Embedded System Designs for Compute-Intensive Applications

Chuanjun Zhang
Intel Science and Technology Center for Embedded Computing
Outline

- Introduction to the ISTC-EC
  - New applications in home, retail stores, and automotive
  - Compute-intensive applications
- Characterizations of compute-intensive applications
  - An example application - HMAX
- Implications on embedded system design
  - An example challenge – memory design
- EMERALD workloads
  - Eleven emerging applications and algorithms in low power devices
- Conclusion
Application-Inspired Research

Disruptive advances in algorithms and systems to transform connected embedded computing

Collaborative Perception

Real-time Knowledge Discovery

Robotics

Embedded Systems

Retail  Automotive  Home
Applications in Automotive

Driving in Rain  Standard Headlight  Smart Headlight

Applications in Automotive

People Intention Aware Automotive System

University of Pennsylvania
Applications at Home
Applications at Home

- Reliable (100% success rate) folding of towels
- Learning to tie knots from demonstrations [ICRA2010]

University of California, Berkeley
Applications in Retail Stores

- Restocking
- Misplaced items
- Satisfaction analysis
- Making suggestions

Penn State University and Carnegie Mellon University
Real Time Compute-Intensive Applications

- The emerging applications in automotive, home, and retail are compute-intensive and need real time processing
  - Real time processing
    - Safety critical: autonomous cars
    - User experience: retail stores, robots at home, facial expression and hand gesture controlled TV, etc.
  - Compute-intensive
    - Computer vision, image processing, and machine learning algorithms are used
Characterization of Compute-Intensive Applications: An Example Algorithm - HMAX

- HMAX is a computational model mimicking the visual cortex in mammal’s brain
  - The biological algorithm can provide robust and accurate results in a wide range of recognition tasks including object, face/facial expression and action recognitions

Many different implementations of the HMAX model

Our implementation: input image is 256×256, 11 image pyramids are generated using bi-cubic interpolation

The 11 image pyramids are 214×214, …, 44×44, and 38×38

Scale factor is $2^{0.25} = 1.19$

Memory capacity for the image pyramids is 215KB

S1 Layer of the HMAX

- Gabor filters detect edges and orientations
  - The kernel size is $11 \times 11$ with 12 orientations
  - Generate 12 scales image pyramids: $246 \times 246$, $204 \times 204$, ..., $34 \times 34$, $28 \times 28$
  - There are a total of 144 images

$$\text{Im} \otimes K = \sum_{i=-m}^{m} \sum_{j=-n}^{n} \text{Im}(x-i, y-j) \times K(i, j)$$

- Memory capacity for the generated images is 2.3MB

C1 Layer of the HMAX

- Finding the local invariance
  - It takes in the 144 image outputs from the S1 layer
  - finds the local maximum within $10 \times 10$ square across two neighboring layers
  - Subsampling by moving around the S1 pyramid in steps of 5
- Memory capacity for the output of the C1 stage is 7.1KB

The S2 stage consumes 80-95% of the execution time
- The S2 layer convolves images from the C1 layer with 5120 pre-trained patches with each has 12 orientations
- Memory capacity for the template prototype is ~20MB
- S2 output memory capacity is 25-100MB, depends on number of bytes used for the feature responses
- If combined with the C2 stage to find the local maxima, the memory for feature responses is trivial

C2 Layer of the HMAX

- The last layer takes the maximum response of a position across all orientations and all scales for each prototype’s responses from the S2 layer. After processing, it outputs a vector of features for later classification.
- Memory capacity: output a 5120 dimension vector.
Summary of the HMAX

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>C1</th>
<th>S2</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exe. Time %</td>
<td>0.02%</td>
<td>3.01%</td>
<td>.16%</td>
<td>96.61%</td>
<td>.2%</td>
</tr>
</tbody>
</table>

- On Atom machine: MPKI: 0.04, 652.8 seconds for a 256×256 image
- On Core-2 machine: MPKI: 0.00, 81.0 seconds for the 256×256 image
- Cannot meet real time processing on embedded and desktop systems
Two Accelerator Designs of the HMAX

- Maashri’s implementation
  - Less FPGA resource
  - No buffer for the prototypes
  - Frame per second: 1fps

- Park’s implementation
  - More FPGA resources
  - Buffer for prototypes
  - 107 times faster than Maashri’s

<table>
<thead>
<tr>
<th></th>
<th>Register slices</th>
<th>LUT Slices</th>
<th>BRAM</th>
<th>DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maashri’s HMAX</td>
<td>46944</td>
<td>42172</td>
<td>24(108KB)</td>
<td>259</td>
</tr>
<tr>
<td>Park’s HMAX</td>
<td>256672</td>
<td>197252</td>
<td>803(3.6MB)</td>
<td>1536</td>
</tr>
<tr>
<td>Ratio</td>
<td>5.5</td>
<td>4.7</td>
<td>33.5</td>
<td>5.9</td>
</tr>
</tbody>
</table>

Logic gates, on chip buffer, and off-chip bandwidth are critical to HMAX.


Motivations of Heterogeneous SoC

- Meet real time processing with ultra low energy
- Fixed and programmable accelerators

Many challenges exist
An Example of Challenges: Memory Hierarchy Design For Heterogeneous SoC

Buffer-capacity is non-trivial

<table>
<thead>
<tr>
<th>Buffer Type</th>
<th>Capacity</th>
<th>Storage Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SURF</td>
<td>872KB</td>
<td>Scratchpad, FIFO</td>
</tr>
<tr>
<td>HMAX</td>
<td>108KB- 3.6MB</td>
<td>FIFO, Scratchpad</td>
</tr>
<tr>
<td>NIC</td>
<td>1536KB</td>
<td>FIFO</td>
</tr>
<tr>
<td>MRF-TRW-S</td>
<td>544KB</td>
<td>FIFO</td>
</tr>
<tr>
<td>GIST</td>
<td>3645KB</td>
<td>Scratchpad, FIFO</td>
</tr>
<tr>
<td>Saliency</td>
<td>1678.5KB</td>
<td>FIFO</td>
</tr>
</tbody>
</table>

- **Buffer-Integrated-Caches** (Zhen Fang, et al. Cost-Effectively Offering Private Buffers for SoCs and CMPs. ICS, 2011.)
  - Several private SRAM blocks being instantiated (cache to core, or scratchpads for IPs)
  - Opportunity for centralized shared SRAM that exposes both buffers and caches

  - Several Accelerators share the same store
Motivations of Building Workloads

Compute-Intensive Applications

- Workloads/Benchmarks
  - Evaluation of processor designs
    - Predictions of power and performance at varied processor, accelerator architecture and parameters configurations of a workload or multiple workloads
    - Computation or memory intensive
    - Instruction mixes, branch prediction, bus bandwidth, cache characteristics memory footprint
  - Compute intensive applications on low power devices
    - Domain or application specific accelerator designs

- Examples of workload suite
  - SPEC 2006/2000: general purpose processors
  - Mibench: automotive and industrial control, consumer devices, office automation, networking, security, and telecommunications
Eleven Workloads in the EMERALD

- Audio denoise and separation (Univ. of Illinois)
- Markov Random Field inference algorithms (Univ. of Illinois)
  - MRF-TRW-S, MRF-Dual decomposition, and MRF graph cut
- Physics simulation systems (Univ. of California, Berkeley)
  - Bullet physical engine and Blender 3D game engine
- Biological inspired algorithms (Penn State and Univ. of Southern California)
  - HMAX, GIST
- Smart headlight (Carnegie Mellon Univ.)
- K-Nearest Neighbor (Univ. of British Columbia and Carnegie Mellon Univ.)
Experiments Methodology

- Fulltime execution on real computing platform
  - Intel ATOM™ and Core™ 2
  - Simulation based method can only executed limited time
- ATOM based Medfield processor is used for Intel Smartphone
- Use Vtune to collect hardware events
  - IPC, cache misses, branch instruction misses, etc.

<table>
<thead>
<tr>
<th>Features</th>
<th>CONFIGURATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATOM™ BASED STELLARTON BOARD</strong></td>
<td></td>
</tr>
<tr>
<td>Operating System</td>
<td>Linux 2.6.32-41-generic</td>
</tr>
<tr>
<td>Processor</td>
<td>Dual Atom core 1.3GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>1GB</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>32KB instruction cache, 24KB data cache</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>512KB</td>
</tr>
<tr>
<td>Intel® Core™ 2 Quad Q6700</td>
<td></td>
</tr>
<tr>
<td>Operating System</td>
<td>Linux 2.6.32-41-generic</td>
</tr>
<tr>
<td>Processor</td>
<td>Two Core 2.67GHz</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>32KB</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>4MB</td>
</tr>
<tr>
<td>Memory</td>
<td>4GB</td>
</tr>
</tbody>
</table>
Overall Execution Time of EMERALD Workloads

- From less than 1 second (KNN) to 6.5 hours (MRF-DD)
- Hundreds times of speedup is required to meet real time processing for most of the applications

Execution Time (s)

<table>
<thead>
<tr>
<th>Workload</th>
<th>Atom</th>
<th>Core-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-D-offline</td>
<td>205.98</td>
<td></td>
</tr>
<tr>
<td>A-D-online</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD-sep-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD-sep-32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRF-TRW-S-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRF-TRW-S-16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRF-dd-dyn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRF-dd-static</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRF-gra-low</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRF-gra-high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bullet-SB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bullet-Phy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blender-SB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blender-clothe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMAX-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMAX-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smart-headlight</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KNN-linear</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KNN-k-d-tree</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

15 seconds audio

720 frames

Less than 1 second

Ave
L2 Misses Per Kilo Instructions of EMERALD

- Mostly low MPKI; HMAX is 0.04
- Increasing cache capacity does not improve performance

Bar chart showing L2 MPKI for different applications and processors.
## Computation Primitives of the EMERALD

<table>
<thead>
<tr>
<th>Name</th>
<th>Computation Primitives Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio denoise</td>
<td>Fast Fourier Transform (FFT) and General Matrix Multiply (GEMM), vector multiplication</td>
</tr>
<tr>
<td>Audio separation</td>
<td>Tree based MRF solving using fixed grid matrix operation, two passes of the matrix</td>
</tr>
<tr>
<td>MRF-TRW-S</td>
<td>Tree based MRF solving with two passes of the matrix, maximum (minimum), vector operation</td>
</tr>
<tr>
<td>MRF-dd</td>
<td>Continuous sub-gradient optimization for MRF solving, vector based floating point operations</td>
</tr>
<tr>
<td>MRF-graph-cut</td>
<td>Network-flow-based MRF solving, vector based operations on floating point number</td>
</tr>
<tr>
<td>Bullet engine</td>
<td>Matrix operations, graphic operations for display</td>
</tr>
<tr>
<td>Blender 3D</td>
<td>Matrix operations and vector based operations</td>
</tr>
<tr>
<td>HMAX</td>
<td>Down sampling of an image matrix, Gabor filter, correlation, maximum</td>
</tr>
<tr>
<td>Gist</td>
<td>Scaling, integration, comparison, Gaussian filter, Gabor filter</td>
</tr>
<tr>
<td>KNN</td>
<td>Euclidean square distance, floating point number operations</td>
</tr>
<tr>
<td>Smart Headlight</td>
<td>An image pixel summation and moment prediction using floating point numbers</td>
</tr>
</tbody>
</table>
Conclusion

- New compute-intensive applications are emerging in home, retail, and automotive
- Meeting real time processing of the emerging applications faces many challenges
- Heterogeneous SoC designs are desirable for these applications but facing many challenges
- The EMERALD workloads represent a suite of compute-intensive applications from computer vision and machine learning algorithms in low power devices