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

\[
\begin{array}{c}
\text{Video} \\
\downarrow
\end{array}
\begin{array}{c}
\text{Stationary Background} \\
\uparrow
\end{array}
\begin{array}{c}
\text{Moving Object}
\end{array}
\]
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

  ![Video Frame 1]  =  ![Video Frame 2]  +  ![Background Frame]

- 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

\[
M = \begin{pmatrix}
\vdots & \vdots & \cdots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots \\
\end{pmatrix}
\]

\[m \times n = 110,592 \times 100\]

- 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

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

\[m \times n = 110,592 \times 100\]

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

Threshold these singular values

Converge?
Dense Linear Algebra - Tall-Skinny QR Factorization

- Tall-Skinny Matrix (M) - containing video frames as columns
  
  \[
  M = \begin{pmatrix}
  \vdots & \vdots & \cdots & \vdots & \vdots \\
  \vdots & \vdots & \cdots & \vdots & \vdots \\
  \vdots & \vdots & \cdots & \vdots & \vdots \\
  f_1 & f_2 & \cdots & f_{99} & f_{100} \\
  \vdots & \vdots & \cdots & \vdots & \vdots \\
  \vdots & \vdots & \cdots & \vdots & \vdots 
  \end{pmatrix}
  \]

  \(m \times n = 110,592 \times 100\)

- 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
Dense Linear Algebra - Tall-Skinny QR Factorization

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

\[
M = \begin{pmatrix}
... & ... & ... & ... & ...
& f_1 & f_2 & f_99 & f_{100}
& ... & ... & ... & ... & ...
& ... & ... & ... & ... & ...
& ... & ... & ... & ... & ...
\end{pmatrix}
\]

\[m \times n = 110,592 \times 100\]

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

CPU-Only \(\sim 10\) min
GPU + CPU \(\sim 17\) sec

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs
Dense Linear Algebra - Tall-Skinny QR Factorization

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

\[
\begin{pmatrix}
    \vdots & \vdots & \cdots & \vdots & \vdots \\
    \vdots & \vdots & \cdots & \vdots & \vdots \\
    \vdots & \vdots & \cdots & \vdots & \vdots \\
    f_1 & f_2 & \cdots & f_{99} & f_{100} \\
    \vdots & \vdots & \cdots & \vdots & \vdots \\
    \vdots & \vdots & \cdots & \vdots & \vdots \\
    \vdots & \vdots & \cdots & \vdots & \vdots \\
\end{pmatrix}
\]

\[m \times n = 110,592 \times 100\]

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

Threshold these singular values

CPU-Only \(\sim 10\) min
GPU + CPU \(\sim 17\) sec
FPGA + CPU \(\sim 2-3\) sec
Some Extreme Aspect Ratios of Tall-Skinny Matrices ....

<table>
<thead>
<tr>
<th>QR Application</th>
<th>Aspect Ratio ($\frac{m}{n}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background Subtraction</td>
<td>$\sim 1000$</td>
</tr>
<tr>
<td>Least Squares</td>
<td>$\sim 1000$</td>
</tr>
<tr>
<td>Iterative Solvers</td>
<td>$\sim 100,000$</td>
</tr>
</tbody>
</table>
What we want to achieve?

![Graph showing the number of GFLOPs vs. number of columns (n) for CULA (C2050). The graph illustrates an exponential increase in GFLOPs as the number of columns increases.]
What we want to achieve?

![Graph showing GFLOPs vs Number of Columns (n) with Tall-Skinny Matrices (m >> n) and CULA (C2050).]

- Tall-Skinny Matrices: $m >> n$
- CULA (C2050)

- Performance metrics:
  - 0.22%
  - 35%
What we want to achieve?

![Graph showing GFLOPs vs. Number of Columns (n) for Tall-Skinny Matrices (m >> n)](image)

- **CULA (C2050)**
- **CAQR (C2050)**

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs
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
**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

---

$k = 1$

\[
\begin{bmatrix}
\mathbf{x} & \mathbf{x} & \mathbf{x} \\
\mathbf{x} & \mathbf{x} & \mathbf{x} \\
\mathbf{x} & \mathbf{x} & \mathbf{x} \\
\mathbf{x} & \mathbf{x} & \mathbf{x} \\
\mathbf{x} & \mathbf{x} & \mathbf{x} \\
\end{bmatrix}
\]

$A$
**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
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
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

$k = 3$

$$
R = P_{n-1}P_{n-2} \cdots P_1 A
$$

$$
Q = \overline{P_1} \overline{P_2} \cdots \overline{P_{n-2}} \overline{P_{n-1}}
$$
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

$\begin{align*}
  & v_k (1) = x_k (1) \\
  & v_k^T x_i \\
  & \tau_k d_4 \\
  & d_5 \\
  & x'_i + d_5 v_k
\end{align*}$

Processor $0$

$\begin{bmatrix}
  \times & \times & \times \\
  \times & \times & \times \\
  \times & \times & \times \\
  \times & \times & \times \\
  \times & \times & \times \\
\end{bmatrix}$

Processor $1$

$\begin{bmatrix}
  \times & \times & \times \\
  \times & \times & \times \\
  \times & \times & \times \\
  \times & \times & \times \\
  \times & \times & \times \\
\end{bmatrix}$

Processor $2$

$\begin{bmatrix}
  \times & \times & \times \\
  \times & \times & \times \\
  \times & \times & \times \\
  \times & \times & \times \\
  \times & \times & \times \\
\end{bmatrix}$

$A$

$k = 1$

$N_p = 3$
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

$\triangleright \quad x_k^T x_k - \log_2 N_p$
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**

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

\[ O(\log N_p) \]
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?

- Small register file size lead to low arithmetic intensity
- Global communication in merge stage leads to high latency
- Low utilization of SMs as few are active in successive merge stages
Why GPUs under-perform for CAQR?

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs
Why GPUs under-perform for CAQR?

- Small register file size lead to low arithmetic intensity
- Global communication in merge stage leads to high latency
- Low utilization of SMs as few are active in successive merge stages
Why GPUs under-perform for CAQR?

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

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs
Why GPUs under-perform for CAQR?

- **Small register file** size lead to low arithmetic intensity
- **Global communication** in merge stage leads to high latency
- **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?

Local QR

Merge Tree

Global Memory

A

Dense Matrices

Triangle Matrices

Householder QR

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs 13/18
How do we enhance performance using FPGAs?

- **Local QR**
- **Merge Tree**
- **Global Memory**
  - $A$

Diagram showing the integration of Local QR, Merge Tree, and Global Memory with operations like Dense Matrices, Triangle Matrices, and Householder QR.

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs
How do we enhance performance using FPGAs?

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs
How do we enhance performance using FPGAs?

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs
How do we enhance performance using FPGAs?
How do we enhance performance using FPGAs?

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs
How do we enhance performance using FPGAs?
How do we enhance performance using FPGAs?

Local QR

Merge Tree

Global Memory

A

Dense Matrices

Triangle Matrices

Householder QR

2n

sel0

2n

sel1

2n

sel2

2n

Enhancing Performance of Tall-Skinny QR Factorization 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

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs
Householder QR Datapath

$X_n, \ldots, X_1, V_k, X_k \rightarrow O(n)$ Resources

$V_k, X_k \rightarrow O(n) MemPorts$

DOT PRODUCT

$O(n^2)$ Latency

$\sqrt{v_k} = \frac{x_k}{d_3}$

$\tau_k = \frac{d_4}{d_2}$

$X_i' = X_i + d_5 v_k$

$O(n)$ Resources

$O(n) MemPorts$

$O(1)$ Latency

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs 14/18
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 ($m = 6400$)

Graph showing the performance in GFLOPs of Tall-Skinny Matrices ($m >> n$) using GPUs (CULA) and (CAQR). The graph illustrates the number of columns ($n$) on the x-axis and GFLOPs on the y-axis.
FPGAs Performance vs. Matrix Width ($m = 6400$)

- **GPU (CULA)**
- **GPU (CAQR)**
- **FPGA (Tai et.al.)**

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs
FPGAs Performance vs. Matrix Width ($m = 6400$)

Enhancing Performance of Tall-Skinny QR Factorization using FPGAs
FPGAs Performance vs. Matrix Height ($n = 51$)

- GPU (CULA)
- GPU (CAQR)
- FPGA (Tai et al.)
- FPGA (Proposed)

**Diagram Description:**
- The graph plots GFLOPs (Giga Floating Point Operations) on the y-axis against the number of rows ($m$) on the x-axis.
- Tall-Skinny Matrices ($m \gg n$) are noted with a dashed line.

**Analysis:**
- The proposed FPGA solution shows significantly higher performance as $m$ increases, outperforming both GPU solutions and the existing FPGA approach by Tai et al.
- The performance gap is particularly noticeable as the number of rows increases, indicating potential advantages of FPGA architectures for these types of computations.
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?