Lab B:

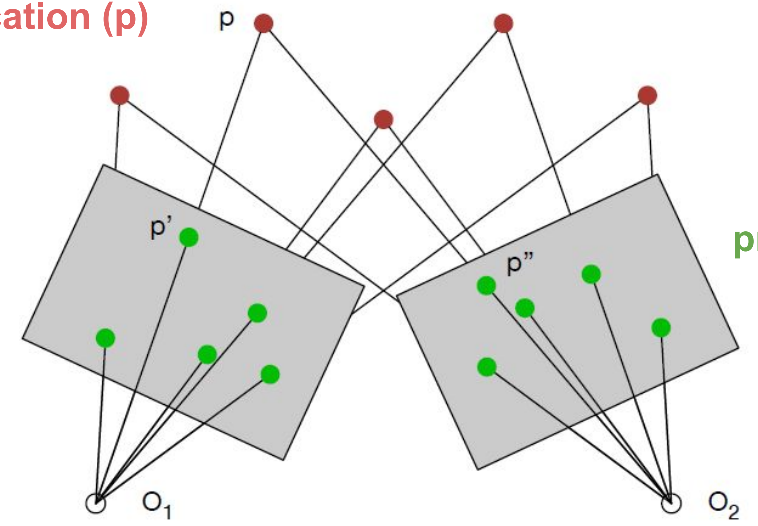
5-Point Relative Pose Problem

Team 7 吴宜凡

5-Point Relative Pose Problem

3D location (p)

calibrated views
(focal length, …)



projection (p, p’)

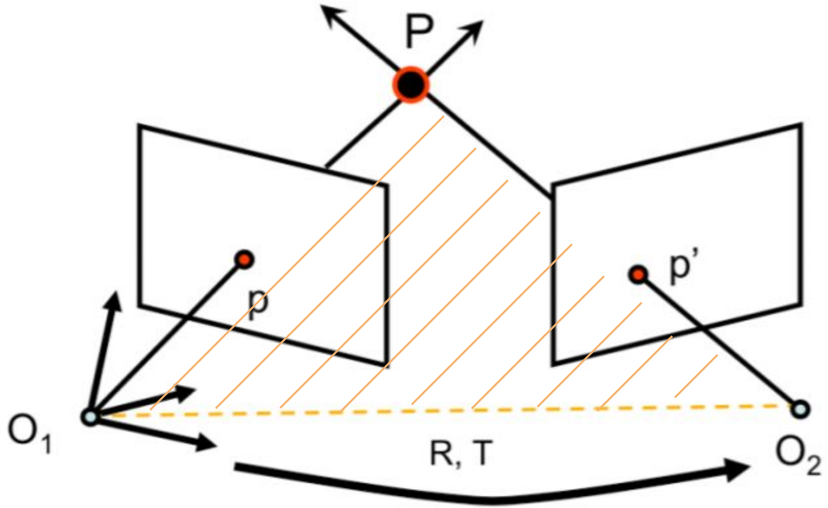
camera centres (of symmetry) (o1, o2)

translation + rotation

Epipolar Geometry

p, o1, o2 forms an **epipolar plane**

p, p' must be on the epipolar lines



The 5-Point Algorithm

- With these basic understandings, we can derive a compact expression for the **epipolar constraint**

$$q'^{\top} E q = 0,$$

  where **q**, **q'** are the image points and ***E*** is the **essential matrix**.
- The constraint can be rewritten as

$$\tilde{q}^{\top} \tilde{E} = 0,$$

where

$$\tilde{q} \equiv [q_1 q'_1 \;\; q_2 q'_1 \;\; q_3 q'_1 \;\; q_1 q'_2 \;\; q_2 q'_2 \;\; q_3 q'_2 \;\; q_1 q'_3 \;\; q_2 q'_3 \;\; q_3 q'_3]^{\top}$$
$$\tilde{E} \equiv [E_{11} \;\; E_{12} \;\; E_{13} \;\; E_{21} \;\; E_{22} \;\; E_{23} \;\; E_{31} \;\; E_{32} \;\; E_{33}]^{\top}.$$

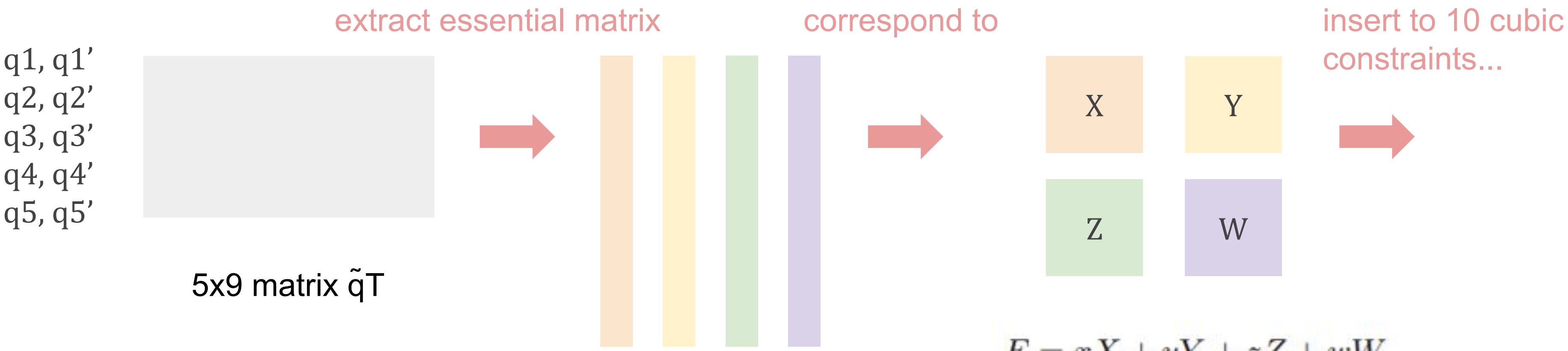
5-Point Algorithm

- Stacking vectors of the 5 points we obtain a **5x9 matrix** to compute **four vectors** that span the right null space of this matrix, which directly corresponds to four 3x3 matrices **X, Y, Z, W** of the form (assume $w = 1$)

$$E = xX + yY + zZ + wW$$

- Methodologies
  - Singular value decomposition (SVD)
  - QR factorisation (more efficient)
- Perform Gauss-Jordan elimination with partial pivoting we obtain a polynomial system A in 10 equations, and we can derive a 3x3 matrix B containing polynomial expressions only in variable z
- From det(B) = 0

5-Point Algorithm



$$E = xX + yY + zZ + wW$$

Vectors that span the null space of q̃T

Computed using

- Singular value decomposition (SVD)
- QR factorisation (more efficient)

5-Point Algorithm (cont.)

insert to 10 cubic constraints...

A	x^3	y^3	x^2y	xy^2	x^2z	x^2	y^2z	y^2	xyz	xy	x	y	1
$\langle a \rangle$	1	.	.	.	.	.	.	.	.	.	[2]	[2]	[3]
$\langle b \rangle$		1	.	.	.	.	.	.	.	.	[2]	[2]	[3]
$\langle c \rangle$			1	.	.	.	.	.	.	.	[2]	[2]	[3]
$\langle d \rangle$				1	.	.	.	.	.	.	[2]	[2]	[3]
$\langle e \rangle$					1						[2]	[2]	[3]
$\langle f \rangle$						1					[2]	[2]	[3]
$\langle g \rangle$							1				[2]	[2]	[3]
$\langle h \rangle$								1			[2]	[2]	[3]
$\langle i \rangle$									1		[2]	[2]	[3]
$\langle j \rangle$										1	[2]	[2]	[3]

system **A**

Gauss-Jordan elimination w/ partial pivoting

B	x	y	1
$\langle k \rangle$	[3]	[3]	[4]
$\langle l \rangle$	[3]	[3]	[4]
$\langle m \rangle$	[3]	[3]	[4]

3x3 matrix **B** containing polynomial expressions in **z**

From det(B) = 0

z1, z2, ..., z10

Recover from

x, y, z

From $E = xX + yY + zZ + wW$

Essential matrix **E** (Rotation **R**, Translation **T**)

Solution: Nistér's 5-Point Algorithm

Pros

- Easily represented as a chain of well separated computational stages
- Nister's interesting suggestions for efficient software implementations for most portions of the algorithm

Limitations

- One-to-one mapping of computational steps to hardware is suboptimal
  - unbalanced computational load

> Deeply pipelined implementation, where coarse-grained steps are subdivided into granular dataflow modules

Proposed Architecture

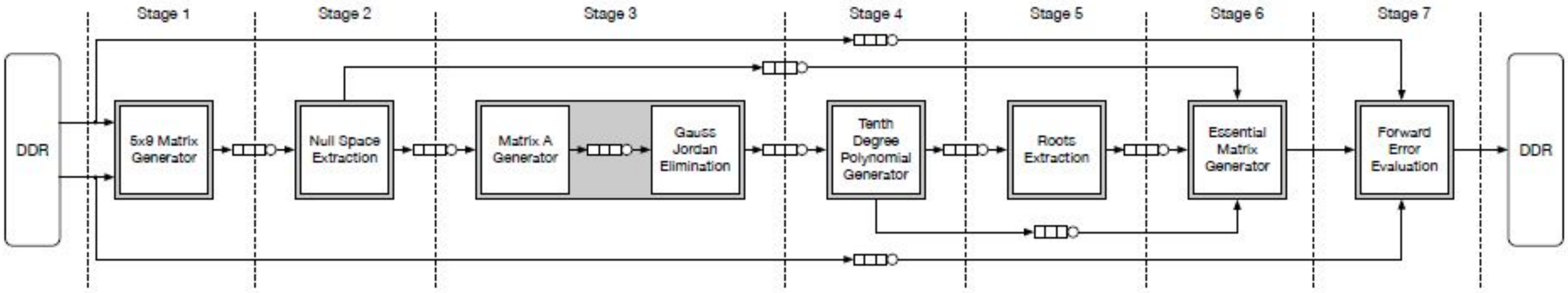


Figure 4.1: The computational pipeline that implements the 5-point algorithm. Each stage of the pipeline communicates with the others by means of FIFOs.

Implementation

- Kernel

```
void kernel(point* pts1_in, point* pts2_in, my_type* out, int iterations, my_type thresh){

  #pragma HLS INTERFACE m_axi depth=10 port=pts1_in bundle=gmem0
  #pragma HLS INTERFACE m_axi depth=10 port=pts2_in bundle=gmem1
  #pragma HLS INTERFACE m_axi depth=200 port=out bundle=gmem2

  #pragma HLS INTERFACE s_axilite register port=pts1_in bundle=control
  #pragma HLS INTERFACE s_axilite register port=pts2_in bundle=control
  #pragma HLS INTERFACE s_axilite register port=out bundle=control
  #pragma HLS INTERFACE s_axilite register port=iterations bundle=control
  #pragma HLS INTERFACE s_axilite register port=thresh bundle=control
```

Implementation

```
#pragma HLS DATAFLOW

    // 4 times the proper depth to make sure that we hide the time for memo
    hls::stream<point> pts1("pts1_stream"), pts2("pts2_stream");
#pragma HLS STREAM variable=pts1 depth=pts_depth dim=1
#pragma HLS STREAM variable=pts2 depth=pts_depth dim=1

    hls::stream<point> pts1_0("pts1_0_stream"), pts2_0("pts2_0_stream");
#pragma HLS STREAM variable=pts1_0 depth=pts_depth dim=1
#pragma HLS STREAM variable=pts2_0 depth=pts_depth dim=1

    hls::stream<point> pts1_1("pts1_1_stream"), pts2_1("pts2_1_stream");
#pragma HLS STREAM variable=pts1_1 depth=pts_1_depth dim=1
#pragma HLS STREAM variable=pts2_1 depth=pts_1_depth dim=1

    hls::stream<rType> r_stream("r_stream");
#pragma HLS STREAM variable=r_stream depth=r_stream_depth dim=1

    hls::stream<my_type> e_stream("e_stream");
#pragma HLS STREAM variable=e_stream depth=e_stream_depth dim=1
```

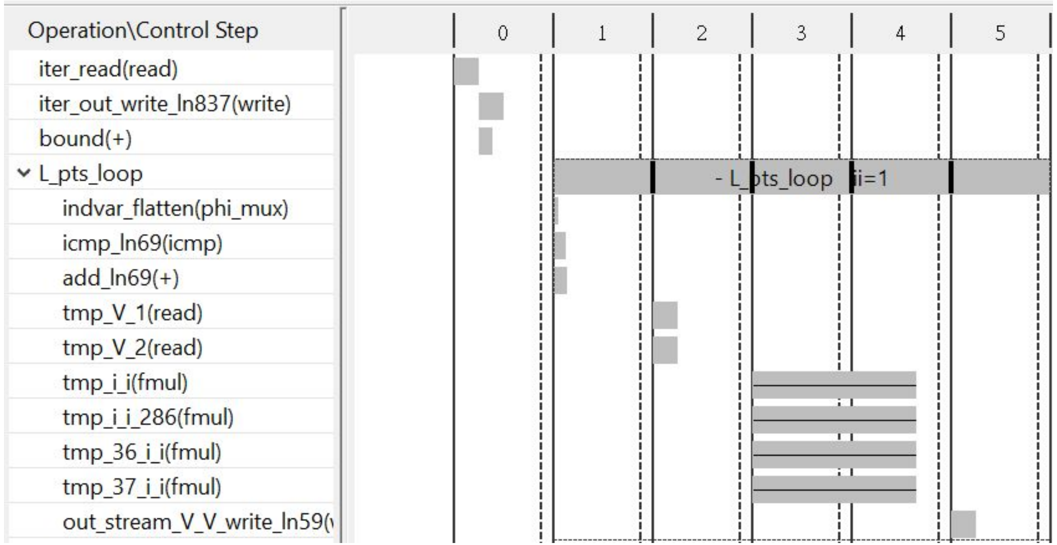
Enable task-level pipelining

Stage 1: Epipolar Constraints Matrix Generation

- Generation of the epipolar constraints matrix
- **Stage inner pipeline**: produces one row of the matrix per cycle

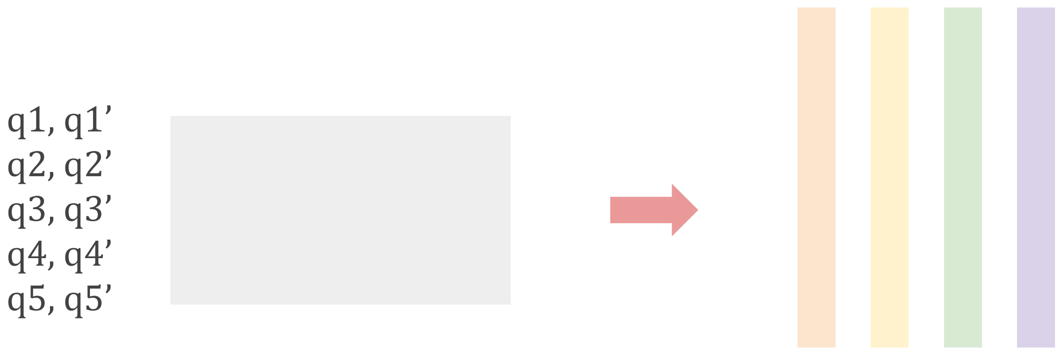
q1, q1’
q2, q2’
q3, q3’
q4, q4’
q5, q5’

5x9 matrix q̃T



Stage 2: Null Space Extraction

- Computes the null space of a 5x9 matrix leveraging a custom QR factorisation
- Decompose NxM matrix with QR factorisation
  - **Q**: NxN (5x5) orthogonal matrix
  - **R**: NxM (5x9) upper triangular matrix



5x9 matrix $\tilde{q}T$

Vectors that span the null space of $\tilde{q}T$

Stage 2: Null Space Extraction

- Tailored dataflow computational model for QR-f in HLS
  - Processes in **batches** the non-dependent array elements (b1-9)
  - Enqueue only the elements of the null space into the outgoing FIFOs (a 9x4 matrix H) (b10)

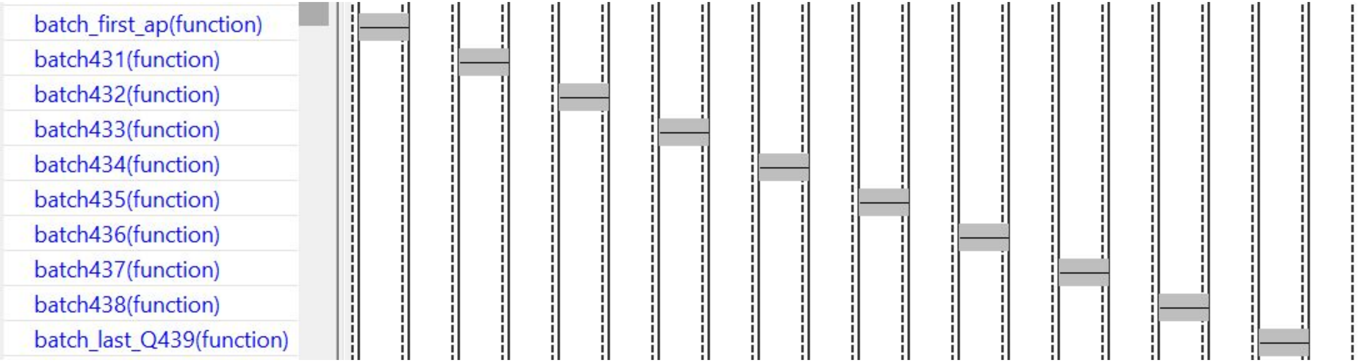
```
#pragma HLS STREAM variable=r_stream depth=RowsA*ColsA*2 dim=1

        batch_first_ap<4, RowsA, ColsA, false, rType, OutputType>(in_stream, q_stream[0], r_stream[0],
        batch<4, RowsA, ColsA, false, OutputType>(q_stream[0], r_stream[0], q_stream[1], r_stream[1],
        batch<4, RowsA, ColsA, false, OutputType>(q_stream[1], r_stream[1], q_stream[2], r_stream[2],
        batch<3, RowsA, ColsA, false, OutputType>(q_stream[2], r_stream[2], q_stream[3], r_stream[3],
        batch<3, RowsA, ColsA, false, OutputType>(q_stream[3], r_stream[3], q_stream[4], r_stream[4],
        batch<3, RowsA, ColsA, false, OutputType>(q_stream[4], r_stream[4], q_stream[5], r_stream[5],
        batch<3, RowsA, ColsA, false, OutputType>(q_stream[5], r_stream[5], q_stream[6], r_stream[6],
        batch<3, RowsA, ColsA, false, OutputType>(q_stream[6], r_stream[6], q_stream[7], r_stream[7],
        batch<2, RowsA, ColsA, false, OutputType>(q_stream[7], r_stream[7], q_stream[8], r_stream[8],
        batch_last_Q<1, RowsA, ColsA, RowsQ, true, OutputType>(q_stream[8], r_stream[8], out_stream, s
```

Stage 2: Null Space Extraction

- Batches work as a **pipeline** (individual latency ~430 clock cycles)

```
in_row_copy : for(int r=0; r<RowsA; r++){
    // Merge loops to parallelize the A input read and the Q matrix prime.
    #pragma HLS LOOP_MERGE force
    in_col_copy_q_i : for(int c=0; c<RowsA; c++) {
      #pragma HLS PIPELINE
      q_i[r][c] = q_stream_in.read();
    }
    in_col_copy_r_i : for(int c=0; c<ColsA; c++) {
      #pragma HLS PIPELINE
      r_i[r][c] = r_stream_in.read();
    }
  }
```



Stage 3: System A Generation

- Null space H - polynomial **system A** (10x20 matrix) - **system A*** (10x10 matrix) *via Gauss-Jordan elimination method and partial pivoting*
- Original: full equation implementation w/ target latency of 680 clock cycles
  - Large # registers to store temp values
  - Complex state machine
- Optimisations
  - **Polynomial expressions reduction**
    Leverage sub-expression reuse
  - **Minimal load/store parallel architecture**
    Reduce resource util. of original SM

A	x^3	y^3	x^2y	xy^2	x^2z	x^2	y^2z	y^2	xyz	xy	x	y	1
$\langle a\rangle$	1	.	.	.	.	.	.	.	.	.	[2]	[2]	[3]
$\langle b\rangle$		1	.	.	.	.	.	.	.	.	[2]	[2]	[3]
$\langle c\rangle$			1	.	.	.	.	.	.	.	[2]	[2]	[3]
$\langle d\rangle$				1	.	.	.	.	.	.	[2]	[2]	[3]
$\langle e\rangle$					1						[2]	[2]	[3]
$\langle f\rangle$						1					[2]	[2]	[3]
$\langle g\rangle$							1				[2]	[2]	[3]
$\langle h\rangle$								1			[2]	[2]	[3]
$\langle i\rangle$									1		[2]	[2]	[3]
$\langle j\rangle$										1	[2]	[2]	[3]

system **A***

Stage 3: System A Generation

- **Polynomial expressions reduction**
  - Observation: no 2 expressions have a common term
  - Find properly partitioned sub-expressions
- **Minimal load/store parallel architecture**

$$a_i = \sum_{l,m,n} c_{i,l,m,n} \cdot h_l \cdot h_m \cdot h_n$$

$$a_0 = h_0 h_1^2 + h_4^3 + 2h_0^2 h_3$$

$$a_1 = h_1^3 + 2h_0 h_1 h_3 + h_2^3$$

$$\downarrow$$

$$s_0 = h_1^2 + 2h_0 h_3$$

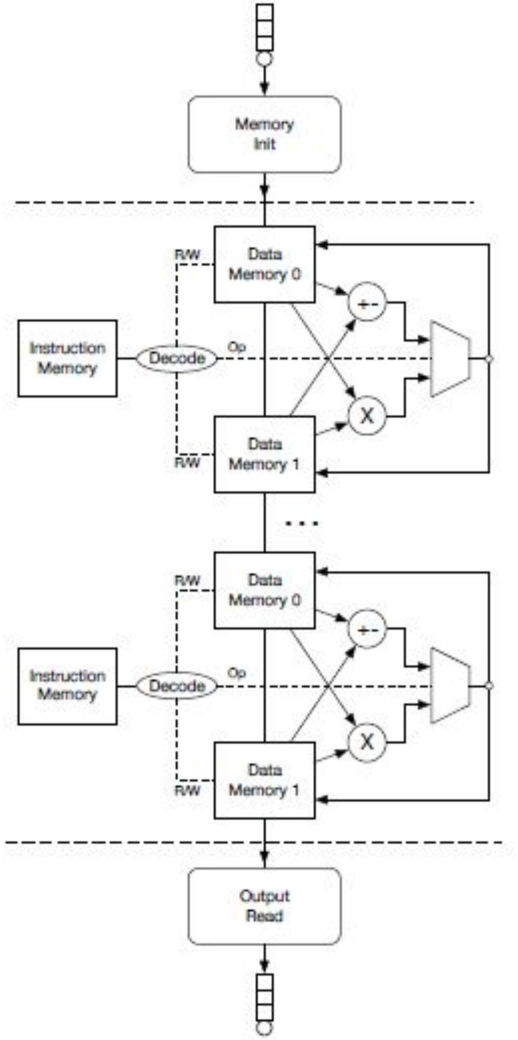
$$s_1 = h_4^3$$

$$s_2 = h_2^3$$

$$a_0 = h_0 s_0 + s_1$$

$$a_1 = h_1 s_0 + s_2$$

Reuse of sub-expressions



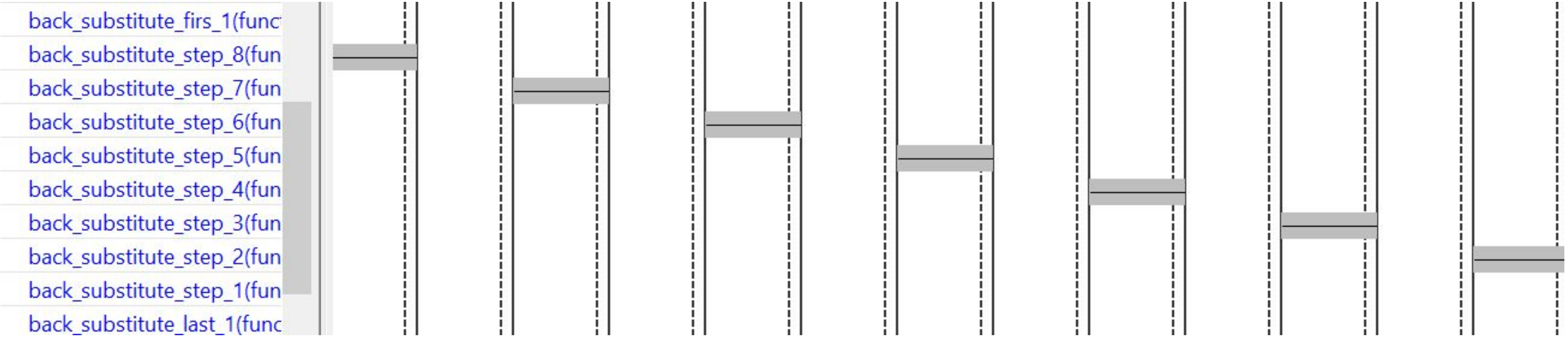
Stage 3: System A Generation

- Assign 10 load/store cores for the generation of each submatrix
- Reimplement matrix inverse and multiply computations available in Vivado HLS

```
#pragma HLS UNROLL
        back_substitute_step<RowsColsA, Type>(streamA[i], streamA[i+1], streamB[i-1], streamB[i], streamRo
    }

    back_substitute_last_step<RowsColsA, Type>(streamA[RowsColsA-1], streamB[RowsColsA-2], streamB[RowsCols

    transpose_B<RowsColsA, Type>(streamB[RowsColsA-1], out_stream, iter);
```



Stage 4: 10th Degree Polynomial Computation

- Generates 3x3 polynomial matrix B from matrix A*
- Expands the determinant of B to generate the coefficients of a 10th degree polynomial
- Apply the same methodologies as in Stage 3
    - Last operation is expressed with fully unrolled analytic formulas

```
#pragma HLS UNROLL
        back_substitute_step<RowsColsA, Type>(streamA[i], streamA[i+1]
```

- Sends coefficients to Stage 5
- Enqueues matrix B into the FIFO linked to Stage 6

B	x	y	1
$\langle k \rangle$	[3]	[3]	[4]
$\langle l \rangle$	[3]	[3]	[4]
$\langle m \rangle$	[3]	[3]	[4]

3x3 matrix **B** containing polynomial expressions in z

From det(B) = 0

$z1, z2, \ldots, z10$

Stage 5: Roots Extraction

- Solve the 10th degree polynomial and find up to 10 roots
- Optimised Sturm sequence approach
  - A compact array of 22 fp single precision coefficients
- Steps
  - Generate Sturm sequence
  - Generate search interval for each of the 10 potential roots
  - Keep track of sign changes and the position of the root
  - Refine roots interval with a pipeline of K(32) intervals bisector
- Buildsturm
  - Split into 2 stages in the pipeline to reduce dataflow interval
  - Most resource-consuming operations

B	x	y	1
$\langle k \rangle$	[3]	[3]	[4]
$\langle l \rangle$	[3]	[3]	[4]
$\langle m \rangle$	[3]	[3]	[4]

3x3 matrix **B** containing polynomial expressions in $\mathbf{z}$

From det(B) = 0

$\Downarrow$

$\mathbf{z1, z2, \ldots, z10}$

Stage 6: Essential Matrix Generation

- Recovers the essential matrix from the roots in Stage 5
- Performs the actual back-substitution
- Compute the inverse of the system matrix by leveraging 3x3 custom QR factorisation
- Recover essential matrices from the solutions for **x,y,z**

B	x	y	1
$\langle k \rangle$	[3]	[3]	[4]
$\langle l \rangle$	[3]	[3]	[4]
$\langle m \rangle$	[3]	[3]	[4]

3x3 matrix **B** containing polynomial expressions in **z**

Recover from

z1, z2, …, z10

x, y, z

Stage 7: Forward Error Estimation

- Compute the forward error of solutions according to a threshold
- Evaluate the ‘goodness’ of the solution found

aws marketplace

Sign in or Create a new account

Categories | Delivery Methods | Solutions | AWS IQ | Resources | Your Saved List | Partners | Sell in AWS Marketplace | Amazon Web Services Home | Help

FPGA Developer AMI

By: Amazon Web Services   Latest Version: 1.10.0

The FPGA (field programmable gate array) AMI is a supported and maintained CentOS Linux image provided by Amazon Web Services. The AMI is pre-built with FPGA development tools

Show more

Linux/Unix   6 AWS reviews

Free Tier

Continue to Subscribe

Save to List

Typical Total Price

$0.744/hr

Total pricing per instance for services hosted on z1d.2xlarge in US East (N. Virginia). View Details

Overview | Pricing | Usage | Support | Reviews

Product Overview

The FPGA (field programmable gate array) AMI is a supported and maintained CentOS Linux image provided by Amazon Web Services. The AMI is pre-built with FPGA development tools and run time tools required to develop and use custom FPGAs for hardware acceleration. The FPGA Developer AMI along with the FPGA Developer Kit(https://github.com/aws/aws-fpga) constitutes a development environment which includes scripts and tools for simulating your FPGA design, compiling code, building and registering your AFI (Amazon FPGA Image). Developers can deploy the FPGA developer AMI on an Amazon EC2 instance and quickly provision the resources they need to write and debug FPGA designs in

Highlights

- Xilinx Vitis 2020.2(v1.10.x), Xilinx Vitis 2020.1(v1.9.x), 2019.2(v1.8.x), SDx 2019.1(v1.7.x), 2018.3(v1.6.x), 2018.2(v1.5.x) or 2017.4 (v1.4.X) and Free license for F1 FPGA development
- AWS Integration - includes packages and configurations that provide tight integration with Amazon Web

Hardware Emulation

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.88 ns	5.425 ns	0.74 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
?	?	?	?	?	?	dataflow

Detail

- Instance
- Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	51	-
FIFO	1027	-	14245	39862	-
Instance	800	1916	464800	439868	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	54	-
Register	-	-	9	-	-
Total	1827	1916	479054	479835	0
Available	2688	5952	1743360	871680	640
Available SLR	1344	2976	871680	435840	320
Utilization (%)	67	32	27	55	0
Utilization SLR (%)	135	64	54	110	0

Discussion

- Most step latencies are hidden in pipeline
- Deep pipelining makes it difficult to isolate the effects of pipeline pragmas
- Utilisation exceeds SLR

Questions

- What does pragma HLS loop_merge do?
- If Vivado HLS already provides a library that implements our desired operators, under what conditions can we opt for a customised module?

References

- 5 Points to Rule Them All
- D. Nistér, “An efficient solution to the five-point relative pose problem,” IEEE transactions on pattern analysis and machine intelligence, vol. 26, no. 6, pp. 756–770, 2004.
- CS231A Course Notes 3: Epipolar Geometry

https://github.com/mouvemance/HLSLabB_point5