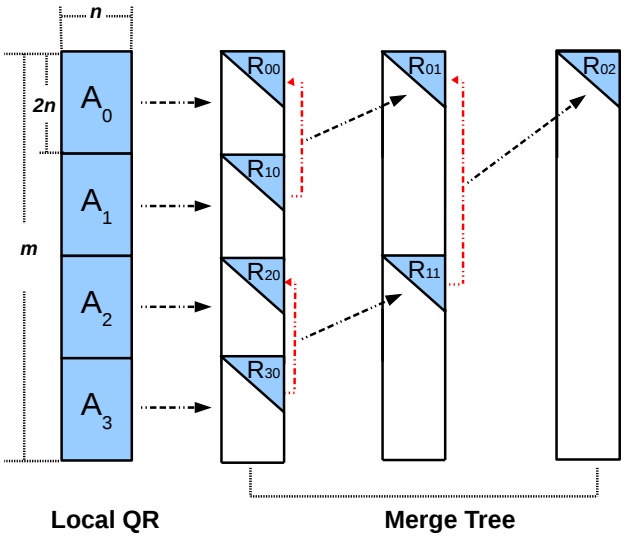


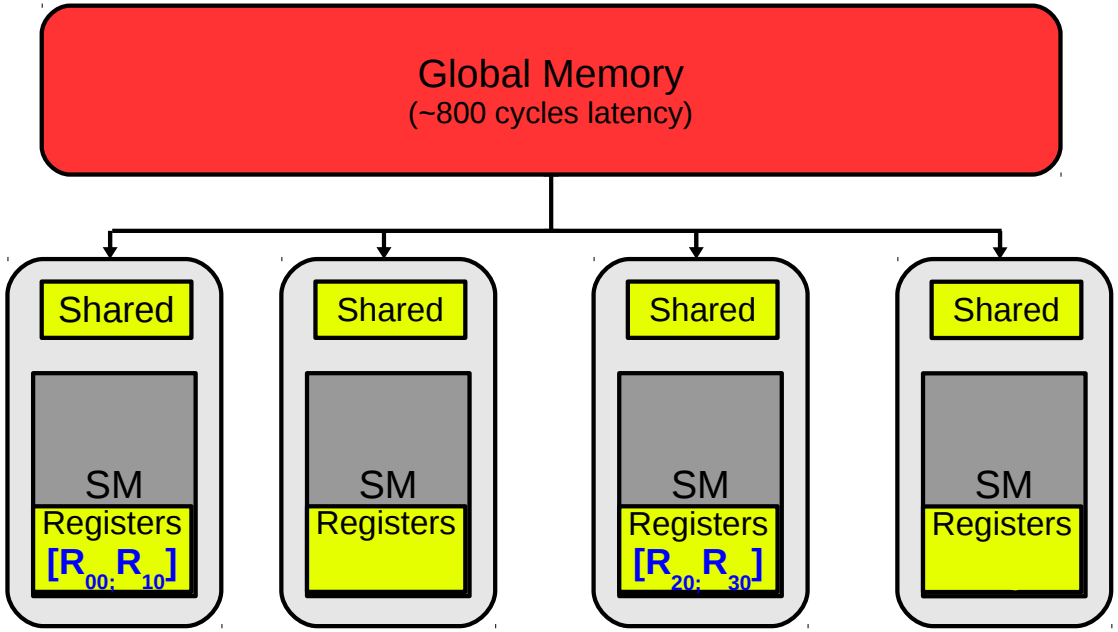
Why GPUs under-perform for CAQR?



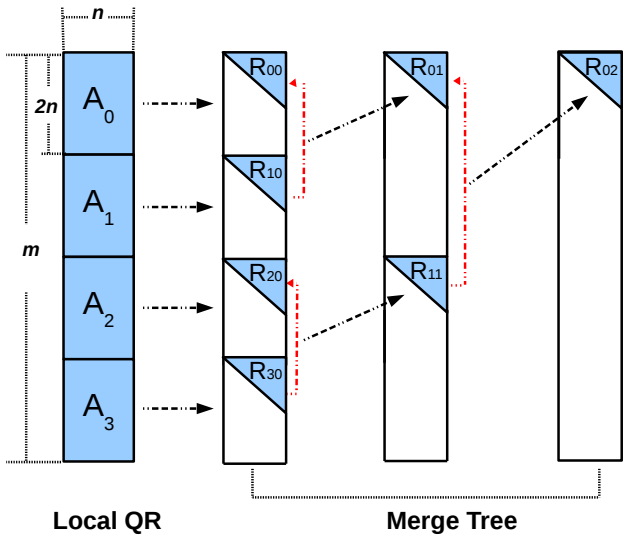


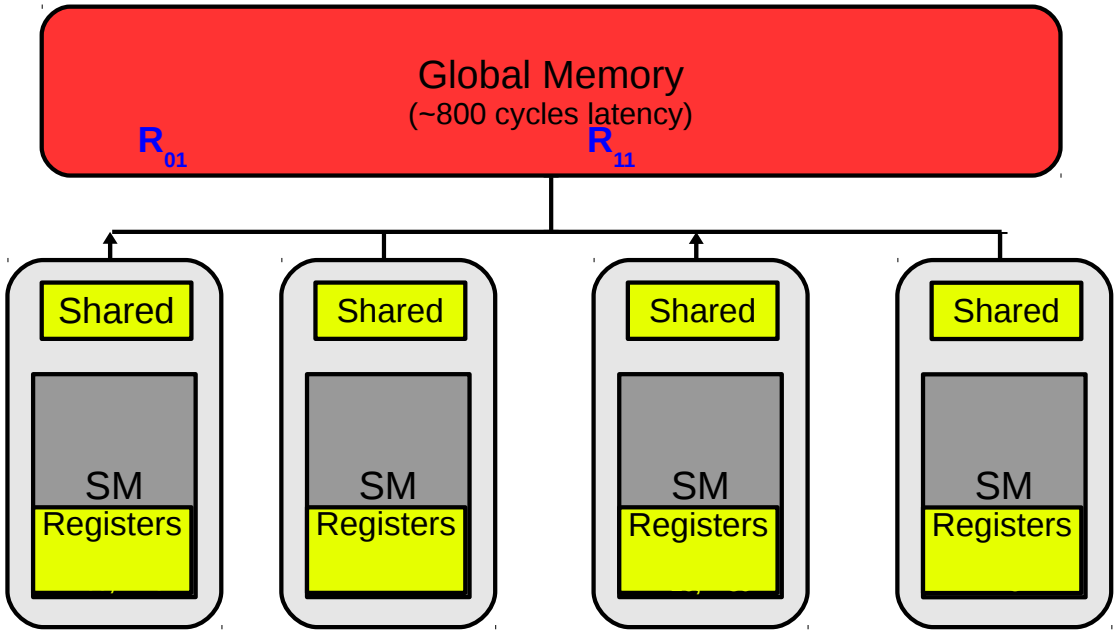
Why GPUs under-perform for CAQR?



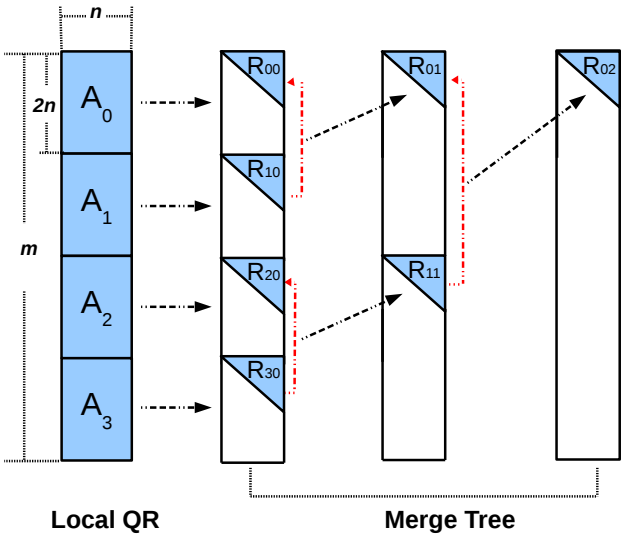


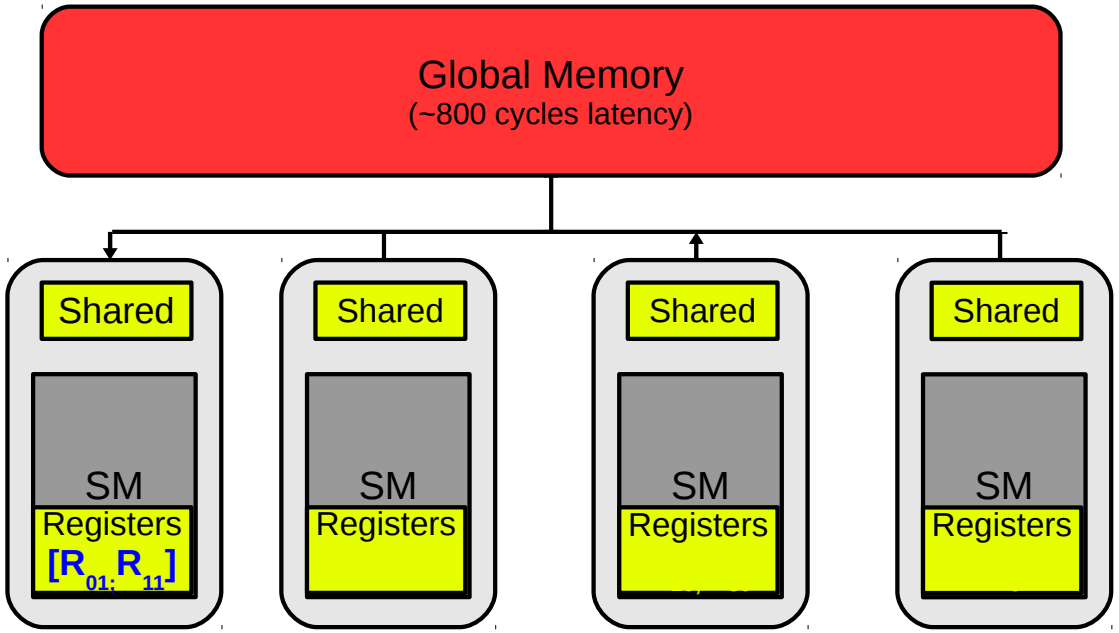
Why GPUs under-perform for CAQR?





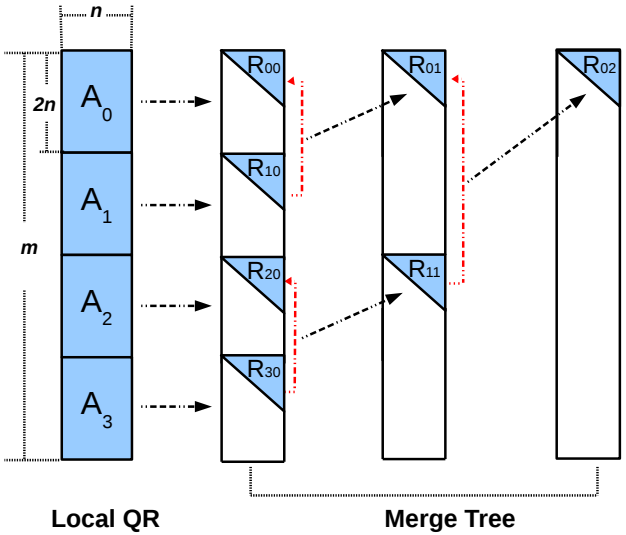
Why GPUs under-perform for CAQR?

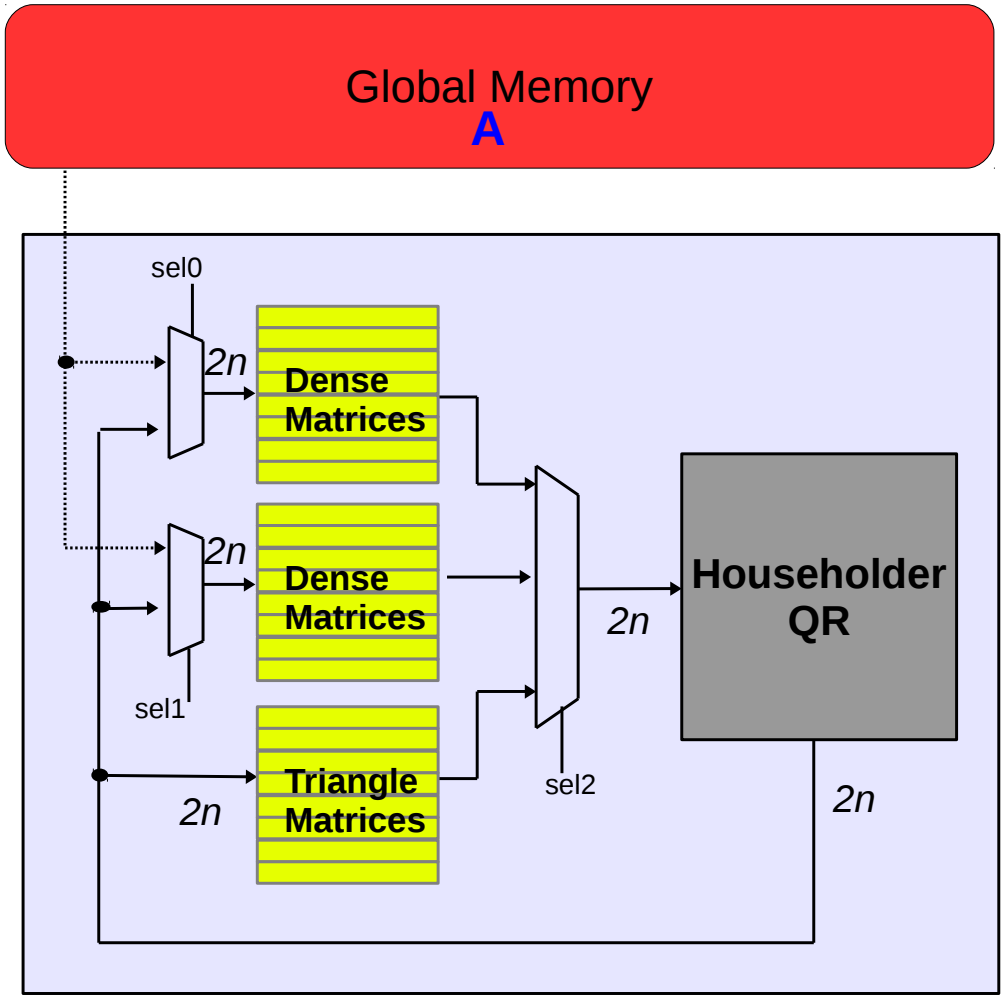




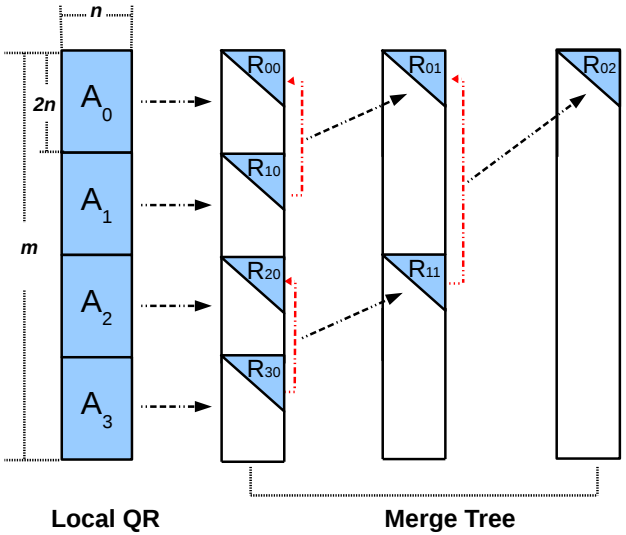
- **Small register file** size lead to low arithmetic intensity
- **Global communication** in merge stage leads to high lateny
- **Low utilization of SMs** as few are active in successive merge stages

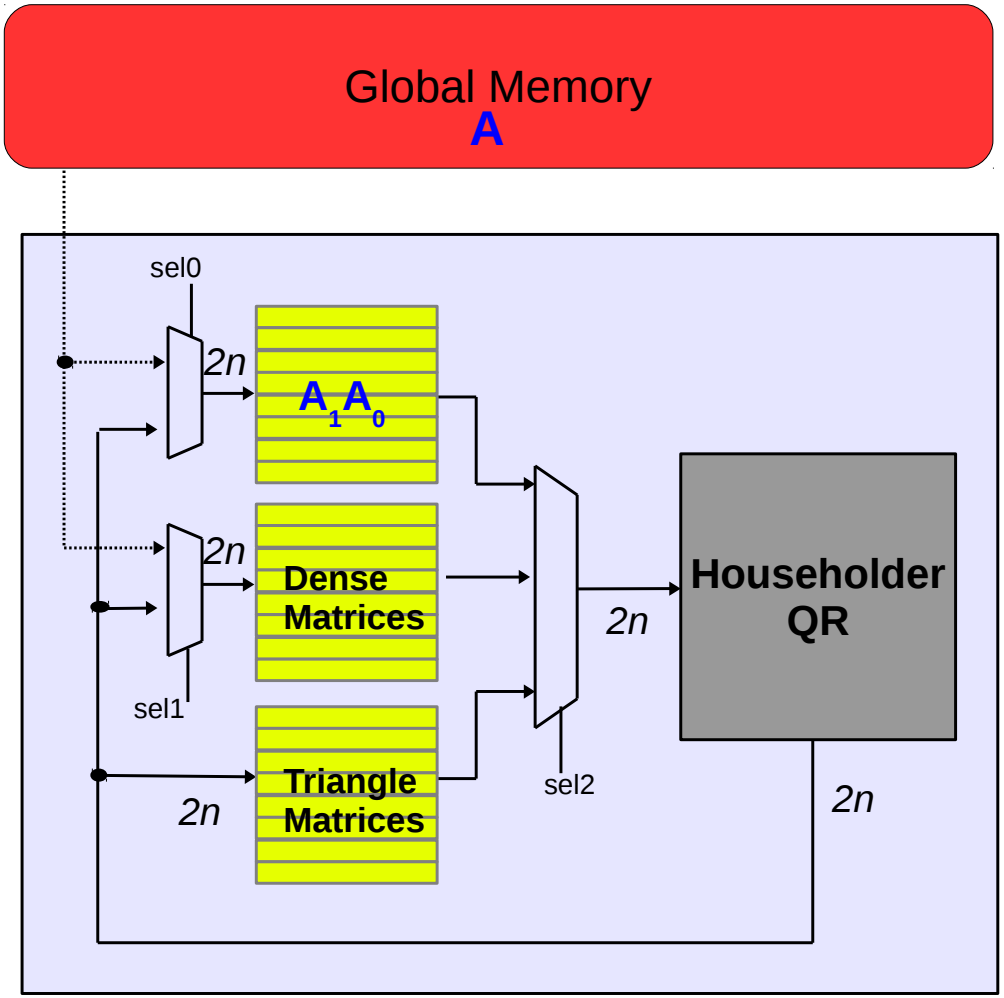
How do we enhance performance using FPGAs?



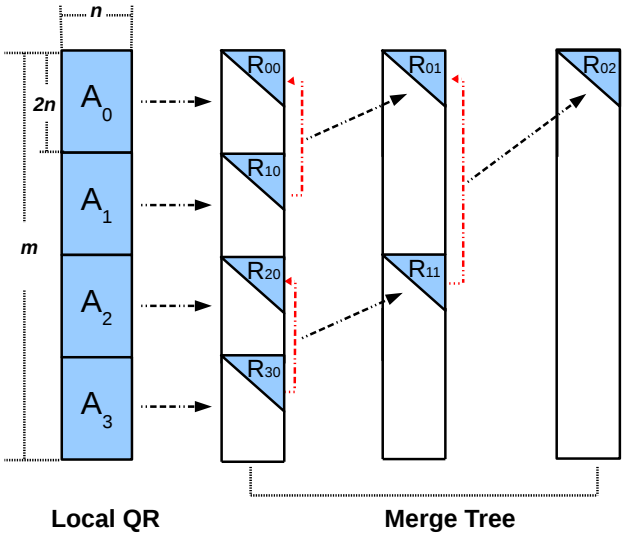


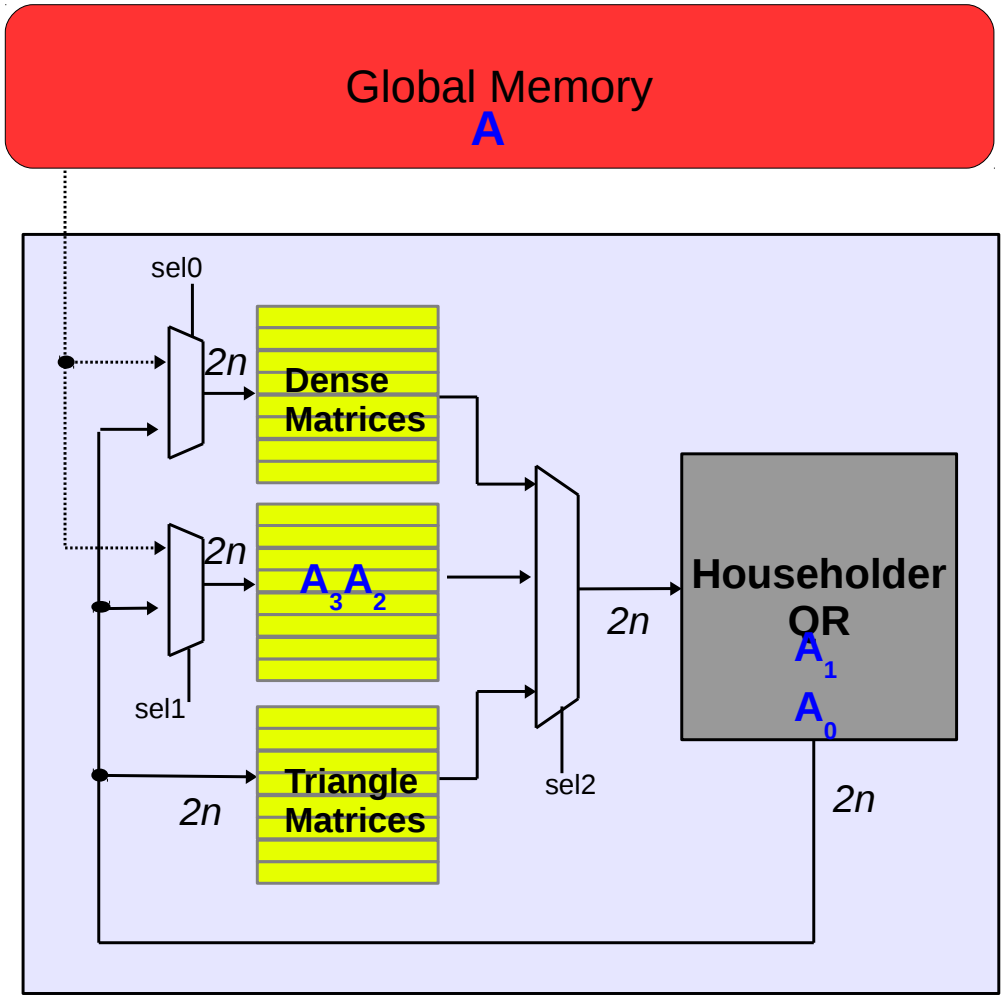
How do we enhance performance using FPGAs?



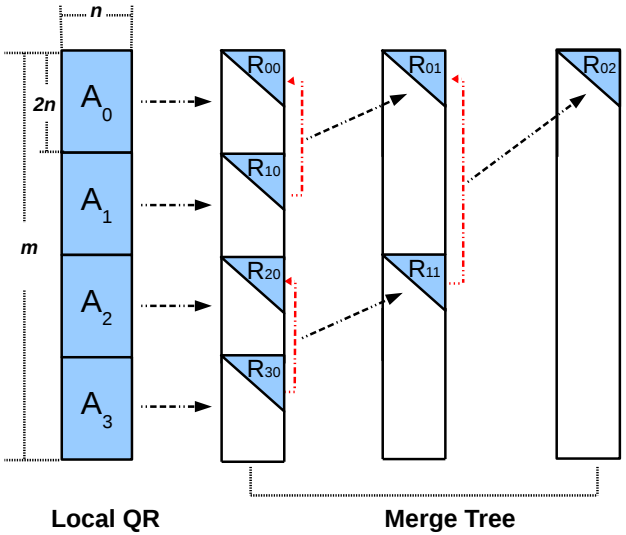


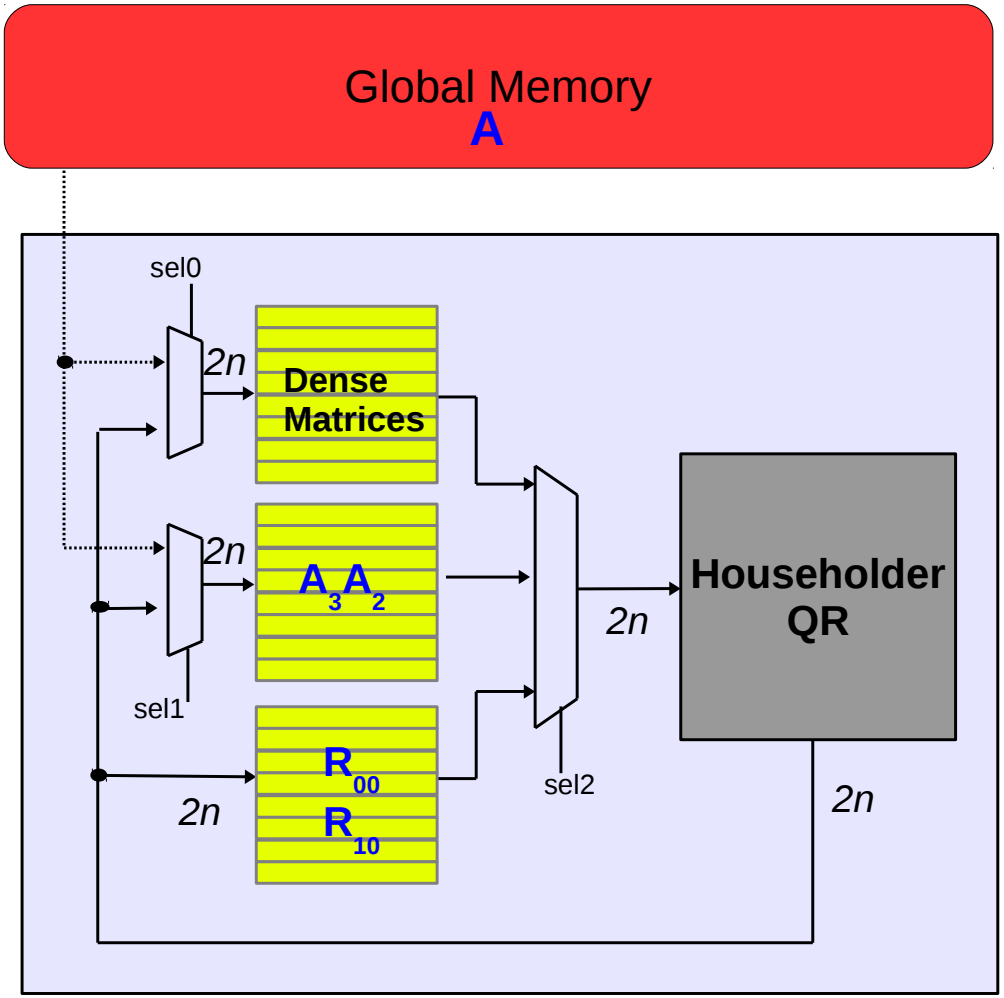
How do we enhance performance using FPGAs?



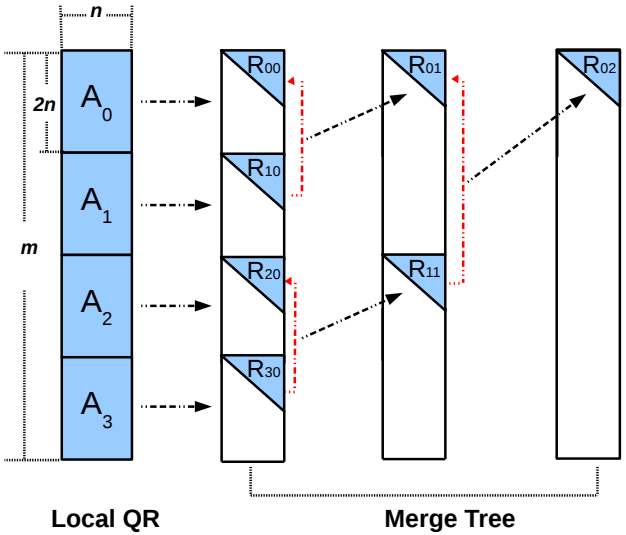


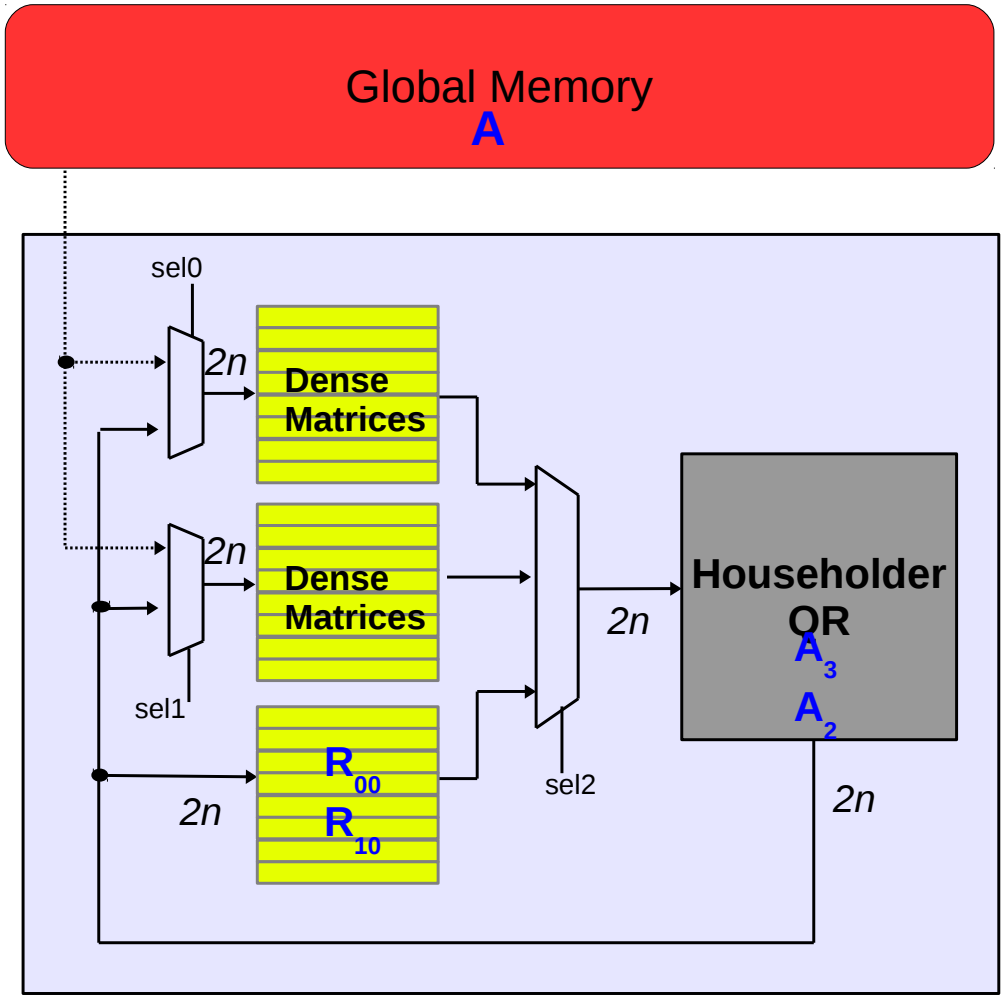
How do we enhance performance using FPGAs?



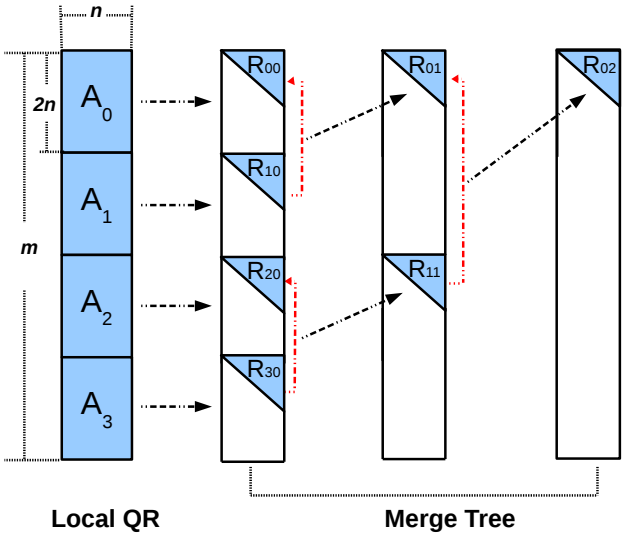


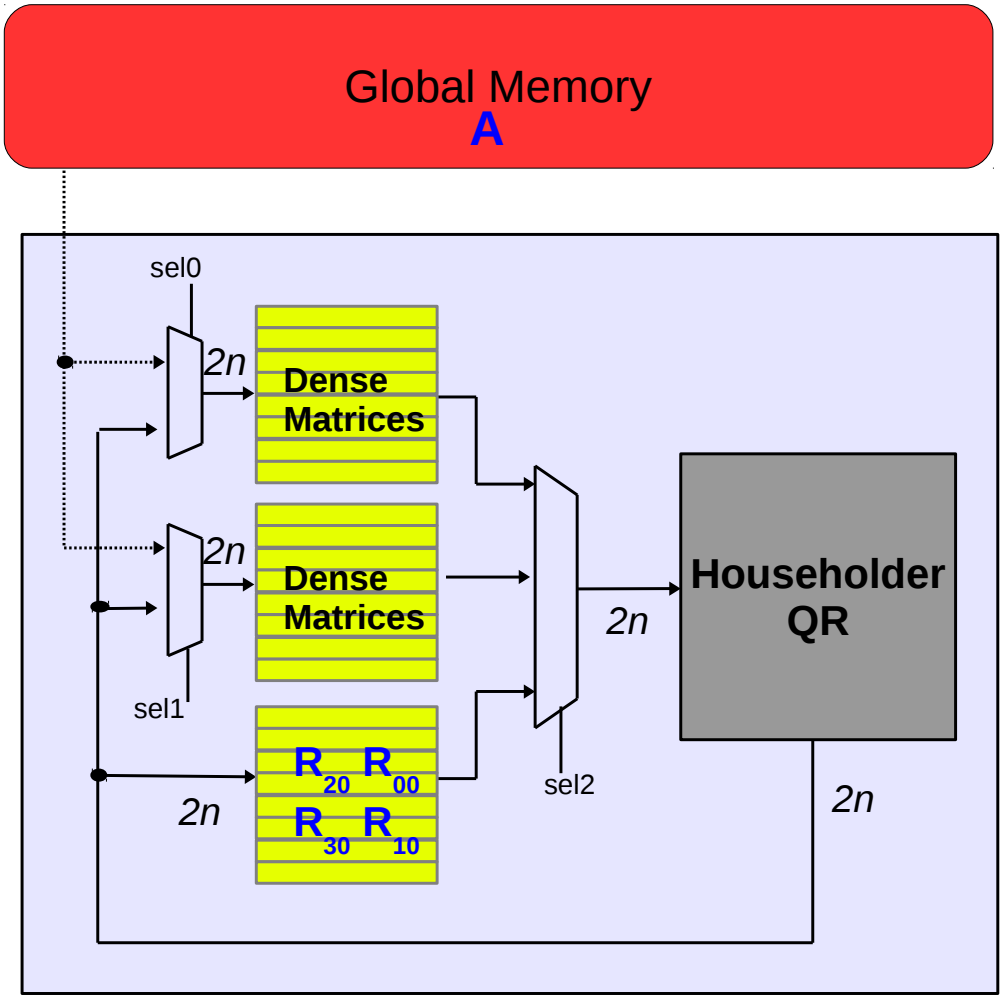
How do we enhance performance using FPGAs?



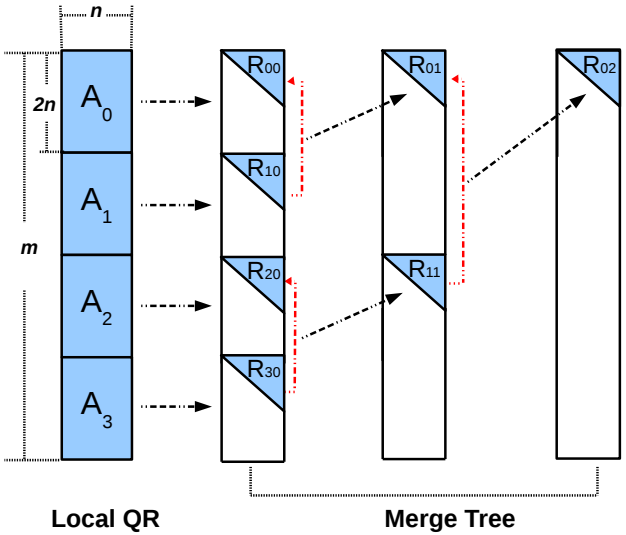


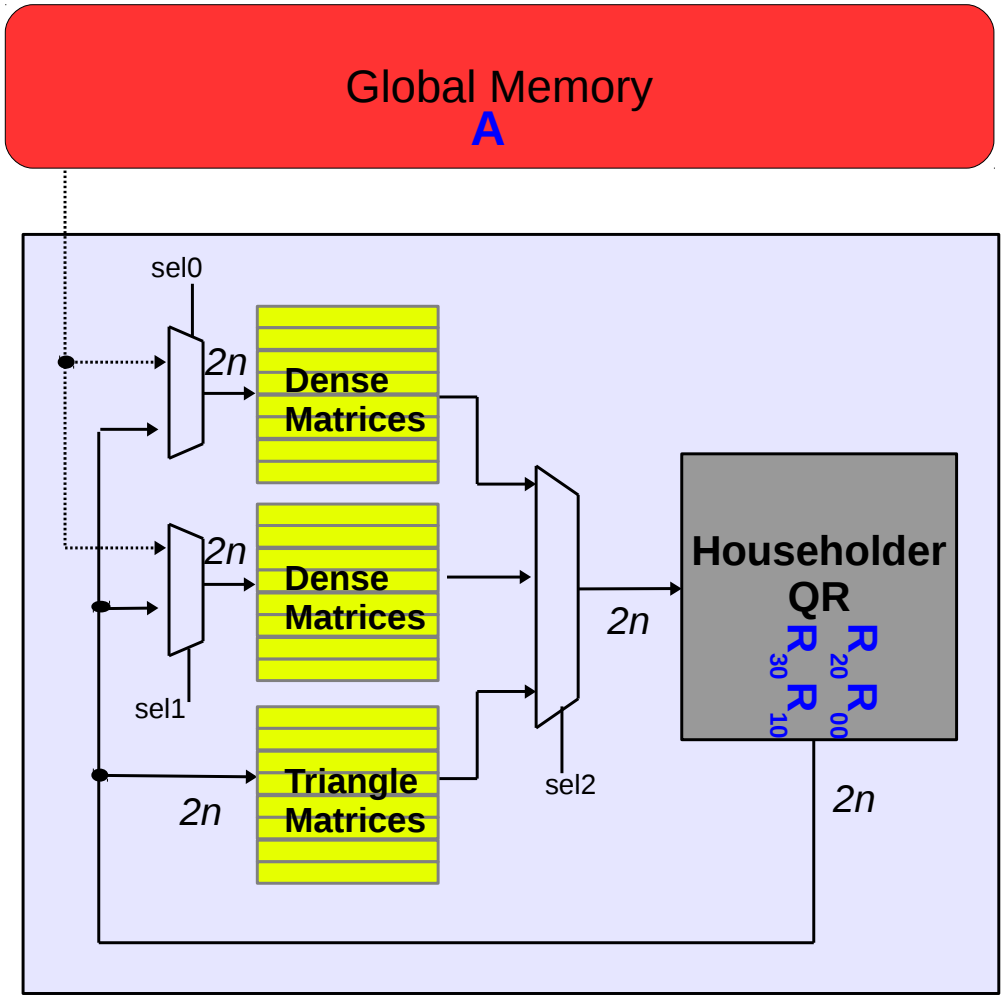
How do we enhance performance using FPGAs?



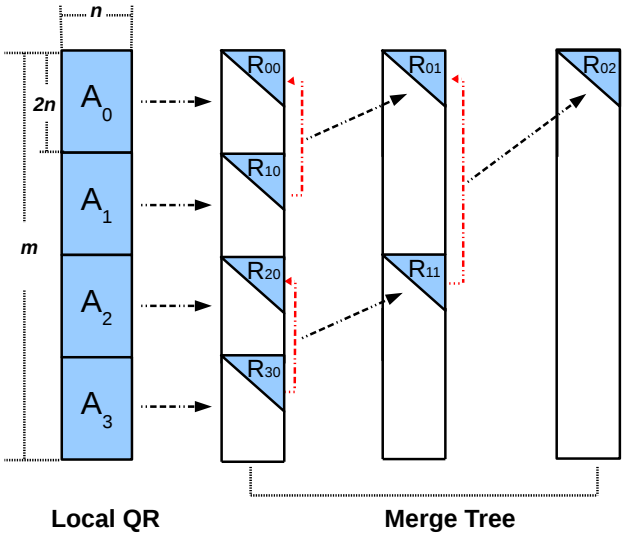


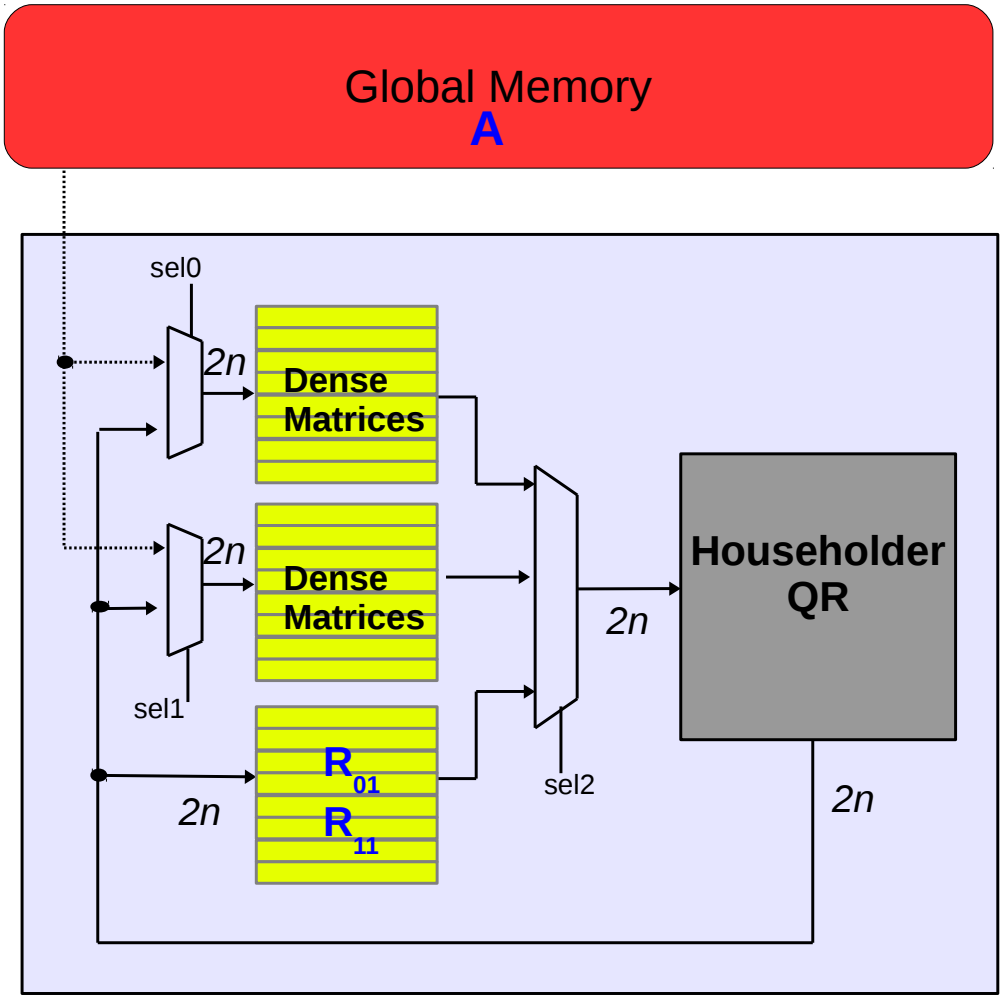
How do we enhance performance using FPGAs?



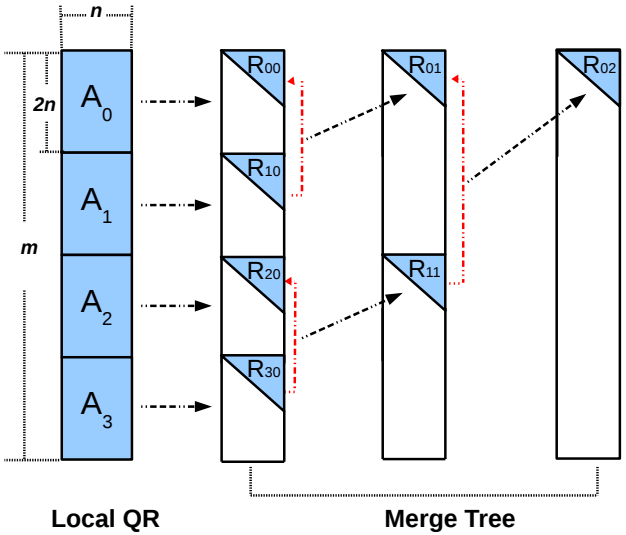


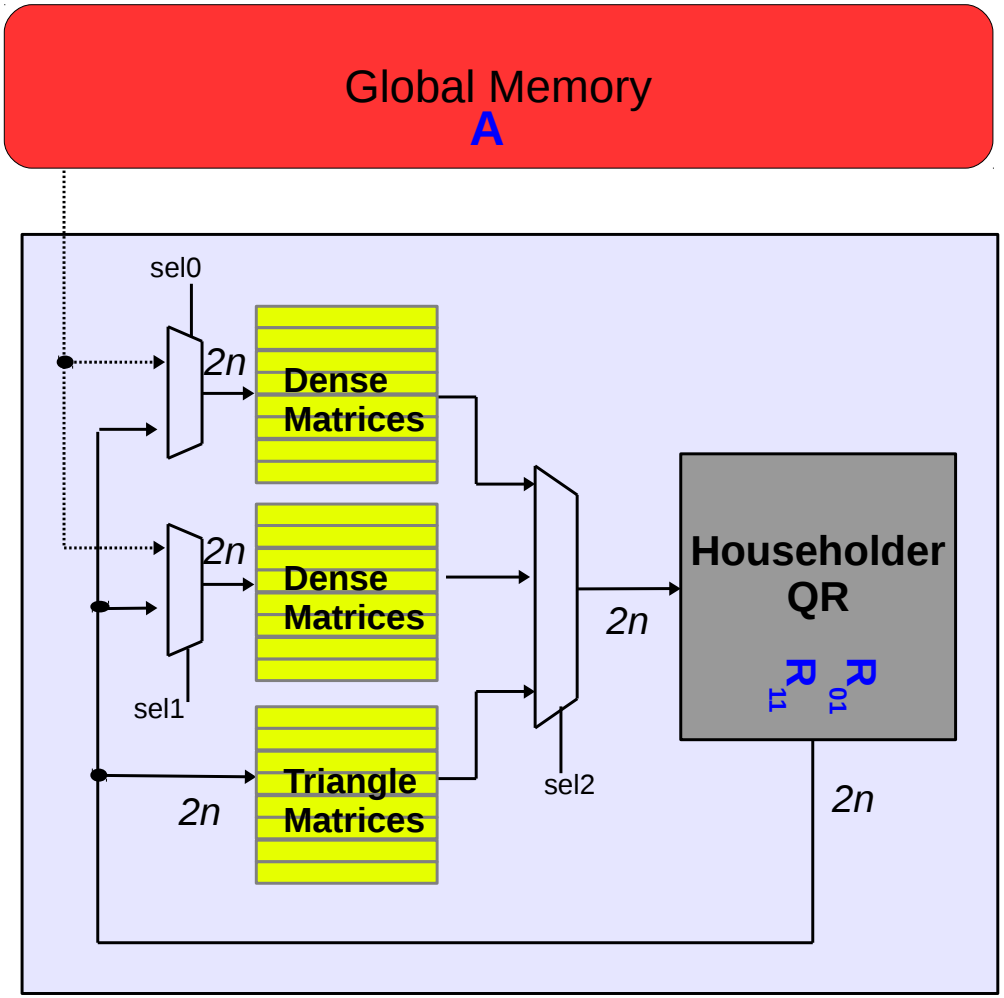
How do we enhance performance using FPGAs?



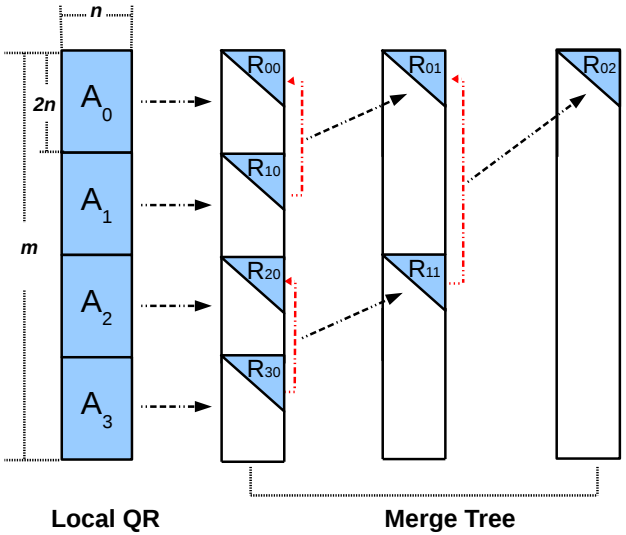


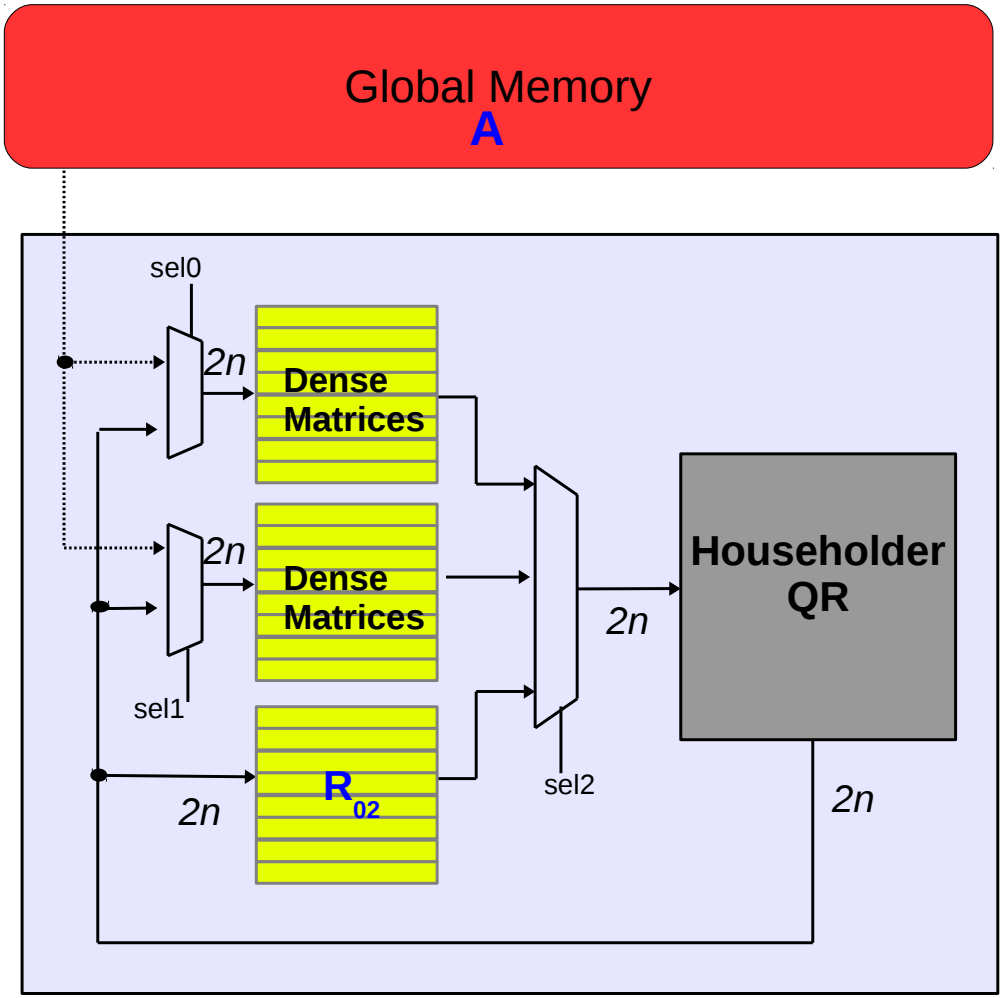
How do we enhance performance using FPGAs?





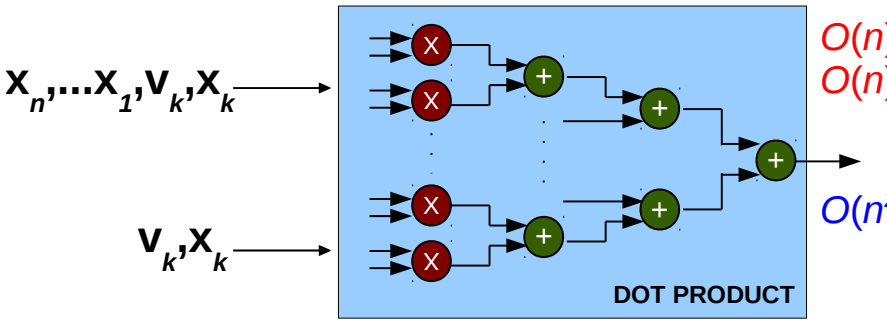
How do we enhance performance using FPGAs?





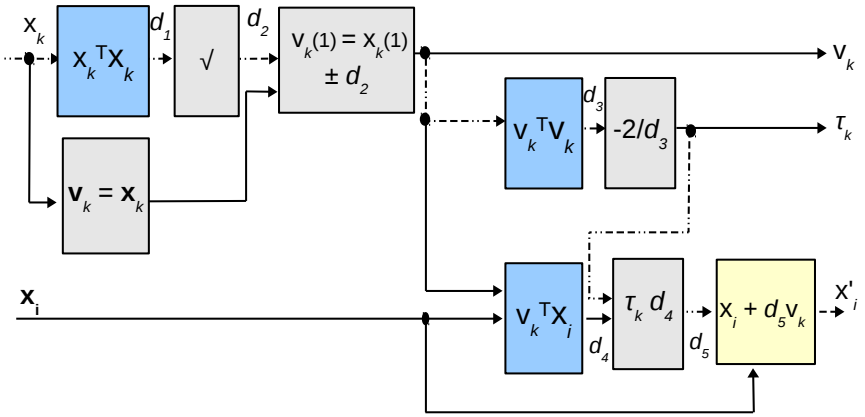
- **Large Scratch Pad Memory** arithmetic intensity $\sim 16\times$
- **Local communication**
- **High utilization** due to pipeline parallelism

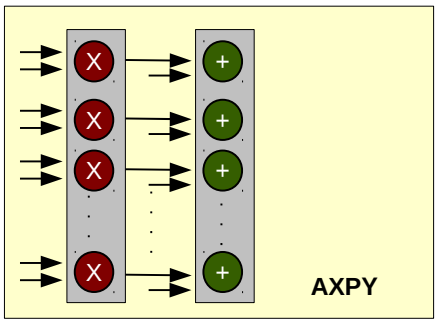
Householder QR Datapath



$O(n)$ Resources
$O(n)$ MemPorts

$O(n^2)$ Latency

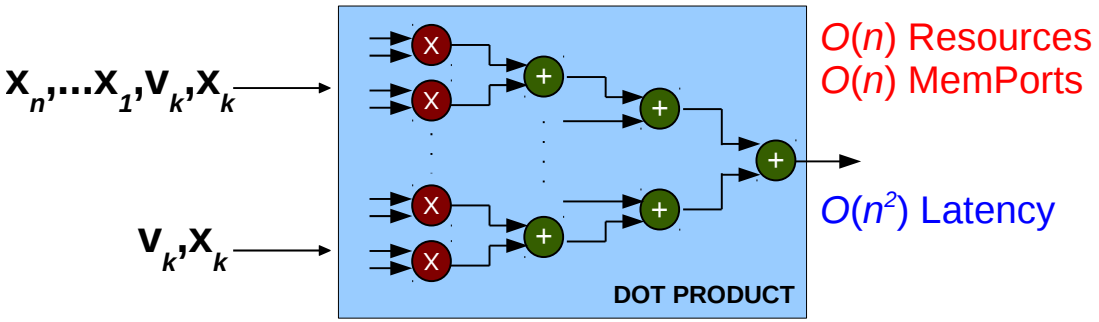




$O(n)$ Resources
$O(n)$ MemPorts

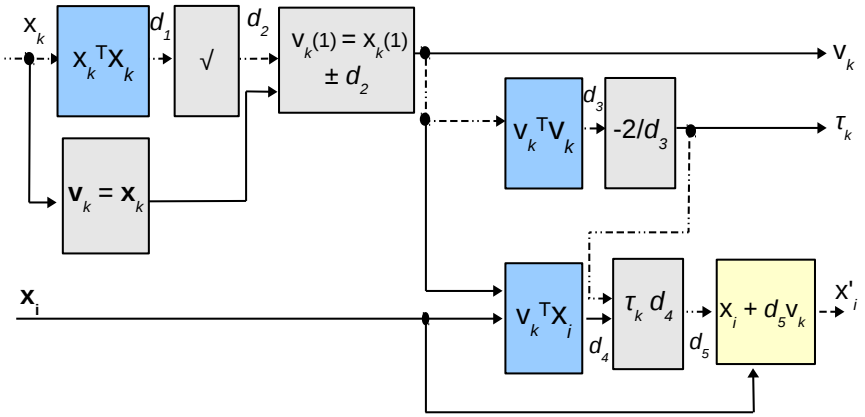
$O(1)$ Latency

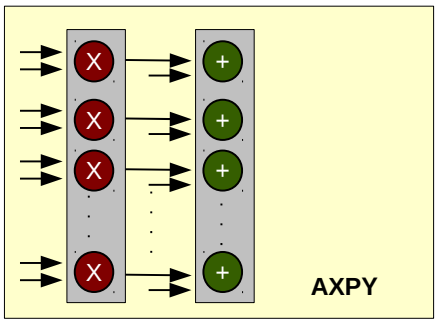
Householder QR Datapath



$O(n)$ Resources
$O(n)$ MemPorts

$O(n^2)$ Latency





$O(n)$ Resources
$O(n)$ MemPorts

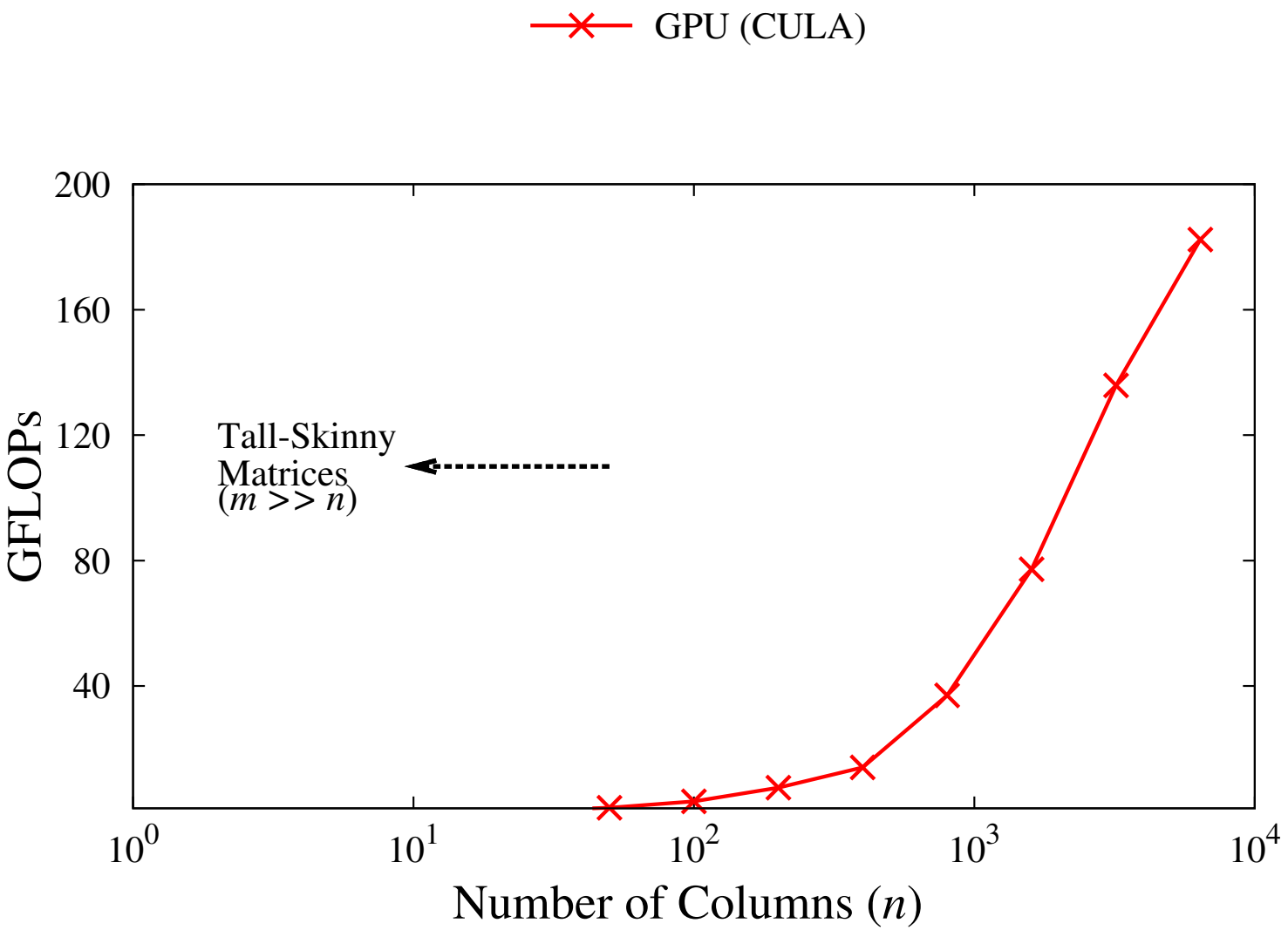
$O(1)$ Latency

- Utilizing **high memory bandwidth** and **full resource utilization** of FPGA we get 129 Peak GFLOPs (Double-Precision) on Virtex6-SX475T.

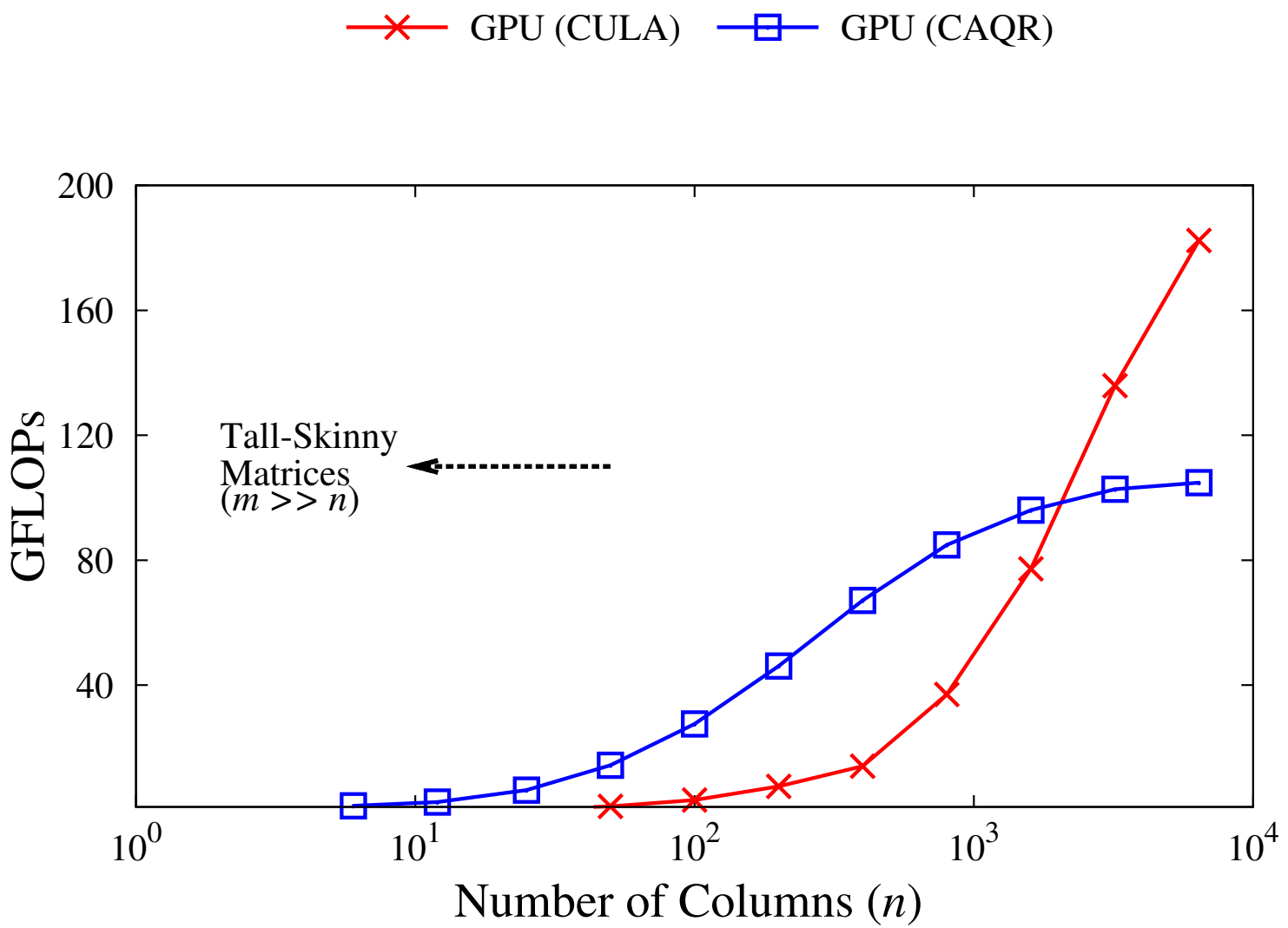
Experimental Setup

- Nvidia C2050 Fermi **(515 Peak Double Precision GFLOPs)**.
  - CULA QR Routine
  - CAQR Routine from UC Berkeley (Thanks to Michael Anderson).
- Xilinx Virtex-6 SX475T with Xilinx Coregen Library**(225 Peak Double Precision GFLOPs)**.
  - Synthesizing Tiled Matrix Decomposition on FPGAs. Tai *et.al.* (*FPL 2011*)

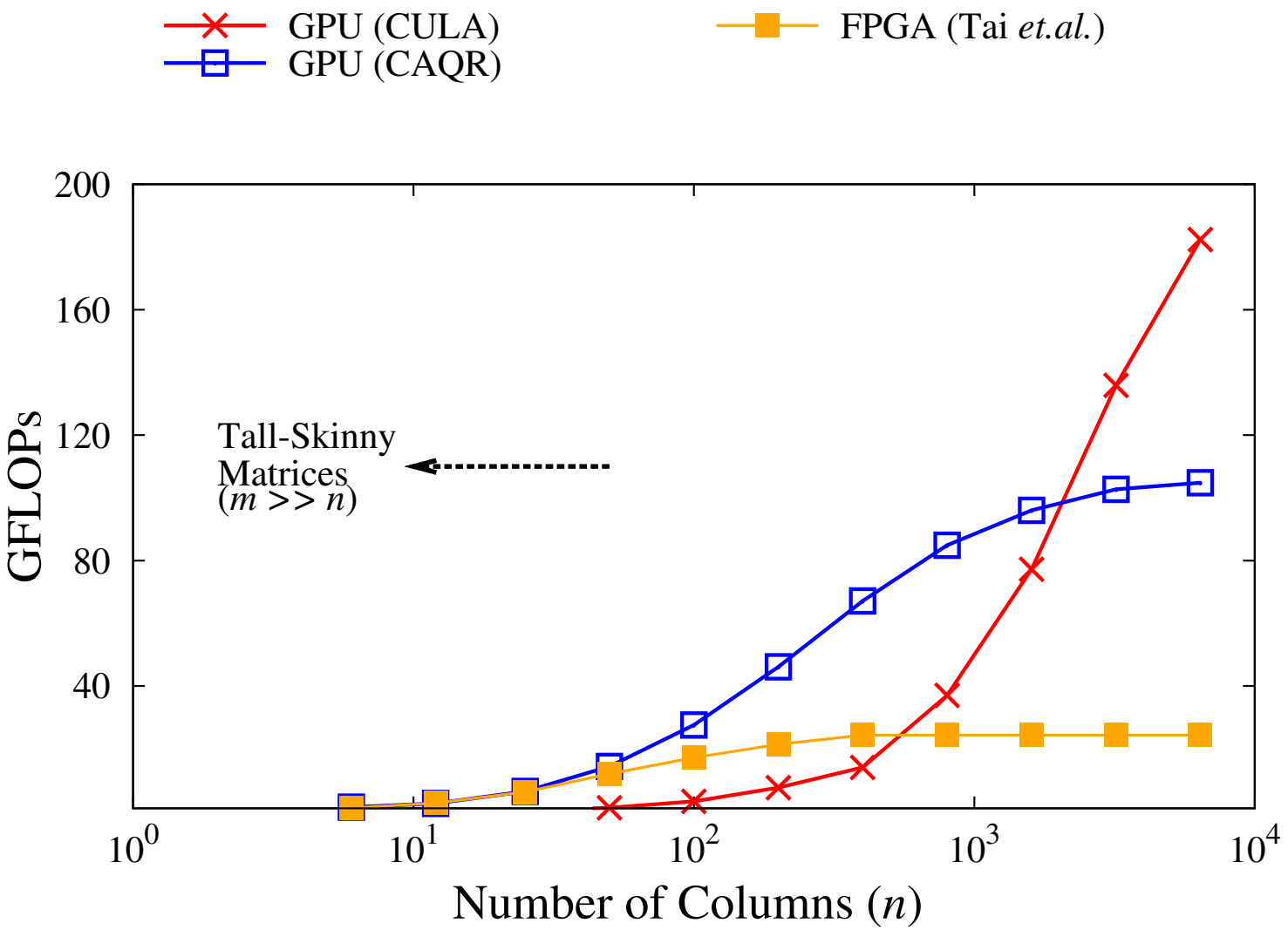
FPGAs Performance vs. Matrix Width ($m = 6400$)



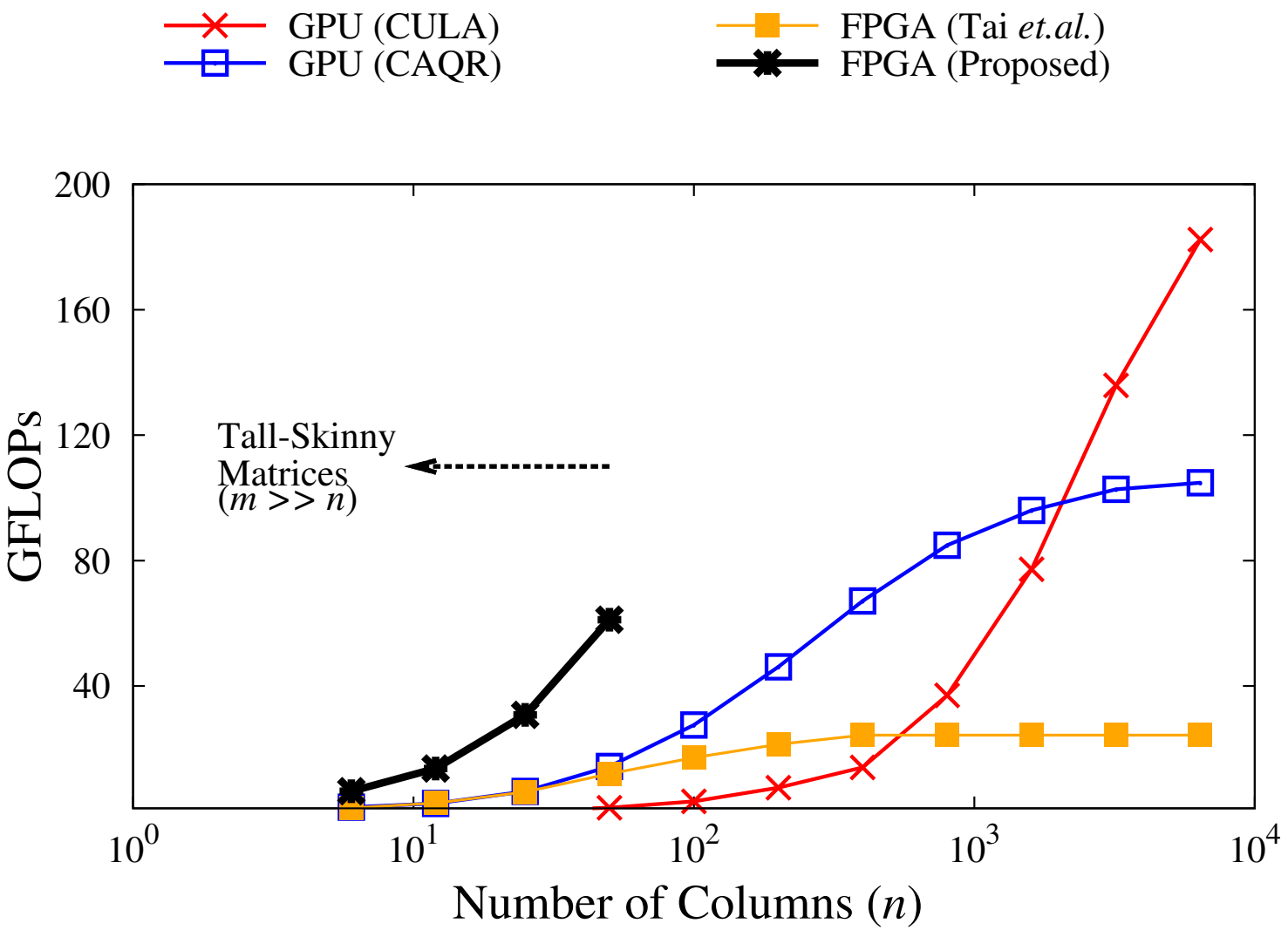
FPGAs Performance vs. Matrix Width ($m = 6400$)



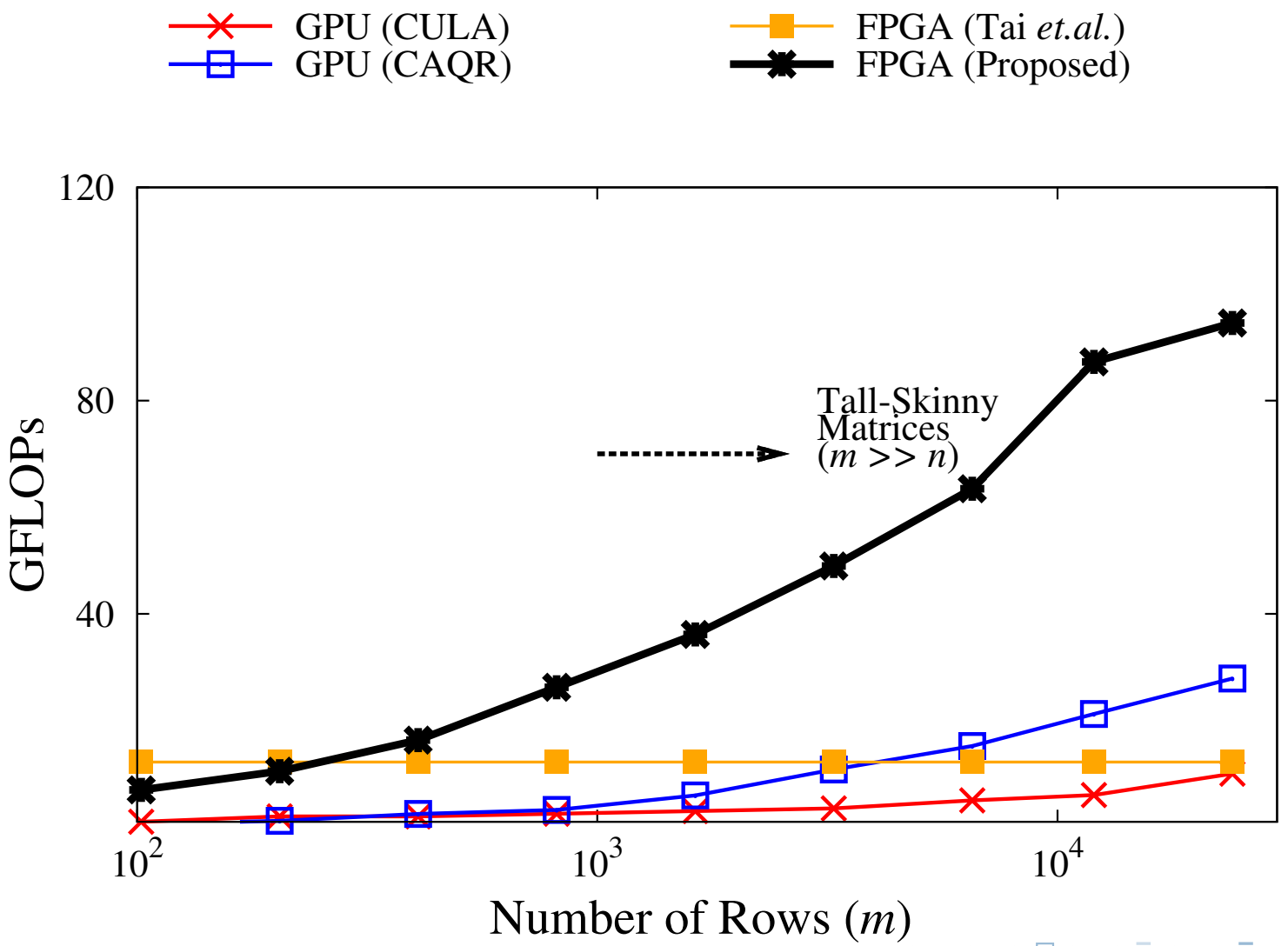
FPGAs Performance vs. Matrix Width ($m = 6400$)



FPGAs Performance vs. Matrix Width ($m = 6400$)



FPGAs Performance vs. Matrix Height ($n = 51$)



Conclusions

- The communication-bound tall-skinny QR factorization becomes compute-bound with communication-avoiding linear algebra.
- Tall-skinny QR performance is good on FPGAs compared to GPUs
  - **Large scratch pad memory** vs. Register file
  - **Local communication** vs. Global communication in merge stage
  - **High utilization of floating point cores** vs. low utilization of SMs during the irregular merge stage
- Comparison
  - 4.5× speed up over CAQR
  - 7.7× speed up over previous FPGA work
  - Much higher speed up factors over Intel MKL and CULA QR routines.

Questions?