Harnessing Parallel Processing to Get from Algorithms to Embedded Vision

Michael Tusch

CEO, Apical

www.apical.co.uk

Embedded vision application

- Want this to work on an embedded device, not a PC
- Want this to work on every pixel
- Want to implement complex video processing

The challenge

How do we go

from this  to this 

With

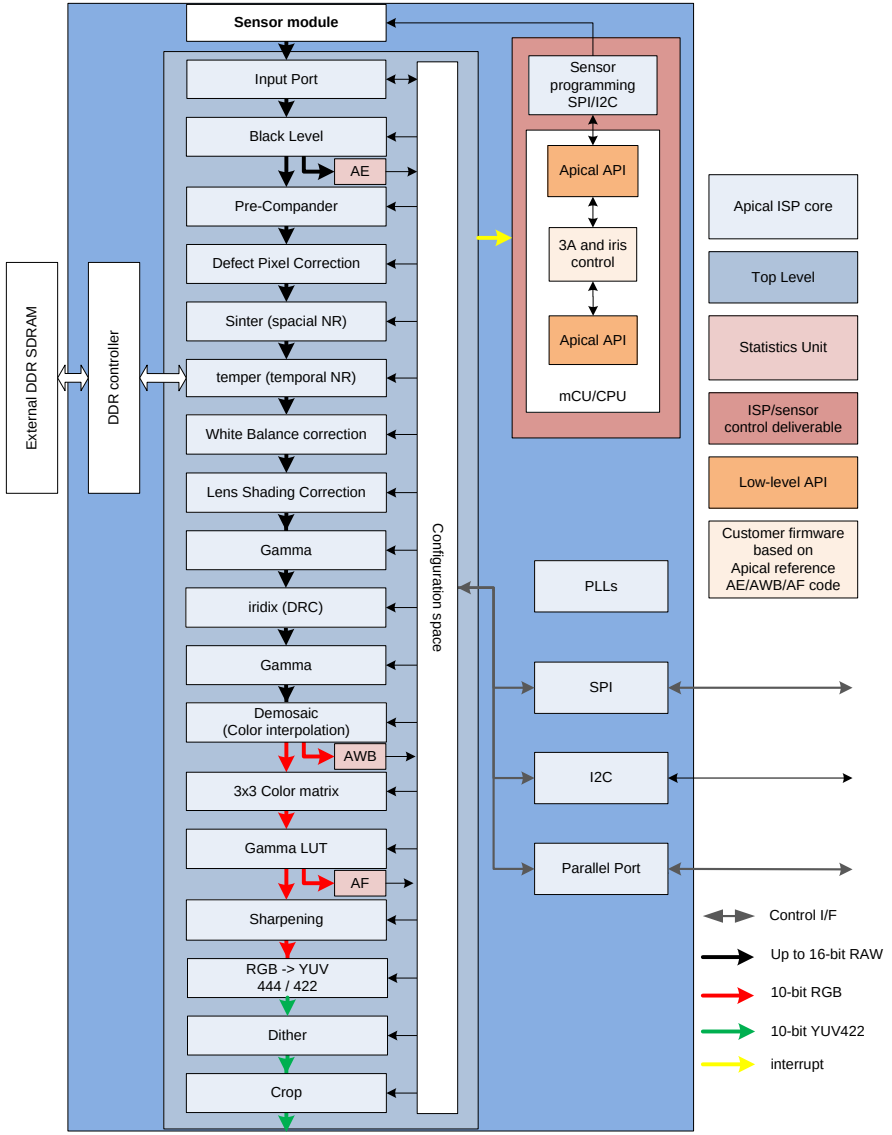
- High resolution and high frame rates
- Efficient computation
- Low power

?

RTL: Ultimate parallelism

Example: Image Signal Processor (ISP)

- Hundreds of operations per pixel
- All processing done in gates (Verilog/VHDL)
- Fully parallel
- Dedicated fast memory
- Up to 500 Megapixel/sec
- Low power



RTL is good but…

- Costly: silicon area devoted just to this function
- Inflexible: once chip is designed, functionality cannot be changed
- Takes a long time to develop and verify

Embedded software is attractive but…

- Serial computation (“one instruction per cycle”)
  - Is slow, even on multi-GHz processors
  - Consumes too much power
  - Requires big compromises

- Need to look for parallelization wherever possible
  - Limited by hardware accelerator resources
  - And skills of designers!

Example accelerators

Accelerator	Comment
ARM Neon	SIMD extensions to ARM instruction set are readily available
TI iMX (e.g. OMAP4)	Powerful SIMD accelerator but access not typically provided to third parties
GPU	Highly parallel accelerator originally designed for graphics now redesigned for general computation

Simple example: pixel luminance calculation

Luminance = 0.299 * Red + 0.587 * Green + 0.114 * Blue

```
uint8_t *p_in_rgb=p_rgm_img+y*width*3;
uint8_t *p_out_lum=p_lum+y*width;
for(x=0;x<width;++x) // 1 loop counting op
{
uint8_t r,g,b,l;
r=p_in_rgb[0]; // 1 load
g=p_in_rgb[1]; // 1 load
b=p_in_rgb[2]; // 1 load
l=(uint8_t)(((r*77)+(g*150)+(b*29))>>8); // 3 muls, 2 adds, 1 shift
p_out_lum[x]=l; // 1 store
p_in_rgb+=3; // 1 add
++p_out_lum; // 1 add
}
```

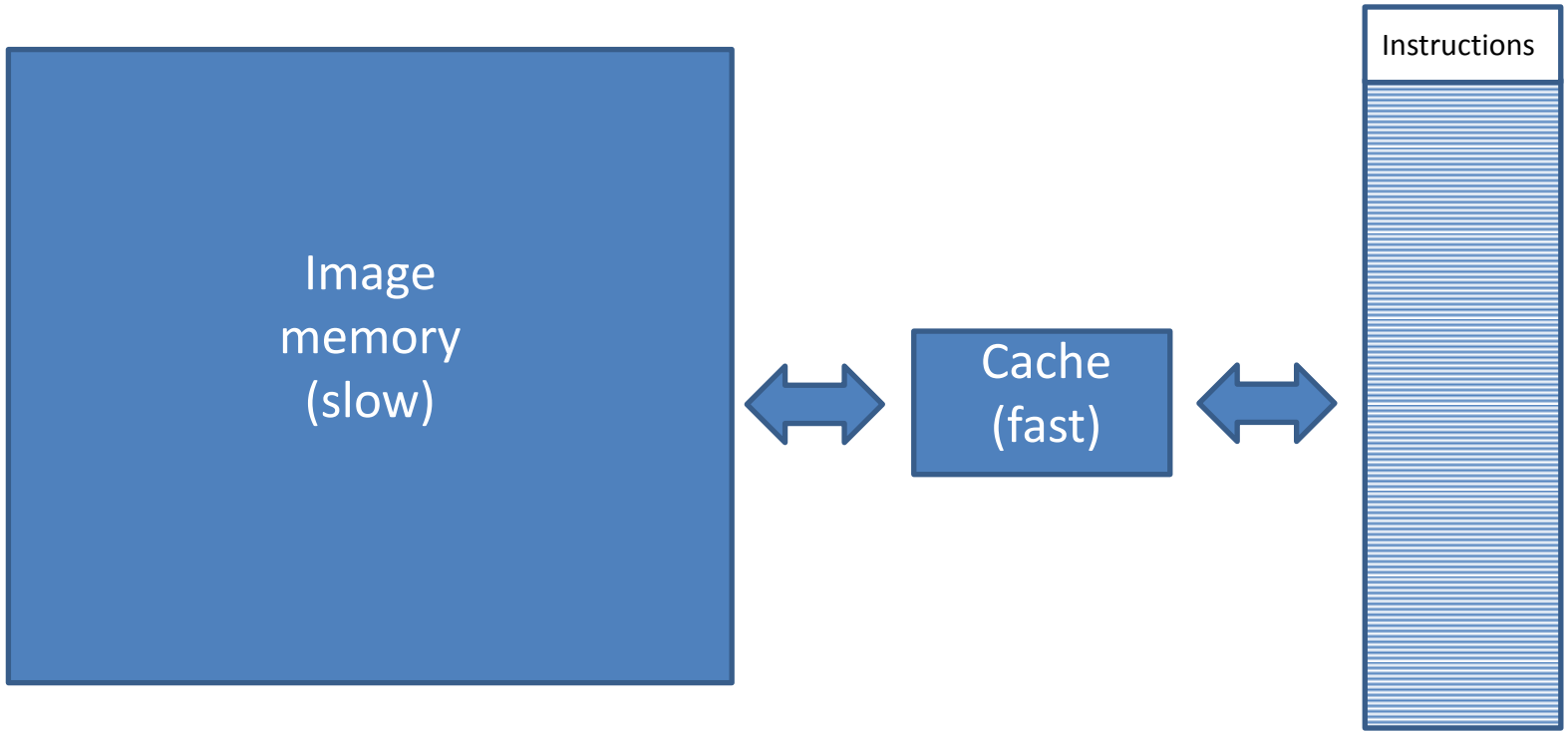
13 operations per pixel

Neon optimization

3.8 operations per pixel

So code should speed up by more than 3x

But we don't see this
--limited by access to fast memory (cache)



Cache reloading is always a bottleneck

More complex example: blur

```
for(y=0;y<height;++y)
{
uint8_t *p_in_lum_m1=p_lum+max(y-1,0)*width;
uint8_t *p_in_lum=p_lum+y*width;
uint8_t *p_in_lum_p1=p_lum+min(y,height-1)*width;
uint8_t *p_out_lum=p_in_lum;
for(x=0;x<width;++x)
{
uint8_t l00,l01,l02;
uint8_t l10,l11,l12;
uint8_t l20,l21,l22;
uint8_t l;
l00=p_in_lum_m1[x=0?0:x-1];
l01=p_in_lum_m1[x];
l02=p_in_lum_m1[x+1==height?x:x+1];
l10=p_in_lum[x=0?0:x-1];
l11=p_in_lum[x];
l12=p_in_lum[x+1==height?x:x+1];
l20=p_in_lum_p1[x=0?0:x-1];
l21=p_in_lum_p1[x];
l22=p_in_lum_p1[x+1==height?x:x+1];
l= l00+ 2*l01+ l02+
2*l10+ 4*l11+2*l12)+
l20+ 2*l21+ l22;
p_out_lum[x]=l; // 1 store
}
}
```

Each line looks like it is read 3x but from cache only need 1 read –**good!**

But, we have register spillage:

Embedded processor normally has only 12 registers available at one time

Spillage adds extra load/store ops: **bad!**

Need to consider breaking this loop into smaller loops

e.g. separate 2D filter into 1D filters

Results

- We should see 300% acceleration – in reality we get 30%
- Processing operations are not the bottleneck
- Memory management and dataflow are

Summary

- Hardware accelerators have the potential to increase operations/cycle by 10-100x
- Very important for embedded programming
- Optimization has to be planned **at the algorithm stage**
- Optimization in these key areas:

Processor operations	Memory usage	Memory bandwidth

One bottleneck is enough to kill performance!

Outlook

- Typical computer vision/image processing algorithms combine hundreds of filters with intermediate data management
- Requires great skill and patience to optimize
  - Even to judge how much optimization is done
- Need better compilers which can genuinely exploit algorithm parallelization and fast memory management