Computer Vision for Automotive/ADAS Market: Challenges and Embedded Vision Solutions

By
Texas Instruments Ltd.

CVPR, Las Vegas (http://cvpr2016.thecvf.com/)
26 June, 2016

For more details see [link]
Agenda

1. **Automotive/ADAS Market and Challenges**
   – Mihir Mody

2. **Embedded Hardware and SoC**
   – Pramod Swami

---

**Break (15 Min)**

3. **Embedded Software Architecture and Framework**
   – Kedar Chitnis

4. **Embedded Software Optimizations and Tuning**
   – Prashanth Viswanath

**Note**: Each Part is around 50 minutes with 5 minutes Q & A
Part 1: Automotive/ADAS Market and Challenges

Mihir Mody
Sr. Principal Architect – Vision and Imaging
Senior Member Technical Staff (SMTS),
Texas Instruments India Ltd.
Agenda

• Automotive Market
  • Advanced Driver Assistance System (ADAS)
    – Passive and Active System
    – Multiple Applications & Sensors
    – Software Stacks
    – Market Trend
  • ADAS Applications in Detail
    – Front Camera System
    – Surround View and Park Assistance System
    – Camera Mirror Replacements
    – ...
• Challenges and Next Step
• Summary
Increasing demands for the auto industry

**GREENER**

**SAFER** AND **SMARTER**

Long term market opportunities

- Opportunity to advance the car of the future
- $25 billion TAM in 2014
- Average shipping life of 5-10 years

1970 to 2016

- Miles Driven: 1 Trillion in 1970, 3 Trillion in 2013
- Driving Deaths: 52.6 K in 1970, 32 K in 2013

*Source: wikipedia.org, energy.gov*
Irrefutable truths of the auto industry

- **Slow to Market**
  - Emphasis of *caution* over *speed*

- **Safety**, liability and responsibility
  - Mistakes in automotive are costly to the consumer
Automotive Market Trends

- Safer – predict and prevent accidents
- Greener – more efficient, lighter, aerodynamic
- Smarter – use explosion of data to protect, save time and entertain
- Cheaper - democratization of technology

Car of the Future Video
Worldwide Road Traffic Fatalities

Total 2004

<table>
<thead>
<tr>
<th>Leading Cause</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Ischemic heart disease</td>
<td>12.2</td>
</tr>
<tr>
<td>2. Cerebrovascular disease</td>
<td>9.7</td>
</tr>
<tr>
<td>3. Lower respiratory infections</td>
<td>7.0</td>
</tr>
<tr>
<td>4. Chronic obstructive pulmonary disease</td>
<td>5.1</td>
</tr>
<tr>
<td>5. Diarrheal diseases</td>
<td>3.6</td>
</tr>
<tr>
<td>6. HIV</td>
<td>3.5</td>
</tr>
<tr>
<td>7. Tuberculosis</td>
<td>2.5</td>
</tr>
<tr>
<td>8. Trachea, bronchus, lung cancers</td>
<td>2.3</td>
</tr>
<tr>
<td>9. Road traffic injuries</td>
<td><strong>2.2</strong></td>
</tr>
<tr>
<td>10. Prematurity and low birth weight</td>
<td>2.0</td>
</tr>
</tbody>
</table>

~1.3 million

Total 2030

<table>
<thead>
<tr>
<th>Leading Cause</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Ischemic heart disease</td>
<td>14.2</td>
</tr>
<tr>
<td>2. Cerebrovascular disease</td>
<td>12.1</td>
</tr>
<tr>
<td>3. Chronic obstructive pulmonary disease</td>
<td>8.0</td>
</tr>
<tr>
<td>4. Lower respiratory infections</td>
<td>3.8</td>
</tr>
<tr>
<td>5. Road traffic injuries</td>
<td><strong>3.6</strong></td>
</tr>
<tr>
<td>6. Trachea, bronchus, lung cancers</td>
<td>3.4</td>
</tr>
<tr>
<td>7. Diabetes mellitus</td>
<td>3.3</td>
</tr>
<tr>
<td>8. Hypertensive heart disease</td>
<td>2.1</td>
</tr>
<tr>
<td>9. Stomach cancer</td>
<td>1.9</td>
</tr>
<tr>
<td>10. HIV</td>
<td>1.8</td>
</tr>
</tbody>
</table>

~2 million

Source World Health Organization
Eliminating Human Error Can Save Lives

• According to Tri-Level Study of the Causes of Traffic Accidents published by NHTSA in 1979, "human errors and deficiencies" are a definite or probable cause in 90-93% of the incidents examined.

• By eliminating human errors that cause traffic accidents ADAS can save lives, reduce severity of injuries and reduce property damage.

Source: NHTSA
Agenda

• Automotive Market

• Advanced Driver Assistance System (ADAS)
  – Passive and Active System
  – Multiple Applications & Sensors
  – Software Stacks
  – Market Trend

• ADAS Applications in Detail
  – Front Camera System
  – Surround View and Park Assistance System
  – Camera Mirror Replacements
  – ...

• Challenges and Next Step

• Summary
ADAS - Providing Integrated Safety

**Passive Safety**
- Systems in place to minimize impact of an accident
- Pre-Crash: Active Belts, Active Structures, Anti-whiplash
- In-Crash: Adaptive Belts, Adaptive Bags, Pedestrian Protection
- Post-Crash: Crash Notification, Drive Recorder, Emergency Call

**Active Safety**
- Systems in place to avoid accidents
- Assistance: Night Vision (NV), High Beam Assist (HBA), Adaptive Cruise Control (ACC), Pedestrian Detection (PD), Lane Departure Warning (LDW), Blind Spot Detection, Park Assist
- Warning: Forward Collision Warning (FCW)
- Emergency: Stability Control, Emergency Braking, Crash Avoidance

**ADAS**
- Passive
- Active ADAS
- Highly Automated
- Fully Autonomous

**Texas Instruments**
ADAS Applications: Building toward Autonomy

Core Applications

- Rear Camera
  - Low Power
  - Small Footprint
  - Scalable Analytics
- Surround View
  - Park Assist
  - Integrate 3D Graphics
  - Scalable Analytics
  - Security
- Front Camera
  - Scalable Performance
  - Low Power
  - Safety
- Radar
  - Scalable performance
  - MCU Integration
  - Safety

Emerging Applications

- Mirror Replacement
  - Performance
  - ISP Integration
  - Analytics
- Driver Monitoring
  - Small Footprint
  - Scalable Analytics
- Sensor Fusion
  - Performance
  - Safety
ADAS Increasing Number of Sensors

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>Vision</th>
<th>Infrared</th>
<th>Long Range Radar 76..81GHz</th>
<th>Short / Mid Range Radar 24..26 / 76..81 GHz</th>
<th>Lidar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive Front Lighting (AFL), Traffic Sign Recognition (TSR)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Night vision (NV)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adaptive Cruise Control (ACC)</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Lane Departure Warning (LDW)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low-Speed ACC, Emergency Brake Assist (EBA), Lane Keep Support (LKS)</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Pedestrian detection</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Blind Spot Detection (BSD), Rear Collision Warning (RCW), Lane Change Assist (LCA)</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Park Assist (PA)</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Camera monitor systems (CMS)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **LRR**: 1 to 280m
- **Infrared**: 0.2 to 120m
- **Video**: 0 to 80m
- **SRR/MRR**: 0.2 to 160m
- **Lidar**: 0.2 to 90m
- **Video**: 0 to 80m
Overview of ADAS Software Stack

ADAS Applications
- Lane Keep Assist
- Auto Emergency Brake
- Forward Collision Warning
- Speed Limit Warning

Application Framework / Middleware / Services

Operating System
- Sensor Drivers
- Communication Protocols
- Algorithms
- Diagnostics
- Power Management

Hardware Abstraction Layer

Hardware Components:
- CPU
- DSP
- HWA
- Camera/Radar IF
- Flash
- CAN
ADAS Vision Systems – Trends

**TREND 1 – Higher Accuracy & Robustness**
- Multi-mono / Stereo / Radar / Lidar / Sensor Fusion

**TREND 2 – Add Front Peripheral Vision**
- Side cameras & Radar with blind spot detect
- Wide Angle Peripheral Vision Camera / Radar

**TREND 3 – Add Vision All Around car**
- Rear camera with backover prevent

**TREND 4 – More accurate classification**
- Deep Learning methods, CNN
ADAS Market Trends

Performance and Growth
- Rapid Expansion of applications
- Rapid Expansion of platforms
- Widespread expansion over all regions

Scalability
- Moving from Luxury to Entry-line cars

Cost Pressure, Integration
- Legislation driving widespread projects:
  - US NHTSA Rear Camera
  - Euro NCAP

System Miniaturization
- Integration, smaller packages, low power

Safety
- ISO26262, ASIL-x support
Agenda

• Automotive Market
• Advanced Driver Assistance System (ADAS)
  – Passive and Active System
  – Multiple Applications & Sensors
  – Software Stack
  – Market Trend

• ADAS Applications in Detail
  – Front Camera System
  – Surround View and Park Assistance System
  – Camera Mirror Replacements
  – ...

• Challenges and Next Step
• Summary
Basic Rear View Camera

- Mounted on back of car (Above of license plate)
- Challenges
  - Lowest form factor (Aesthetics & Style drives to size of coin i.e. 1 inch space)
  - Lowest Cost (Mass volume)
  - Lower Power (form factor)
- First application and mass deployed
- Trend is Smart rear view with Computer Vision and Analytics

Source: Nissan
Source: Ford owner
Surround View & Parking Assistance

2D Surround View

3D Surround View

Challenges

High Performance
• Multiple Camera (4 to 6) processing performance
• Warping and Advance 2D & 3D stitching algorithm
• Analytics for automated parking system
• Display processing

Integrated solution with lower BOM for mass deployment
Surround View (Details)

Multi-camera Calibration

Lens Distortion correction

3D Surround View

2D Surround View on DSP

SGX Rendering

3D Mesh Table

Perspective Transform
Front Camera: EU-NCAP Requirements

### Challenges

<table>
<thead>
<tr>
<th>Ultra High compute performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Ever increasing Camera resolution to address higher distance and velocity of distance object</td>
</tr>
<tr>
<td>- Ever increasing set of accuracy requirements (including usage of Deep leaning /CNN)</td>
</tr>
<tr>
<td>- Multiple set of algorithm for end systems (around 5 or more algorithms)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strict Power /Thermal Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Mounted back of back of Rear view Mirror (exposed to sunlight) and small form factor limits maximum power consumption to &lt; 3-4 Watts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Robustness</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Working in all conditions (rain, snow, night,…) for all kind of condition on road scenarios (Debris,..)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Low Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Cost to ensure mass deployment</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Autonomous Braking for Cars &amp; VRU (domain II)</th>
<th>Lateral Assist Systems (domain III)</th>
<th>Speed &amp; Impaired Driving (domain IV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Pedestrian detection</td>
<td>✓ Lane Departure Warning</td>
<td>x Driver Monitoring (Interior)</td>
</tr>
<tr>
<td>✓ Cyclist/PTW detection</td>
<td>✓ Road edge, barrier detection</td>
<td>✓ Speed Limit Sign Recognition</td>
</tr>
<tr>
<td>✓ Vehicle Detection</td>
<td>✓ Adaptive headlights</td>
<td>✓ Traffic Light Recognition</td>
</tr>
<tr>
<td>x Adaptive headlights</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Ultra High compute performance

- Ever increasing Camera resolution to address higher distance and velocity of distance object
- Ever increasing set of accuracy requirements (including usage of Deep leaning /CNN)
- Multiple set of algorithm for end systems (around 5 or more algorithms)

Strict Power /Thermal Constraints

- Mounted back of back of Rear view Mirror (exposed to sunlight) and small form factor limits maximum power consumption to < 3-4 Watts

Robustness

- Working in all conditions (rain, snow, night,…) for all kind of condition on road scenarios (Debris,..)

Low Cost

- Cost to ensure mass deployment
Front Camera Systems

Mono Camera
- Single Lens
- Lower Cost
- Lower Power
- Difficult to attain high reliability
  - Need multiple cameras or mono + Radar for high reliability

Stereo Camera
- Dual Lens
- Higher Cost
- Higher Power
- High Computation
- Easy to attain High Reliability

Source: continental-automotive

Source: Bosch-mobility

Source: Bosch-mobility
Mono Front Camera High Level Block Diagram

- Battery
- DC/DC
- PMIC
- DDR
- SPI
- VIN
- /CSI
- i2c
- video
- Enet PHY
- Debug Only
- EMIF
- QSPI
- FLASH
- SoC (TDA)
- VIN /CSI
- SPI
- Timer
- RGMII
- i2c
- MCU
- CAN
- CAN Phy
- Vehicle CAN bus
Stereo Front Camera High Level Block Diagram
Typical Front Camera Signal Chain

- **Image Sensor**
- **Imaging Subsystem**
  - **Multiresolution Representation**
  - **Feature Extraction**
  - **Depth / Motion Estimation**
  - **Low Level Vision Processing**
    - **Pedestrian Detection**
    - **Cyclist Detection**
    - **Vehicle Detection**
    - **Lane Detection**
    - **Barrier Detection**
    - **Traffic Sign Recognition**
    - **Other Detection**
    - **Object Tracking Etc.**
    - **Adas Detectors**
      - Auto. Emergency Braking
      - Lane Keeping Assist
      - High-Beam Assist
      - Emergency Steering Assist
      - Adaptive Cruise Control
      - **Adas Applications**
Camera Monitoring System (CMS)

Challenges

- Ultra Low latency (Capture to Display)
- Sophisticated Image Processing including Wide Dynamic Range processing (WDR) (> 100dB)
- Eliminating LED Flicker captured by Camera
- Form Factor
- Analytics (Blind spot detection)
Agenda

• Automotive Market

• Advanced Driver Assistance System (ADAS)
  – Passive and Active System
  – Multiple Applications & Sensors
  – Software Stacks
  – Market Trend

• ADAS Applications in Detail
  – Front Camera System
  – Surround View and Park Assistance System
  – Camera Mirror Replacements
  – …

• Challenges and Next Step

• Summary
<table>
<thead>
<tr>
<th>Rear Camera</th>
<th>Surround View</th>
<th>Front Camera</th>
<th>Mirror Replacement</th>
<th>Driver Monitoring</th>
<th>Radar</th>
</tr>
</thead>
</table>

- **Lowest form factor** (Aesthetics & Style drives to size of coin i.e. 1 inch space)
- **Lowest Cost** (Mass volume)
- **Lower Power** (form factor)
- **Analytics** for smart

- **Multiple Cameras (4 to 6)**
- **Warping and Advance 2D & 3D stitching algorithm**
- **Analytics**
- **Integrated solution with lower BOM for mass deployment**

- **Ultra High compute performance due to multiple algorithms, high accuracy, higher resolution and fps camera**
- **Strict Power**
- **/Thermal Constraints**
- **Robustness**

- **Ultra Low latency**
- **Image processing (with WDR)**
- **LED Flicker**
- **Form Factor**
- **Analytics (Blind spot detection)**

- **IR Sensor Analytics:**
  - **Advanced Face/Eye/Gaze detection**
  - **Biometrics**
  - **Form Factor**

- **Performance / Form Factor Varies Across Applications**
- **Scalability Enables Broader/Cheaper Deployment**

**High precision radar computations in a small form factor**

- **Scalability short/med/long-range**
- **Small form factor**
- **Reduce/Remove DDR with large RAM integration**
Two Key Challenges: Safety, Security

Safety, Robustness by Design

- Video Input Subsystem
- Display subsystem
- Multi-Camera

SafeTI™

Security Enhancements

- Numerous HW & SW Security features provide Secure Boot & Trusted Execution Environment
- Firewalls
- eFuses
- Secure Clock
- Crypto

- Quality/Robustness Requirements have historically been more demanding in Automotive (eg Q100)
- ADAS trend toward active vehicle control and Autonomous driving raises the stakes (ISO26262/ASIL)
- Digital Cockpit integration also drives isolation requirements for robustness.

- Security in Infotainment was historically for content protection (DRM) primarily
- Stakes got raised with recent Jeep hack
- Vulnerability (increase in attack surfaces) will only increase as vehicle connectivity expands
- Standards Emerging (EVITA, SHE)
Next Step: Connectivity (V2V, V2I)

Intelligent Transport System (V2x)

• Advanced connectivity (vehicle-to-vehicle, vehicle-to-infrastructure) is the next major step
  • Safer: Extension of sensing beyond the vehicle itself, one more step towards autonomy
  • Greener: Traffic control/re-routing
  • Smarter: Augmented Reality
• Major Hurdles:
  • Safety and Security
  • QoS
  • Cost & Deployment rate

V2V standards and communication stacks (Source: Jiang, D. and Delgrossi, L.)
Agenda

• Automotive Market
• Advanced Driver Assistance System (ADAS)
  – Passive and Active System
  – Multiple Applications & Sensors
  – Software Stacks
  – Market Trend
• ADAS Applications in Detail
  – Front Camera System
  – Surround View and Park Assistance System
  – Camera Mirror Replacements
  – ...
• Challenges and Next Step
  • Summary
Summary

• Cars trending toward safer, smarter, greener at a lower cost point

• Significant advancements in ADAS on the path to Autonomy with ultimate goal of fully autonomous driving in future.

• Multiple ADAS Applications are now becoming more and more mainstream and offered by multiple OEM.

• Multiple modalities (Camera, Radar and Lidar) as well as multiple sensors of each modalities are getting deployed for better robustness and autonomous operations

• The challenges on Thermal /Power (due to form factor), cost (mass deployment) and high performance (due to better algorithmic accuracy needs, Deep learning), Safety (impacts life), Security (avoid hacker to take control of car) poses challenge for embedded solution.

• The car will continue to be an innovation hotbed for Semiconductor/ embedded processing to addresses above challenges to offer optimal solution.
Agenda

1. Automotive/ADAS Market and Challenges
   – Mihir Mody

2. Embedded Hardware and SoC
   – Pramod Swami

   Break (15 Min)

3. Embedded Software Architecture and Framework
   – Kedar Chitnis

4. Embedded Software Optimizations and Tuning:
   – Prashanth Viswanath

Note: Each Part is around 50 minutes with 5 minutes Q & A
Part 2: Embedded Hardware and SoC (System on Chip)

Pramod Swami
Principal Engineer, Vision and Imaging
Member Technical Staff (MGTS)
Texas Instruments India Ltd.
Agenda

- ADAS Applications and Key Requirements
- Understanding of Compute HW Options
  - Flexibility Vs Efficiency
- Computing elements (besides CPU & GPU)
  - Signal Processor (DSP)
  - Vision Processor (EVE)
  - Image Processor (ISP)
- Building Solution for ADAS
  - Surround View and Park Assist system
  - Front Camera ADAS system
- Scalable ADAS Platform (TDAx)
- What's in future
- Summary
ADAS Applications: Building toward Autonomy

**Core Applications**

- **Rear Camera**
  - Low Power
  - Small Footprint
  - Scalable Analytics

- **Surround View**
  - Park Assist
  - Integrate 3D Graphics
  - Scalable Analytics
  - Security

- **Front Camera**
  - Scalable Performance
  - Low Power
  - Safety

- **Radar**
  - Scalable performance
  - MCU Integration
  - Safety

**Emerging Applications**

- **Mirror Replacement**
  - Performance
  - ISP Integration
  - Analytics

- **Driver Monitoring**
  - Small Footprint
  - Scalable Analytics

- **Sensor Fusion**
  - Performance
  - Safety
Various Stages of ADAS Vision Applications

Mid level classifications and object detections

Best served with DSPs

Low level pixel repetitive operations

Best served dedicated Vision Processing engines

High-level processing responsible for decision making and tracking. Usually done in ARM processors
Being Pulled from all directions !!!

I THINK WE HAVE STRUCK THE RIGHT BALANCE

Power/Thermal

QUALITY SAFETY

COST

COMPUTE PERFORMANCE
Flexibility Vs Efficiency

- MPUs
- GPUs
- DSPs
- Vector Processor
- HWA
Agenda

• ADAS Applications and Key Requirements
• Understanding of Compute HW Options
  – Flexibility Vs Efficiency

• Computing elements (besides CPU & GPU)
  • Signal Processor (DSP)
  • Vision Processor (EVE)
  • Image Processor (ISP)

• Building Solution for ADAS
  – Surround View and Park Assist system
  – Front Camera ADAS system

• Scalable ADAS Platform (TDAx)
• What's in future
• Summary
Valuable Architecture Elements: VLIW

- VLIW – Very Long Instruction Word
- Pipelined parallel processing

Task: \( z_i = a_i x_i + b_i \) (\( i = 1, 2, \ldots, n \))

\[
\begin{align*}
M & \quad y_1 = a_1 x_1 \\
A & \quad z_1 = y_1 + b_1
\end{align*}
\]

\[
\begin{align*}
y_2 = a_2 x_2 \\
z_2 = y_2 + b_2
\end{align*}
\]

\[
\begin{align*}
y_3 = a_3 x_3 \\
z_3 = y_3 + b_3
\end{align*}
\]

\[
\begin{align*}
\vdots & \quad \vdots \\
y_n = a_n x_n & \quad z_n = y_n + b_n
\end{align*}
\]

Instructions (parallel processed)

Loop Prologue    Loop Main Body    Loop Epilogue
Valuable Architecture Elements: SIMD

- SIMD – Single Instruction Multiple Data
- Array processing

Task: \( y = x + 25 \) (\( x, y \) are arrays)
Digital Signal Processor (DSP)

- **C66x Processor**
  - VLIW (Very Large Instruction Word) architecture:
    - Two (almost independent) sides, A and B
    - 8 functional units: M, L, S, D
  - **Up to 8 instructions sustained dispatch rate**
    - Very extensive instruction set:
      - Fixed-point and floating-point instructions
      - More than 300 instructions
      - 8-/16-/32-bit/64-bit/128-bit data support
      - 32 MAC per cycle

- **Available internal Memories**
  - **L1D**: 32 KB (Configurable as SRAM/CACHE)
  - **L1P$**: 32 KB Program Cache
  - **L2**: 288 KB (Configurable as SRAM/CACHE)

- **EDMA**
  - 2 transfer controller
  - 32-64 DMA channels with separate 8 QDMA channels
  - 2D auto incremental transfer support
Vector Signal Processor (EVE)

• Processor
  - Very Powerful **16-way** vector processor with high speed memory interfaces (768 bits/cycles)
  - Additional RISC core to manage control and data transfer tasks
  - Advanced load and store instructions with data formatting, shift, saturate capabilities
  - Zero loop overhead mechanisms
  - Independent address generators
  - Very extensive instruction set

• Available internal Memories
  - **L1D**: 32 KB for ARP32
  - **L1P$**: 32 KB Program Cache
  - **96 KB** of internal memory for VCOP

• EDMA
  - 2 transfer controller
  - 32-64 DMA channels with separate 8 QDMA channels
  - 2D auto incremental transfer support
Image Signal Processor (ISP)

- Hardware accelerated image processing blocks
- Software Configurable algorithm parameters
- Software Configurable data flow
Heterogeneous Vision Processing Pipeline

ARM/DSP are needed for High-level vision stages of the algorithm.

**ARM Cortex Axx:**
- Scalable RISC
- Data Fusion
- Memory Coherency

**DSP:**
- VLIW SIMD+MIMD
- Data Fusion

**EVE Vector Coprocessor:**
- High Bandwidth
- Pixel Operations
- SIMD Parallelism
- Energy Efficiency

**Hardware Acceleration:**
- High Bandwidth
- Pixel Operations
- HW Acceleration
- Configurable
Agenda

• ADAS Applications and Key Requirements
• Understanding of Compute HW Options
  – Flexibility Vs Efficiency
• Computing elements (besides CPU & GPU)
  • Signal Processor (DSP)
  • Vision Processor (EVE)
  • Image Processor (ISP)
• Building Solution for ADAS
  – Surround View and Park Assist system
  – Front Camera ADAS system
• Scalable ADAS Platform (TDAx)
• What's in future
• Summary
Surround View Park Assist System

- Fish Eye Cameras
- Geometric Alignment Analysis
  - Geometric correction Parameters
    - Geometric LUT
  - Blending LUT
- Photometric Alignment Analysis
  - Photometric Statistics
  - Photometric correction Parameters
    - Photometric LUT
- Lens Distortion Correction
  - Photometric Correction
  - Blending/Stitching
- Different Perspective view (3D view) – GPU Rendering
- Bird’s Eye Surround view Output Frame
- Setup time
- Every Frame

Stitched top view
**SOC for surround view system**

- 4-6 Video camera input port
- Video decoder for Ethernet cameras
- Host processor (CPU) for high Level OS
- DSP for signal processing algorithms (LDC, image stitching)
- Internal/External memory and efficient data transfer capabilities
- Display subsystem
- GPU for different smooth 3D transitioning of view
- Interconnect for efficient data transfers b/w SOC components
- Other connectivity peripheral interfaces

Self Park assist systems would additionally need more compute (DSP) and decision making processors (micro controller).
Front camera ADAS systems

Mono / Stereo camera Input

<table>
<thead>
<tr>
<th>LOW LEVEL VISION PROCESSING</th>
<th>ADAS DETECTIONS</th>
<th>ADAS APPLICATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi Resolution Representation</td>
<td>Pedestrian Detection</td>
<td>Auto Emergency Braking</td>
</tr>
<tr>
<td>Feature Extraction</td>
<td>Cyclist Detection</td>
<td>Lane Keeping Assist</td>
</tr>
<tr>
<td>Depth/Motion Estimation</td>
<td>Vehicle Detection</td>
<td>High Beam Assist</td>
</tr>
<tr>
<td></td>
<td>Traffic Sign Recognition</td>
<td>Emergency Steering Assist</td>
</tr>
<tr>
<td></td>
<td>Lane Detection</td>
<td>Adaptive Cruise Control</td>
</tr>
<tr>
<td></td>
<td>Barrier Detection</td>
<td>......</td>
</tr>
<tr>
<td></td>
<td>Other Detection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Object Tracking</td>
<td></td>
</tr>
</tbody>
</table>

Mono/Stereo Camera Image reference
Key Take away
• Very **high compute** requirement
• **Varying** type of processing
  • Low level pixel processing
  • Mid level vision
  • High level decision making processing
• Different Approaches – **Programmability** is the key
SoC for Front Camera ADAS system

High Speed Interconnect
- CPU
- DSP
- On chip Memory
- DMA
- External Memory (DDR) interface

Specialized Vision Acceleration Engine
- Video Front End
  - 4-6 video camera
- Display Subsystem
- Video Codec Accelerator
- Graphics Engine

Connectivity and I/O
- Serial Connectivity
- Vehicle Connectivity
- Storage Connectivity
Agenda

• ADAS Applications and Key Requirements
• Understanding of Compute HW Options
  – Flexibility Vs Efficiency
• Computing elements (besides CPU & GPU)
  • Signal Processor (DSP)
  • Vision Processor (EVE)
  • Image Processor (ISP)
• Building Solution for ADAS
  – Surround View and Park Assist system
  – Front Camera ADAS system
  • Scalable ADAS Platform (TDAx)
• Summary
Generic and Scalable ADAS Platform

**PLATFORMING**

Focus on software reuse and TTM

**Scalable platforms**

Choose the right core for the right job

**Heterogeneous Architectures**

- MPUs
- GPUs
- DSPs
- AMPUs
- HWA

**Signal Processing**

- Imaging
- Vision
- Multimedia
- Audio

**Integration**

- Hardware Accelerators
  - PCIe
  - eAVB
  - USB2/3
  - MLB
  - CAN-FD
  - Power

- Vehicle Connectivity
  - Security
  - Safety MCU

**Generic and Scalable ADAS Platform**

- **Vehicle Connectivity**
- **Hardware Accelerators**
- **Signal Processing**
- **Integration**
TI’s TDA2x & TDA3x ADAS SoC Family

Enabling applications on the same architecture to facilitate:

- Reuse
- Reduced Integration risk
- Shorter time to market
- Reduced cost

High Performance at Lower Power

- Front Camera
- Surround View
- Sensor Fusion

TI’s TDA2x & TDA3x ADAS SoC Family
**TDA2x ADAS System On Chip**

### Device Integration
- Industry’s broadest range of IP blocks on one device: 2x A15, 4xM4, 2xC66xDSP, 4xEVE, 2xSGX544, IVA-HD
- Connectivity for FPD-Link, Gig Ethernet, PCIe & more

### Flexibility & Scalability
- Multiple algorithms running concurrently
- Supports multiple stages of an algorithm on different cores
- Vision SDK framework allows easy integration of multiple algorithms

### Performance
- Enables multiple ADAS applications to run simultaneously in real-time
TDA Family HW & SW Scalability

<table>
<thead>
<tr>
<th>Front</th>
<th>Surround</th>
<th>Fusion</th>
<th>Rear Camera</th>
<th>Radar</th>
<th>Driver Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>TDA2xA ADAS Processor</td>
<td>TDA2xV ADAS Processor</td>
<td>TDA2xV ADAS Processor</td>
<td>TDA2xF ADAS Processor</td>
<td>TDA2xV ADAS Processor</td>
</tr>
<tr>
<td>Mid</td>
<td>TDA3xA ADAS Processor</td>
<td>TDA2EcoV ADAS Processor</td>
<td>TDA2EcoV ADAS Processor</td>
<td>TDA3xF ADAS Processor</td>
<td>TDA3xR ADAS Processor</td>
</tr>
<tr>
<td>Entry</td>
<td>TDA3xV ADAS Processor</td>
<td>TDA3xV ADAS Processor</td>
<td>TDA3xV ADAS Processor</td>
<td>TDA3xV ADAS Processor</td>
<td>TDA3xV ADAS Processor</td>
</tr>
</tbody>
</table>

Common Software & Algorithm Investment
Binary Compatible SW Across Cores
Common SoC Architecture & Tools
Safety Isolation: Freedom from Interference

**FFI:** QM tasks should not *interfere* with ASIL tasks.

- **A8/A15**
  - ARM MMU
  - MMU for all memories

- **DSP**
  - MPU
  - Firewall: DDR and OCMC
  - MPU: L2 RAM

- **EVE**
  - EVE MMU

**Interconnect**

**M4**

---

**Simplified TDAx Block diagram**

- **DSP**
  - Firewall: DDR and OCMC
  - MPU: L2 RAM

- **M4**
  - Firewall: DDR and OCMC

- **HYBRID**
  - FFI

- **A8/A15**
  - MMU for all memories

- **EVE**
  - Firewall: DDR and OCMC

---

**Texas Instruments**
Security

• Remote hacking of Head-Unit of Chrysler Jeep Cherokee controlling speed, brake...  [YouTube Video](https://www.youtube.com/watch?v=MK0SrxBc1xs)

• Hack based on an unexpected security hole
  – Infotainment system had access to ADAS systems and it was not considered as an attack surface and left unsecure
  – Unnecessary and un-controlled access to MCU from Infotainment device

• Security Features in TDA2x
  – Boot authentication/encryption
  – JTAG lock
  – AES128 support
  – “SHE compliant”
  – Almost matches features of “FULL Evita HSM” except dedicated CPU for security
  – Support for additional keys, anti-rollback mechanism
Agenda

• ADAS Applications and Key Requirements
• Understanding of Compute HW Options
  – Flexibility Vs Efficiency
• Computing elements (besides CPU & GPU)
  • Signal Processor (DSP)
  • Vision Processor (EVE)
  • Image Processor (ISP)
• Building Solution for ADAS
  – Surround View and Park Assist system
  – Front Camera ADAS system
• Scalable ADAS Platform (TDAx)
  • What's in future
  • Summary
What’s Next: Deep Learning (CNN)

- Fixed / Handcrafted Features
  - Domain Expertize
  - Simple "Trained" Classifier
  - Machine Learning
- Trained Features
  - "Trained" Classifier
  - Machine Learning
- Specialized Processing element for CNN
- Cascaded layers
- Hierarchical representation
- Mathematical model
- Domain independent

Ultra High signal processing and efficient memory access
Summary

- Multiple adjacent ADAS applications area with good overlap
  - Front camera, surround view, rear view and emerging applications like Camera Mirror System and Driver Monitoring
- Needs heterogeneous architecture with good balance of programmable and specialized processor to meet power and performance envelop
  - Power and cost drives specialized processor like DSP, Vector Processor, ISP instead of GPU and CPU
- Lot of algorithmic innovations with different approaches, demands programmable MIPS for customer differentiation
- Platform approach is required to maximize software investment across adjacent application space
- TI’s TDAx family offers cost and power optimized solutions for multiple ADAS application, yet allowing common platform and software
- Future is moving towards autonomous driving – higher demands on SoCs
  - Hardware acceleration of matured technologies
  - Specialized programmable architecture for deep learning
  - Enhanced Security and safety feature support along with vehicle connectivity
Agenda

1. Automotive/ADAS Market and Challenges
   – Mihir Mody

2. Embedded Hardware and SoC
   – Pramod Swami

   Break (15 Min)

3. Embedded Software Architecture and Framework
   – Kedar Chitnis

4. Embedded Software API, Optimizations and Tuning:
   – Prashanth Viswanath

Note: Each Part is around 50 minutes with 5 minutes Q & A
Part 3: Embedded Software Architecture and Framework

Kedar Chitnis
Principal Software Architect – ADAS
Member Group Technical Staff (MGTS),
Texas Instruments India Ltd.
Agenda

• ADAS Data Flow Representation
• Front Camera Data flow
• SW Framework Definition
• Functional Safety considerations
• Comparison with Open compute frameworks
• Summary
Agenda

• ADAS Data Flow Representation
• Front Camera Data flow
• SW Framework Definition
• Functional Safety considerations
• Comparison with Open compute frameworks
• Summary
Software Representation of ADAS data flow

• Typically algorithm processing represented as a graph

• Ex, single algorithm within a compute core

• Ex, multiple algorithms across different compute core’s
Graph Execution Model

• Centralized
  – “master” core submits “work” to “worker” core
  – Advantages
    • “worker” core SW can be simpler
  – Disadvantages
    • More overheads at “master” core
    • More complex logic at “master” core

• Distributed
  – Peer to Peer communication between all core’s
  – Advantages
    • Lower overheads at each core
    • Identical SW logic at each core
  – Disadvantages
    • More complex logic at each core
Agenda

• ADAS Data Flow Representation

• Front Camera Data flow

• SW Framework Definition

• Functional Safety considerations

• Comparison with Open compute frameworks

• Summary
Example: Building data flow for Front Camera

Key Characteristics,
• Shows logical block diagram
• Not yet sure where to run different algorithms stages
• Assumes algorithm data flow within a block is well defined
Example: Building data flow for Front Camera

Key Characteristics,
- Shows first cut assignment of algorithms blocks to compute core’s
- Selection based on processing characteristics
- Inter processor communication (IPC) introduced when traversing across core’s
- Distributed graph execution model used
Example: Building data flow for Front Camera – Scenario 1

Key Characteristics,

• Assume platform is changed and new platform has single EVE but it has HWA for image pyramid
• Assume due to usage of HW additional pre-processing of data is required
• Need to shuffle core allocation of some algorithm blocks
Example: Building data flow for Front Camera – Scenario 2

Key Characteristics,
• Assume during development new requirement of front collision warning, object classification added
• Need to shuffle core allocation, change control flow
Agenda

- ADAS Data Flow Representation
- Front Camera Data flow
- SW Framework Definition
- Functional Safety considerations
- Comparison with Open compute frameworks
- Summary
Characteristics of Software Framework for ADAS

- Allows users to run multiple algorithms on different compute core’s like ARM, DSP, HWA, EVE, GPU
- Provides data-flow and control flow management across different compute core’s in the system
- Allows flexibility in changing system level parameters
  - Ex, how many and what type of compute core’s to use
  - Ex, how to distribute an algorithm across different compute core’s
- Provides a well defined, consistent API between different sub-systems
  - Allows clean separation of different sub-systems so as to allow changes in one sub-system to not affect other
  - Allows application to scale to multiple different applications using existing sub-systems
  - Enables easy integration and benchmarking of algorithms
- Follows Automotive SW development practices for functional safety (Ex, Misra-C, static allocation, predictable execution, run time diagnostics)
SW framework for ADAS applications

A "link" is the basic processing step in a video/vision data flow. A link consists of a OS thread coupled with a SW message box.

Links can run in parallel to each other. Processing is pipelined across "links".

The message box allows users as well as other links to talk to that link.

A link implements a specific interface which allows other links to directly exchange data buffers with each other. i.e intervention of the user not needed on a frame to frame basis.

Link API allows user to create, control and connect the links from a single "host" CPU. Internally Link API uses IPC to control the links on different processors.

A connection of links is called a chain. User creates a chain on a processor designated as "host" CPU.
# Unit of exchange – Buffer

## Buffer

Type =
- Video Frame OR
- Meta Data OR
- Compressed Stream

Payload
Channel ID
Timestamp

## Video Frame

- Buf Addr [n]
- Width
- Height
- Pitch [n]
- Format

## Meta Data

- Buf Addr [n]
- Buf Size [n]
- Filled Size [n]

## Compressed Stream

- Buf Addr
- Buf Size
- Filled Size
- Format
Buffer Exchange Between Links

Prev Link

1. NEW_DATA_CMD

Get Buffers()

2A

Output Buffer Queue

2B

Free Buffer Queue

Release Buffers()

Current Link

3. Input and output buffer available?

Free Buffer Queue

Output Buffer Queue

5A

Successful Completion

5B

Next Link

NEW_DATA_CMD

HWA Driver or Algorithm

4

Successful Completion

Exchanging buffers across processors

- Each link is implemented thinking that it is exchanging buffers with another link on same core.
  - This approach simplifies the implementation of the link since it need not worry about inter processor communication.
  - Also it allows the same link to be used to talk to links on same or different processors without any changes
- A pair of special links (IPC OUT, IPC IN) are used when buffers need to go across different processor’s
- Example, Image pyramid and Histogram of gradients running on same EVE
Exchanging buffers across processors

- Each link is implemented thinking that it is exchanging buffers with another link on same core.
  - This approach simplifies the implementation of the link since it need not worry about inter processor communication.
  - Also it allows the same link to be used to talk to links on same or different processors without any changes.
- A pair of special links (IPC OUT, IPC IN) are used when buffers need to go across different processor’s.
- Example, Image pyramid and Histogram of gradients, with Image Pyramid on HWA.
Connector Links – Flexible Data flow

**DUP Link**
- From HOG
- Pedestrian Detect
- Vehicle Detect
- Traffic Sign Recognition

One output to multiple input links

**Sync Link**
- Camera
- Front Collision Warning
- Object Visualization

Multiple “Synchronized” outputs to single input link

**Merge Link**
- Pedestrian Detect
- Vehicle Detect
- Traffic Sign Recognition
- MERGE

To Object Classify

Multiple outputs to single input link
Automated Code Generation

Capture
- Alg_FeaturePlaneComputation (EVE1)
- Alg_ObjectDetection (DSP1)
- Display

<table>
<thead>
<tr>
<th>Use-case</th>
<th>Nodes</th>
<th>Description LOC</th>
<th>Generated C code LOC</th>
<th>Original effort</th>
<th>With use-case translator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front Camera UC</td>
<td>79</td>
<td>37</td>
<td>2060</td>
<td>&gt; 10 days</td>
<td>&lt; 1 day</td>
</tr>
</tbody>
</table>
Front Camera Data Flow using “Links”

- Camera
- Image Pre-processing
- Image Pyramid
- Histogram of Gradients
- Vehicle Detection
- Pedestrian Detection
- Traffic Sign Recognition
- Optical Flow
- Structure From Motion
- Object Classification
- SYNC
- Front Collision warning
- Object Visualization
- Display
- DUP
- SYNC
- MERGE

IPC OUT/IN Links NOT shown for clarity
SW Framework within a compute core

Cache-based programming model

Data I/O

DMA-based programming model

Data I/O

Compute unit “sees” the entire image through cache.

Programming model is easier but inefficient in term of power and performance, because 2D access patterns generates frequent cache misses.

Compute core processes data in on-chip RAM.

DMA used to read/write small blocks of image in from/to on-chip.
Implementing the kernels

- Information provided by each kernel
  - Number and size of input, output, internal scratch buffers.
  - Control Parameter for the kernels
- Information calculated by the framework
  - Placement of output and scratch buffers for given input block size and control parameters
Stitching the kernels using “BAM”

- “BAM” – Block Acceleration Manager – is a SW framework which can stitch together discrete kernels to form a larger algorithm.
- Generic DMA Read and DMA Write nodes handle the DMA and abstract the kernel from DMA.
- “Ping-pong” block buffers in internal memory keep the DMA and compute core pipelined at a block level.
- Smart allocation within BAM can re-use “scratch” internal memory when not used by other kernel in the same BAM graph.
Agenda

• ADAS Data Flow Representation
• Front Camera Data flow
• SW Framework Definition

• **Functional Safety considerations**

• Comparison with Open compute frameworks
• Summary
Safety Considerations – Static Allocation

• All OS resources created statically

• All SW queues in system implemented using statically allocated fixed size array’s

• All Links created and waiting for messages at initialization time

• Memory for buffers carved out statically and passed as input during initialization

• IPC implemented using statically allocated shared memory regions

• Run-time diagnostics monitor usage of static resource allocation and system can be tuned to increase or decrease resources based on diagnostics
<table>
<thead>
<tr>
<th>Monitor</th>
<th>Implementation</th>
<th>Safety goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Bandwidth</td>
<td>HW counters with periodic polling</td>
<td>Detect abnormal DMAs, HWAs, and report violation</td>
</tr>
<tr>
<td>Application Statistics</td>
<td>Link level hooks to measure buffer statistics, latency at each link.</td>
<td>Detect pipeline stalls using dropped frames and identify slow links.</td>
</tr>
<tr>
<td>Task loads</td>
<td>OS APIs with time window on multiple CPUs to capture overall system load with task level granularity.</td>
<td>Detect anomalous task behavior of higher or lower CPU load than expected.</td>
</tr>
<tr>
<td>Low power time</td>
<td>Measure the time the CPU has been in low power state.</td>
<td>Detect if the CPU has been in low power for the time desired to meet power and thermal goals.</td>
</tr>
</tbody>
</table>
Agenda

• ADAS Data Flow Representation
• Front Camera Data flow
• SW Framework Definition
• Functional Safety considerations
• Comparison with Open compute frameworks
• Summary
Enabling Open Compute Frameworks

- OpenCV and OpenCL can be used to implement algorithm in a link
- OpenCL can be used to load and run OpenCL kernel C on compatible compute core’s
- OpenVX can be used to schedule graphs on compatible core’s
- All these mechanisms can co-exist and are tied together by “links” framework acting as the system level framework
Agenda

• ADAS Data Flow Representation
• Front Camera Data flow
• SW Framework Definition
• Functional Safety considerations
• Comparison with Open compute frameworks

• Summary
Summary

- SW Architecture for ADAS application needs to encompass system wide as well compute core specific aspects
- Graph based representation is well suited for computer vision SW data flows
- “Links” framework is introduced to handle processing of data across multiple heterogeneous compute core’s
- “BAM” framework is introduced to handle processing of data within a compute core.
- Parallelization of processing across compute core’s as within compute core’s is important to achieve power, performance targets
- Functional Safety should be considered upfront in SW architecture and system design
- SW Architecture should be able to work with existing Open compute frameworks
Agenda

1. Automotive/ADAS Market and Challenges
   – Mihir Mody

2. Embedded Hardware and SoC
   – Pramod Swami

   Break (15 Min)

3. Embedded Software Architecture and Framework
   – Kedar Chitnis

4. Embedded software: Optimizations and Tuning
   – Prashanth Viswanath

Note: Each Part is around 50 minutes with 5 minutes Q & A
Computer Vision For ADAS Market

Embedded Software: Optimizations and Tuning
Prashanth Viswanath
Algorithm Lead, ADAS
Agenda

- Introduction
- DSP Architecture – Introduction
- Kernel Level Optimization
- Loop Transformations
- Example Use case: Feature extraction using ORB
- Summary
What Does “Optimal” Mean?

◆ Every user will have a different definition of “optimal”:

“When my processing keeps up with my I/O (real-time) …”

“When my algo achieves theoretical minimum…”

“When I’ve worked on it for 2 weeks straight, it is FAST ENOUGH…”

“When my boss says GOOD ENOUGH…”

“After I have applied all known (by me) optimization techniques, I guess this is as good as it gets…”

What is implied by that last statement?
Know Your Goal and Your Limits…

\[ Y = \sum_{i = 1}^{\text{count}} \coeff_i \times x_i \]

for \((i = 0; i < \text{count}; i++)\){
    \(Y += \coeff[i] \times x[i];\) 
}

**Goals:**

- A typical goal of any system’s algorithm is to meet *real-time*
- You might also want to approach or achieve “*CPU Min*” in order to maximize #channels processed

**CPU Min (the “limit”):**

- The minimum # cycles the algorithm takes based on *architectural limits* (e.g. data size, #loads, math operations required)

**Real-time vs. CPU Min**

- Often, meeting real-time only requires setting a few compiler options (easy)
- However, achieving “CPU Min” often requires extensive knowledge of the architecture (harder, requires more time)
Optimization – Intro

◆ Optimization is:

Continuous process of refinement in which code being optimized executes faster and takes fewer cycles, until a specific objective is achieved (real-time execution).

◆ When is it “fast enough”? Depends on user’s definition.

◆ Compiler’s personality? Paranoid. Will ALWAYS make decisions to give you the RIGHT answer vs. the best optimization (unless told otherwise)

◆ Bottom Line:
  • Learn as many optimization techniques as possible – try them all (if necessary)
  • This is the GOAL of this chapter…

◆ Keep in mind: mileage may vary (highly system/arch dependent)
Agenda

• Introduction
• DSP Architecture – Introduction
  – SIMD
  – VLIW
  – Memory Hierarchy
• Kernel Level Optimization
• Loop Transformations
• Example Use case: Feature extraction using ORB
• Summary
SIMD

- Single Instruction Multiple Data
- Vector cores (EVE/GPU) are perfect SIMD machines
- Both DSP and ARM NEON contain packed instructions
- Byte operations exploit most of it
  - add(int), add2(short), add4(char)
Vectorization

for $i=1:n$

\[ C[i] = A[i] + B[i]; \]

for $i=1:n/4$

\[ \text{mem}_4(C[i]) = \text{add}_4(A[i],B[i]); \]

- Compiler does this automatically for simple cases.
- Not always straightforward
  - sometimes requires re-structuring or re-formulation
  - eg: Histogram(VLIB), FFT(IMGLIB)
## Pipelining

<table>
<thead>
<tr>
<th>CPU Type</th>
<th>Clock Cycles</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Pipelined</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>F₁ D₁ E₁</td>
<td>F₂ D₂ E₂</td>
<td>F₃ D₃ E₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipelined</td>
<td>F₁ D₁ E₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F₂ D₂ E₂</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F₂ D₂ E₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F₃ D₃ E₃</td>
</tr>
<tr>
<td></td>
<td>F₃ D₃ E₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Texas Instruments*
Instruction level Pipelining

Task: \( z_i = a_i x_i + b_i \) (\( i = 1, 2, \ldots, n \))

- Pipelining requires
  - same set of operations for a sequence of data
  - different operations so that different units can be used

- Most Signal Processing algorithms satisfy this criteria
VLIW vs SIMD hardware

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>SIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>S</td>
<td>M</td>
<td>D</td>
<td>VLIW</td>
</tr>
</tbody>
</table>

- VLIW is a simpler alternative to SIMD in terms of hardware
- But SIMD is not always better than VLIW!
## VLIW DSP

![Diagram of VLIW DSP with functional units](image)

### DSP Functional Units

<table>
<thead>
<tr>
<th></th>
<th>L</th>
<th>S</th>
<th>D</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>Integer Adder</td>
<td>Integer Adder</td>
<td>Integer Adder</td>
<td>Integer Multiplier</td>
</tr>
<tr>
<td></td>
<td>Logical</td>
<td>Logical</td>
<td>Load-Store</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Integer Comparison</td>
<td>Shifting</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bit Counting</td>
<td>Bit Manipulation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Branch/Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dual 16-bit Math</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Image credit: Texas Instruments*
Memory Hierarchy

- L1, L2, DDR - trade off between cost and speed
- Ping-pong buffering + DMA – typically 2X faster than cache
• **Hough Transform**
  - Output of Hough transform is a 2D space holding a histogram of each pair of rho, theta (typically rho range is 0 to 800 and theta range is 0 to 180. So it can be visualized as 800x180 2D buffer
  - Since it is a histogram, typically used approaches are **cache based** with below processing loop
    - For edges [1: N]
      - For theta [0:180]
        » Compute Rho and update histogram at rho, theta location
    - in this loop structure, the Hough space buffer is accessed in **random order** at any where in 800x180 space
  - Re-order the loop structure
    - For theta [0:180]
      - For edges [1: N]
        » Compute Rho and update histogram at rho, theta location
    - Here, we can keep all the edge positions stored in internal memory, have 2 rows (2x800) for Hough space buffer in internal memory
    - With this loop structure and memory arrangement, **CPU always access internal memory** and DMA is used to hide the transfer of data back to external memory (shown in below figure)
Agenda

• Introduction
• DSP Architecture – Introduction
• **Kernel Level Optimization**
  – Operation simplification
  – Reuse across iterations
  – Avoid control code
  – Fixed point design – IQmath
  – General programming guidelines
• Loop Transformations
• Example Use case: Feature extraction using ORB
• Summary
Polynomial Simplification

• Minimizing the number of operations
  – eg: \(ac + ad + bc + bd \Rightarrow (a + b)(c + d)\)

• Horner’s rule
  – \(d + c^2x + b^2x^4 + a^2x^6 \Rightarrow d + x^2(c + x^2(b + a^2x^2))\)
  – reduces the number of multiplications from \(O(n^2)\) to \(O(n)\)
  – better numerical stability

• Moving the operations to constants instead of variables
  – eg: \(\text{val} = \sqrt{\text{var}}; \text{val} > \text{const} \Rightarrow \text{var} > \text{const}^2\)

• Remove un-necessary intermediate computations
  – eg: \(a/\sqrt{x+y} + b/\sqrt{x+y} > 0 \Rightarrow a+b > 0\)
Calculations Reuse Across iterations

FOR J = 1: M
    FOR I = 1: N
        SUM[BLOCK[I,J]];
    
• Try to find overlapping computations in different loop iterations
• May require re-formulation
Control Code Optimization for DSP

• What is Control Code?
  – Code with little or no signal processing
  – Complex decision making (if-rich code)
  – Resists automated optimization by the compiler

• Generally fulfills these types of functions:
  – Maintenance of data-structures
    • Structs, pointers, linked lists…
  – Higher level decision making in the application
    • Allocation of resources
    • Scheduling decisions
    • High-level communication protocol implementation
What is a DSP good at?

• The DSP architecture favors:
  – Loop pipelining
    • Up to 8 instructions per cycle
  – Conditional statements
    • Predicated instructions, no branches required
  – Computation intensive code (SIMD)

• The memory architecture favors:
  – Linear read operations on multiple arrays
    • Cache misses are pipelined
    • Important when reading from “slow” memory (DDR)
If Statements

• Compiler will if-convert short if statements:

Original C code:

\[
\text{if (p) then } x = 5 \text{ else } x = 7
\]

Before if conversion:

\[
[p] \text{ branch thenlabel}
\]
\[
\text{ x = 7 goto postif}
\]
\[
\text{thenlabel: } x = 5
\]
\[
\text{postif:}
\]

After if conversion:

\[
[p] \ x = 5 \ || \ [!]p \ x = 7
\]
Look up tables

• Complex arithmetic functions can be replaced by a LUT to improve speed
• Multivariable functions require multidimensional arrays
  – eg: x/y, x^y
• Pre-compute everything which is possible
  
  Imgs_orien(Imgs_orien >= 160) = 9;
  
  ..................
  Imgs_orien(Imgs_orien >= 40) = 3;
  Imgs_orien(Imgs_orien >= 20) = 2;

Orientation[360]=[1,1,1,…2,2,2,…3,3,3…]
AOS vs SOA

**AOS:**
```c
struct
{
  int x;
  int y;
  char edge;
}POS[100];
```

**SOA:**
```c
struct
{
  int *x;
  int *y;
  char *edge;
}POS;
```

AOS → x1y1e1x2y2e2....
SOA → x1x2......y1y2.....e1e2....

- SIMD
- 32-bit Alignment
- Better memory management
General programming guidelines

• Align I/O arrays to a 32-bit address – aligned loads/stores faster on the DSP

• Use minimum required data-type – error analysis for a trade-off
  – eg: OpenCV/matlab uses float values for everything!

• Use specific constraints of the use case and remove all unnecessary code
  – eg: memset initialization
General programming guidelines

• Avoid C++ – code generation for DSP is poor
• Align I/O arrays to a 32-bit address – aligned loads/stores faster on the DSP
• Use minimum required data-type – error analysis for a trade-off
  – eg: OpenCV/matlab uses float values for everything!
• Use specific constraints of the use case and remove all unnecessary code
  – eg: memset initialization
Restrict Qualifiers

myfunc(type1 input[ ],
    type2 *output)
{
    for (...)
    {
        load from input
        compute
        store to output
    }
}

• DSP depends on overlapping loop iterations for good (software pipelining) performance.

• Loop iterations cannot be overlapped unless input and output are independent (do not reference the same memory locations).

• Most users write their loops so that loads and stores do not overlap.

• Compiler does not know this unless the compiler sees caller or user tells compiler.

• Use restrict qualifiers to tell compiler:

   myfunc(type1 input[restrict],
           type2 *restrict output)
Restrict Qualifiers

original loop

restrict qualified loop

execution time

iter i

load compute store

iter i

load compute store

load compute store

load compute store

load compute store

load compute store

load compute store

load compute store

Agenda

• Introduction
• DSP Architecture – Introduction
• Kernel Level Optimization
• **Loop Transformations**
  – Loop Merging
  – Loop Unrolling
  – Loop Fusion
  – Loop Fission
  – Loop Interchange
  – Loop restructuring to avoid control code
  – Logical versus Bitwise operators
  – Re-arranging Data
  – Combination of Loop techniques
• Example Use case: Feature extraction using ORB
• Summary
Loop Merging

\[
\begin{align*}
&\text{for } l=1:\text{row} \\
&\qquad \text{for } J=1:\text{col} \\
&\qquad \quad \{ \\
&\qquad \quad \quad \text{....} \\
&\qquad \quad \} \\
&\end{align*}
\]

\[
\begin{align*}
&\text{for } l=1:\text{row} \times \text{col} \\
&\qquad \{ \\
&\qquad \quad \text{....} \\
&\qquad \} \\
&\end{align*}
\]
Loop Unrolling

for l=1:row
    for J=1:col
        for r=1:block_width
            for c=1:block_height
                { .... }
        
    
end

- To collapse small inner loops
- Doubling pipelined elements
Loop Fusion

```c
for (i = 0; i < 100; i++)
a[i] = b[i]+c;
for (i = 0; i < 100; i++)
d[i] = a[i+1]+e;

for (i = 0; i < 100; i++)
{
    a[i] = b[i]+c;
    d[i] = a[i+1]+e;
}
```

- Lesser prolog/epilog overhead
- Can reuse/remove operations
- Becomes difficult when addressing patterns are very different
- Be careful with dependencies
Loop Fission

\[
\begin{align*}
\text{FOR } I &= 1: N \\
\quad & \quad \text{FOR } J = 1: M \\
\quad & \quad \quad A(I,J+1) = A(I,J) + C \\
\quad & \quad \quad B(I+1,J) = B(I,J) + D \\
\text{FOR } J &= 1: M \\
\quad & \quad \text{FOR } I = 1: N \\
\quad & \quad \quad A(I,J+1) = A(I,J) + C
\end{align*}
\]

- Register pressure
- Dependencies
Loop Interchange

\[
\begin{align*}
\text{for } i=1:m & \quad \text{for } j=1:n \\
& \quad A(i,j+1) = A(i,j) + B \\
\text{for } i=1:n & \quad \text{for } k=1:n \\
& \quad A(i,j) = A(i,j) + B(i,k) \times C(k,j)
\end{align*}
\]
Loop restructuring to avoid conditional code

\[
\begin{align*}
\text{for } i=1:m \\
\quad \text{for } j=1:n \\
\quad \quad \text{if } A(j)>0 \\
\quad \quad \quad B(j,i) &= B(j,i) + 1 \\
\text{end} \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
\text{for } j=1:n \\
\quad \text{if } A(j)>0 \\
\quad \quad \text{for } i=1:m \\
\quad \quad \quad B(j,i) &= B(j,i) + 1 \\
\text{end} \\
\text{end}
\end{align*}
\]
Logical vs bitwise operators

• For logical operators (a || b), where a and b are expressions, the expression “a” must be evaluated first and “b” will not be evaluated unless “a” is evaluated to false.

• Bitwise operators (a | b) can avoid the control flow (branches) that is required when using logical operators, and improve control loop efficiency.

• Changing from logical to bitwise operators can make some control loops pipeline.
Re-arranging Data

- Some algorithms operate only on interest points rather than all the pixels – Lucas Kanade, Harris Corner, Kalman Filter, SIFT, SURF
- It might be better to bring the data of interest together and then run a single loop over all the points
Agenda

• Introduction
• DSP Architecture – Introduction
• Kernel Level Optimization
• Loop Transformations
  • **Example Use case: Feature extraction using ORB**
    – Fast9 corner detection
    – Fast9 score computation
    – Non maximal suppression
• Summary
Example: Feature extraction

• Feature extraction is a primary step in most automotive applications such as Optical Flow, Structure from Motion etc.

• Motivation
  – We have two images. How do we determine the flow vectors?

* Image from Rick Szeliski, Svetlana Lazebnik, and Kristin Grauman slideset
Example continued

1) Detection: Identify the interest points

2) Description: Extract vector feature descriptor surrounding each interest point.

3) Matching: Determine correspondence between descriptors in two views

\[ x_1 = [x_1^{(1)}, \ldots, x_d^{(1)}] \]

\[ x_2 = [x_1^{(2)}, \ldots, x_d^{(2)}] \]

* Image from Rick Szeliski, Svetlana Lazebnik, and Kristin Grauman slideset
Characteristics of good features

• **Repeatable**
  – The same feature can be found in several images despite geometric and photometric transformations

• **Salient**
  – Each feature is distinctive

• **Compact and efficient**
  – Many fewer features than image pixels

• **Locality**
  – A feature occupies a relatively small area of the image; robust to clutter and occlusion
How to find ORB descriptor

1. Input image
2. Keypoint detection (FAST9)
3. Strength of Keypoint (FAST9 Score)
4. Non-Maximal Suppression
5. Select Best Features
6. Compute rBRIEF Descriptor
7. Select Best Features
8. Harris Score
Theoretical Analysis of Traditional Approach

<table>
<thead>
<tr>
<th>Task</th>
<th>Cycles</th>
<th>Compute MHz</th>
<th>I/O MHz</th>
<th>Final MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image re-size</td>
<td>94500</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Fast9 Corner detect + score + NMS</td>
<td>2194500</td>
<td>2.2</td>
<td>0.2</td>
<td>2.2</td>
</tr>
<tr>
<td>2D to list</td>
<td>64141</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Sorting with payload input (Id)</td>
<td>42064</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Id list, big list -&gt; short list</td>
<td>2307</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>N point gradient + harris score</td>
<td>81641</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Sorting with payload input (Id,level)</td>
<td>10516</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>(Id,level) Big list -&gt; short list</td>
<td>577</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>rBRIEF</td>
<td>1100000</td>
<td>1.1</td>
<td>0.4</td>
<td>1.1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>3590246</strong></td>
<td><strong>3.6</strong></td>
<td><strong>1.4</strong></td>
<td><strong>3.6</strong></td>
</tr>
</tbody>
</table>
FAST (Features from Accelerated Segmented Test) Key point detection

“N” contiguous pixels out of the 16 need to be either above or below $p$ by the value $\text{Thr}$, if the pixel needs to be detected as an interest point.

Input access pattern in a Vector Engine

Addr_in = I2 * VCOP_SIMD_WIDTH * ELEMSZ_IN + I1 * pitch * ELEMSZ_IN;

Vin0 = (vec1+3*pitch+3)[Addr_in];
Vin1 = (vec1+3)[Addr_in];
Vin2 = (vec1+4)[Addr_in];
Vin3 = (vec1+pitch+5)[Addr_in];
Vin4 = (vec1+2*pitch+6)[Addr_in];

Loop 1
Access elements from the top – right corner
Replicate the lower 8 bits indicating whether the 16 pixels on arc are lighter/darker than the central pixel in upper half of 24 bit variable $X_0$.

$X_1 = X_0 >> 1$
$X_2 = X_1 \& X_0$

$X_3 = X_2 >> 2$
$X_4 = X_3 \& X_2$

$X_5 = X_4 >> 4$
$X_6 = X_5 \& X_4$

$X_7 = X_6 >> 1$
$X_8 = X_7 \& X_6$

If $X_8 \neq 0$ mark pixel as corner, else mark it as not a corner.

Core compute performance for Fast9 keypoint detection: **5.3 cycles/pixel**
Fast9 Score computation

Score = 48

Iterative
Non deterministic
Not SIMD friendly
Fast9 Score computation: Proposed Approach

### Score Calculation Rules

<table>
<thead>
<tr>
<th>Condition</th>
<th>Formula</th>
<th>Score Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMAX &gt; p &amp; VMIN &gt; p</td>
<td>Score = ( \max(VMAX - p - 1, VMIN - p - 1) )</td>
<td>48</td>
</tr>
<tr>
<td>VMAX &lt; p &amp; VMIN &lt; p</td>
<td>Score = ( \min(p - VMAX - 1, p - VMIN - 1) )</td>
<td></td>
</tr>
</tbody>
</table>

### Table Example

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>21</td>
<td>33</td>
<td>32</td>
<td>33</td>
<td>95</td>
<td>92</td>
<td>97</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>33</td>
<td>32</td>
<td>33</td>
<td>95</td>
<td>92</td>
<td>97</td>
<td>20</td>
<td>13</td>
</tr>
<tr>
<td>33</td>
<td>32</td>
<td>33</td>
<td>95</td>
<td>92</td>
<td>97</td>
<td>20</td>
<td>13</td>
<td>20</td>
</tr>
</tbody>
</table>

**Non iterative**

**Deterministic**

**SIMD friendly**
Non-maximal Suppression: Traditional Approach

• FAST9 keypoints are clustered, but sparse in nature

• Traditional approach of non-max suppression
  – Apply a 3x3 window across the input and retain only the maxima in the window
  
  ![Image of 3x3 window example]

• Disadvantages of the above approach
  – FAST9 score has to be computed for every pixel
  – If FAST9 score is computed only for keypoints, it has to be mapped back into the 2D image structure
Sparse Non-maximal Suppression

Horizontal Non Maximal suppression

<table>
<thead>
<tr>
<th>XY List</th>
<th>Score List</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0303</td>
<td>45</td>
</tr>
<tr>
<td>0x0403</td>
<td>50</td>
</tr>
<tr>
<td>0x0504</td>
<td>47</td>
</tr>
<tr>
<td>0x0604</td>
<td>45</td>
</tr>
<tr>
<td>0x0804</td>
<td>40</td>
</tr>
</tbody>
</table>

Y is same
X = X - 1

Y is same
X = X + 1

Compare Score

Vertical Non Maximal suppression

<table>
<thead>
<tr>
<th>XY List</th>
<th>Score List</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0303</td>
<td>0</td>
</tr>
<tr>
<td>0x0403</td>
<td>50</td>
</tr>
<tr>
<td>0x0504</td>
<td>47</td>
</tr>
<tr>
<td>0x0604</td>
<td>40</td>
</tr>
<tr>
<td>0x0804</td>
<td>52</td>
</tr>
</tbody>
</table>

X is same
Y = Y - 1

X is same
Y = Y + 1

Compare Score

Sort on X

H-NMS: 1.3 cycles/keypoint
Sort: 4.8 cycles/keypoint
V-NMS: 2.2 cycles/keypoint
## Theoretical Analysis of Proposed Approach

### Flow 1 - Fast9 score+NMS done on entire image

<table>
<thead>
<tr>
<th>Task</th>
<th>Cycles</th>
<th>Compute MHz</th>
<th>I/O MHz</th>
<th>Final MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image re-size</td>
<td>94500</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Fast9 Corner detect + score + NMS</td>
<td>2194500</td>
<td>2.2</td>
<td>0.2</td>
<td>2.2</td>
</tr>
<tr>
<td>2D to list</td>
<td>64141</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Sorting with payload input (Id)</td>
<td>42064</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Id list, big list -&gt; short list</td>
<td>2307</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>N point gradient + harris score</td>
<td>81641</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Sorting with payload input (Id,level)</td>
<td>10516</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>(Id,level) Big list -&gt; short list</td>
<td>577</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>rBRIEF</td>
<td>1100000</td>
<td>1.1</td>
<td>0.4</td>
<td>1.1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>3590246</strong></td>
<td><strong>3.6</strong></td>
<td><strong>1.4</strong></td>
<td><strong>3.6</strong></td>
</tr>
</tbody>
</table>

### Flow 2 - Fast9 score and NMS done on key points only

<table>
<thead>
<tr>
<th>Task</th>
<th>Cycles</th>
<th>Compute MHz</th>
<th>I/O MHz</th>
<th>Final MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image re-size</td>
<td>94500</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Fast9 Corner detect</td>
<td>1092000</td>
<td>1.1</td>
<td>0.1</td>
<td>1.1</td>
</tr>
<tr>
<td>2D to list</td>
<td>64141</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>N point - Fast9 Score (SAD based)</td>
<td>115500</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>Linear NMS</td>
<td>68412</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Sorting with payload input (Id)</td>
<td>42064</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Id list, big list -&gt; short list</td>
<td>2307</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>N point gradient + harris score</td>
<td>81641</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Sorting with payload input (Id,level)</td>
<td>10516</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>(Id,level) Big list -&gt; short list</td>
<td>577</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>rBRIEF</td>
<td>1100000</td>
<td>1.1</td>
<td>0.4</td>
<td>1.1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2671658</strong></td>
<td><strong>2.7</strong></td>
<td><strong>1.4</strong></td>
<td><strong>2.7</strong></td>
</tr>
</tbody>
</table>
Summary

• Know your goal
  – Define the performance that needs to be achieved

• Understand the limit
  – Understand the theoretical limit based on the architecture of the core
  – It will help in closing the gap between the performance achieved and the limit

• Design and implement based on your goal
  – Understand the architecture and design keeping that in mind
  – Try various optimization techniques based on the goal and the time available

• Optimization is a continuous process and might need multiple iterations

• Use standard optimized packages from the vendor