Heterogeneous Embedded Computer Architectures and Programming Paradigms for Enabling Internet of Things (IoT)

Charles Liu, Ph.D.,
Professor,
Department of Electrical Engineering and Computer Engineering
California State University, Los Angeles
cliu@calstatela.edu
From <https://mobilunity.com/blog/iot-developer-salary-rates/>
Figure 1. The IoT market will be massive

THE INTERNET OF THINGS

Connected devices (billions)

Devices, machines, and things are becoming more intelligent

Edge Computing

The ability to do advanced on-device processing and analytics is referred to as “edge computing.”

*Edge computing is a counterpart to the cloud computing*

Edge computing provides new possibilities in IoT applications, machine learning for tasks such as object detection, face recognition, language processing, and obstacle avoidance.
Instead of sending streams of images/videos to the cloud for processing, in-situ pre-processing is performed.

Advantages: Saving in network and computing resources, reducing latency, improving security and privacy (personally identifiable information vs. demographic information)

[E.g.] Proactive in-car service - natural language interface using Edge computing allows smart speakers to react more quickly by interpreting voice instructions locally.
Heterogeneous computer architectures are adopted for edge computing - integrating diverse engines such as CPUs, GPUs and DSPs — in IoT devices so that different workloads are assigned to the most efficient compute engine, thus improving performance and power efficiency.

[E.g.] The Hexagon DSP with Qualcomm Hexagon Vector eXtensions on Snapdragon 835 has been shown to offer a 25X improvement in energy efficiency and an 8X improvement in performance when compared against running the same workloads (GoogleNet Inception Network) on the Qualcomm Kryo CPU.

Convolutional Neural Network CNN Implementation on Altera FPGA using OpenCL

https://www.youtube.com/watch?v=78Qd5t-Mn0s
Heterogeneous Computer Architectures

- CPUs
- GPUs
- Vector Processors
- Image/Signal Processors
- FPGAs
Suppose you want to add two vectors of numbers. There are many ways to spell this – programming paradigms.

C uses a loop spelling

```c
for(i=0;i<n;++i) a[i]=b[i]+c[i];
```

Matlab uses a vector spelling

```matlab
a=b+c;
```
SIMD uses a "short vector" spelling

```c
void add(uint32_t *a, uint32_t *b, uint32_t *c, int n) {
  for(int i=0; i<n; i+=4) {
    //compute c[i], c[i+1], c[i+2], c[i+3]
    uint32x4_t b4 = vld1q_u32(b+i);
    uint32x4_t c4 = vld1q_u32(c+i);
    uint32x4_t a4 = vaddq_u32(b4,c4);
    vst1q_u32(a+i,a4);
  }
}
```

SIMT uses a "scalar" spelling

```c
__global__ void add(float *a, float *b, float *c) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  a[i]=b[i]+c[i]; //no loop!
}
```
NXP S32V
Each engine has a best efficiency on certain type of functions. To let the complete system working at highest efficiency each engine needs to work in parallel in pipeline mode.

Typical ISP functions
- Black Level and Dead Pixel processing
- Black level, Vignetting
- Exposure control, color balance analysis
- Geometric distortion corrections, chromatic aberration
- HDR
- De-mosaic Bayer pattern
- RGB->YUV420, channel gain
- Spatial de-noise
- Gamma correction
- Image Scaling

Typical APEX functions
- Whatever Kernel based filter:
  - Sobel X
  - Median
  - Histogram
  - Integral Image
  - FAST
  - BRIEF
  - ARB
  - FIR
  - Kirsh
  - Laplace filter
  - Gaussian
  - RGB to YUV
  - YUV 2 RGB
  - YUV 420 to RGB565
  - Harris Corner
  - Shi–Tomasi Corner
  - Lens Correction
# Programming Paradigms

<table>
<thead>
<tr>
<th></th>
<th>OpenACC</th>
<th>OpenMP</th>
<th>OpenCL</th>
<th>CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallelism</strong></td>
<td>- data parallelism</td>
<td>- data parallelism</td>
<td>- data parallelism</td>
<td>- data parallelism</td>
</tr>
<tr>
<td></td>
<td>- asynchronous task parallelism</td>
<td>- asynchronous task parallelism</td>
<td>- asynchronous task parallelism</td>
<td>- asynchronous task parallelism</td>
</tr>
<tr>
<td></td>
<td>- device only</td>
<td>- host and device</td>
<td>- host and device</td>
<td>- device only</td>
</tr>
<tr>
<td><strong>Architecture abstraction</strong></td>
<td>- memory hierarchy</td>
<td>- memory hierarchy</td>
<td>- memory hierarchy</td>
<td>- memory hierarchy</td>
</tr>
<tr>
<td></td>
<td>- explicit data mapping and movement</td>
<td>- data and computation binding</td>
<td>- explicit data mapping and movement</td>
<td>- explicit data mapping and movement</td>
</tr>
<tr>
<td><strong>Synchronization</strong></td>
<td>- reduction</td>
<td>- barrier;</td>
<td>- barrier;</td>
<td>- barrier</td>
</tr>
<tr>
<td></td>
<td>- join</td>
<td>- reduction</td>
<td>- reduction</td>
<td></td>
</tr>
<tr>
<td><strong>Framework implementation</strong></td>
<td>compiler directives for C/C++ and Fortran</td>
<td>compiler directives for C/C++ and Fortran</td>
<td>C/C++ extension</td>
<td>C/C++ extension</td>
</tr>
</tbody>
</table>
NVIDIA GUP: GeForce_GTX_480_Fermi - Single Instruction Multiple Thread (SIMT) architecture

https://www.techpowerup.com/reviews/NVIDIA/GeForce_GTX_480_Fermi/
CUDA and OpenCL terminology correspondence.
The SMs schedule and execute threads in lockstep groups of 32 threads called **warps**.
CogniVue's APEX vision processor core architectures

https://www.bdti.com/InsideDSP/2015/06/30/CogniVue
Example vector operator -

```cpp
vec16s a = (vec16s) src[0];
vec16s b = (vec16s) src[1];
vec16s res = a-b;
```
Understand vif

```c
vec08u V0, V1;
vec16u VR;
if (V0 < V1) // condition
{
    VR = V0 + V1; // TRUE
}
else
{
    VR = V0; // FALSE
}
```

CMEM final after executing the velse statement and the second predicated instruction.

![CMEM diagram](image-url)
Typical fast sin can be built with a look-up table with 361 elements and use the angle as the table index.

We can think to build the vector vsin in the same way.

\[ \text{v_angle} \text{ is now a vector and vsin(v_angle)} \text{ returns a vector with} \]

\[ \text{res}(i) = \sin(v\_angle(i)) \]

```plaintext
.vec global .81 VSIN_t V81
0x5500 0x5500 0x5500 0x5500 0x5500 // \sin(-180) * 32768
0xFEE3 0xFEE3 0xFEE3 0xFEE3 0xFEE3 // \sin(-178) * 32768
0xFEE3 0xFEE3 0xFEE3 0xFEE3 0xFEE3 // \sin(-176) * 32768
0x0000 0x0000 0x0000 0x0000 0x0000 // \sin(0) * 32768
0x9A1D 0x9A1D 0x9A1D 0x9A1D 0x9A1D // \sin(1) * 32768
0x023E 0x023E 0x023E 0x023E 0x023E // \sin(2) * 32768
0x9A1D 0x9A1D 0x9A1D 0x9A1D 0x9A1D // \sin(179) * 32768
0x0000 0x0000 0x0000 0x0000 0x0000 // \sin(180) * 32768
```

```c
.vec16s VSIN_t[361];
const vec16s *VSIN = &VSIN_t[0];
vec16s v_sin(vec16s v_angle)
{
    return vload(VSIN, v_angle);
}
```

\[ \text{V_res28} = \sin(\text{v_angle28}) = \sin(-100) = 0x81F2 \text{ that needs to be } (>>15) \]
Field programmable Gate Array (FPGA) is another Choice!
Island-style global FPGA architecture. A unit tile consists of Configurable Logic Block (CLB), Connect Box (CB) and Switch Box (SB).
Xilinx's 16nm UltraScale+
SoC Design Challenges

IP Quality
- Test Methodology
- IP Updates

Verification
- Deep submicron effects
- Integration
- IP verification
- Testing equipment Limitations

Architecture
- Advance Process
- Simulation Models

Tools
- IP Completeness
- system partitioning

Advance Process
- IP reuse
- Time to Market

Power Management
Data streaming

General algorithm execution model
Pipelining
Pipelining

Stage 1 (CPU)
- Do the 1st tile on CPU
- Start second tile

Stage 2 (GPU)
- Pass intermediate data
- Finish 1st tile processing
- Do the last tile on GPU
Data Mapping and Remapping

- Coalescing global memory accesses to minimize number of memory transactions
- Improving memory locality of next-stage thread access
- Improving memory locality of inter-thread accesses

Figure 6: Row-major to column-major transformation.

Figure 8: Diagonal strip matrix transposition.
OpenCL Memory model