IN5050:
Programming heterogeneous multi-core processors

(M-)JPEG

Parts of the code explained…

January 30, 2020

Why ?

Hmmmm… IN5050 is about programming heterogeneous multi-core processors.

Why video coding?

We want to look at parallelism that is required for everyday tasks…

According to Cisco in 2016, Internet video will in 2017 globally …

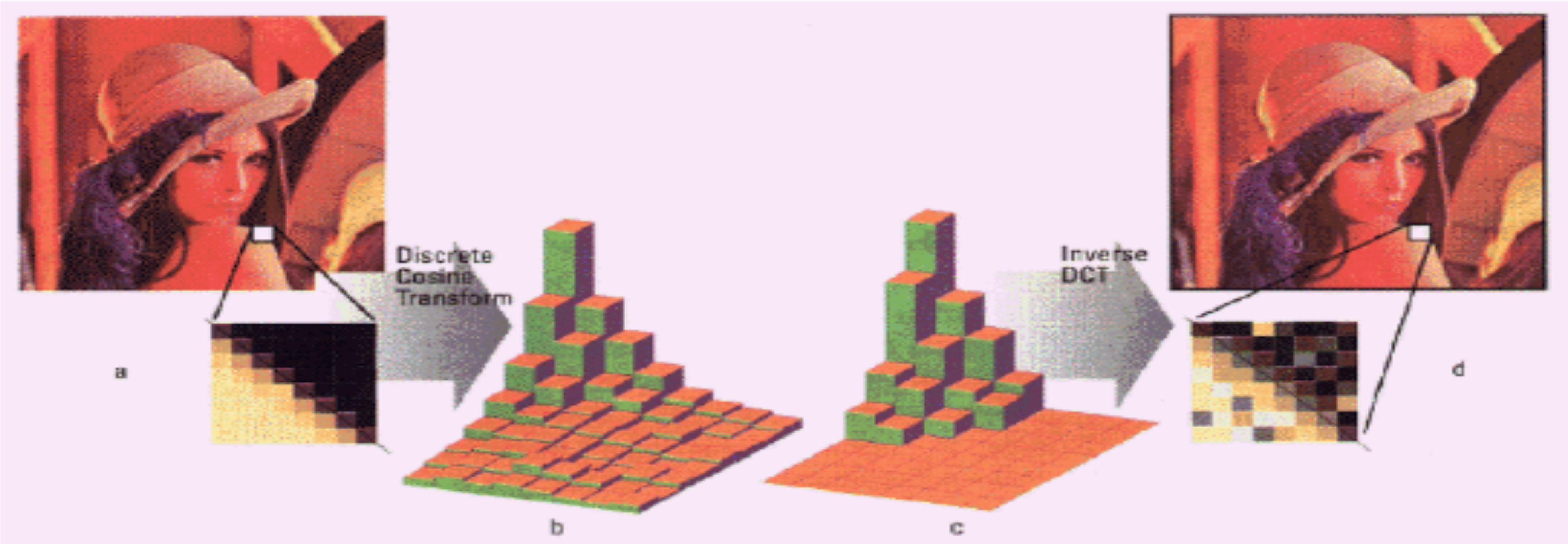
- have a compound annual growth rate of 30%
- reach 62.7 Exabytes per month with 69% of Internet traffic
- have 2 trillion minutes (5 million years) of video content crossing the Internet each month. That's 914,100 minutes of video every second streamed or downloaded …

- Many of the video codec applications are time-critical
- A codec can become memory-bound, CPU-bound, IO-bound

→ opportunities for both data and execution parallelism AND real-world relevant

(Today, we define a video as a sequence of still images – (M)JPEG, more next time)

Data Compression



The human eye

- is good at seeing small differences in brightness over a large area
- not so good at distinguishing the exact strength of a high frequency brightness variation

➔ can reduce the amount of information in the high frequency components

Data Compression

- Alternative description of data requiring less storage and bandwidth

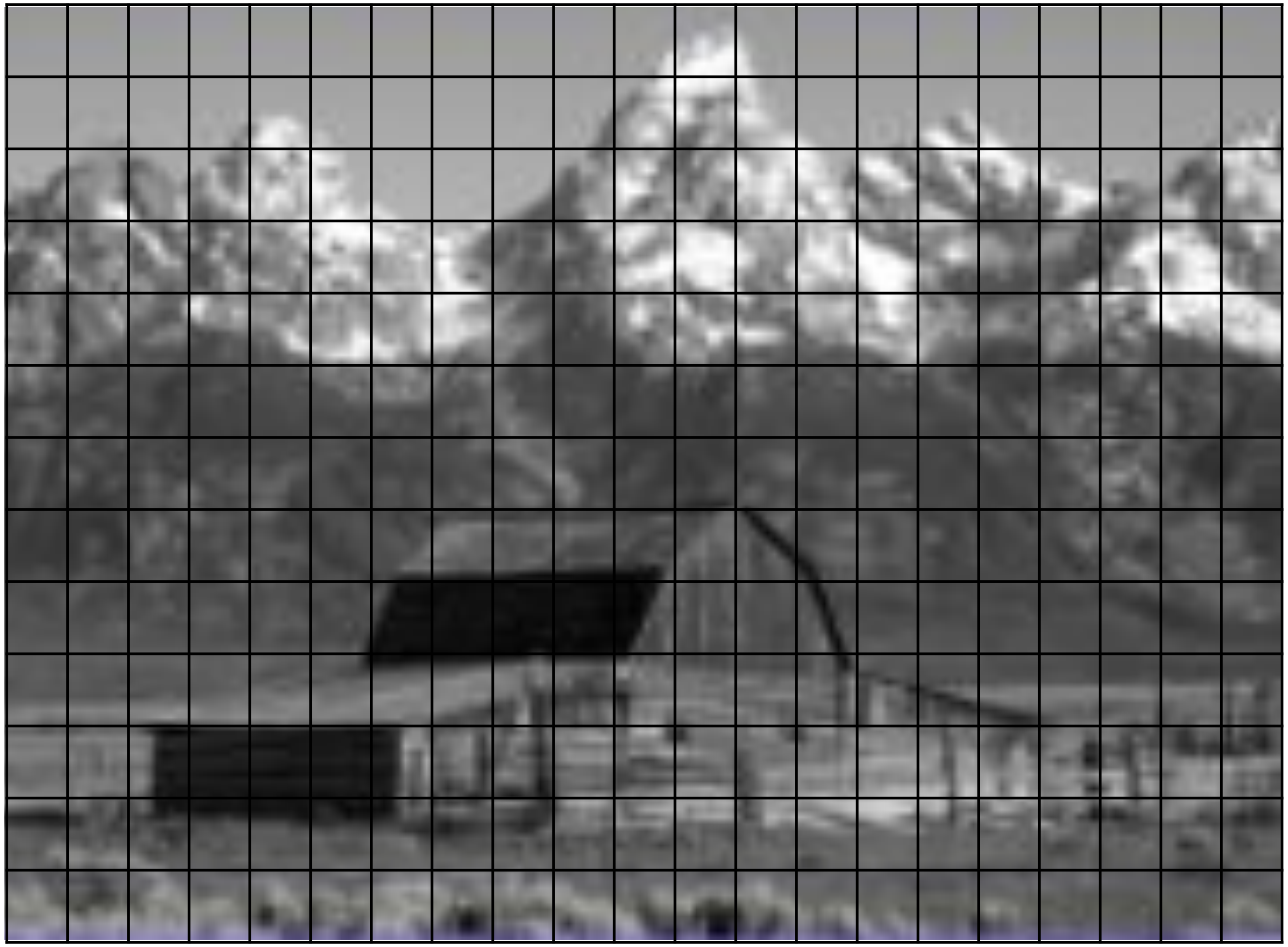
Uncompressed: **1 Mbyte**

Compressed (JPEG)**: 50 Kbyte** (20:1)

... while, for example, a 20 Megapixel camera creates 6016 x 4000 images, in 8-bit RGB that makes more than 72 uncompressed Mbytes per image

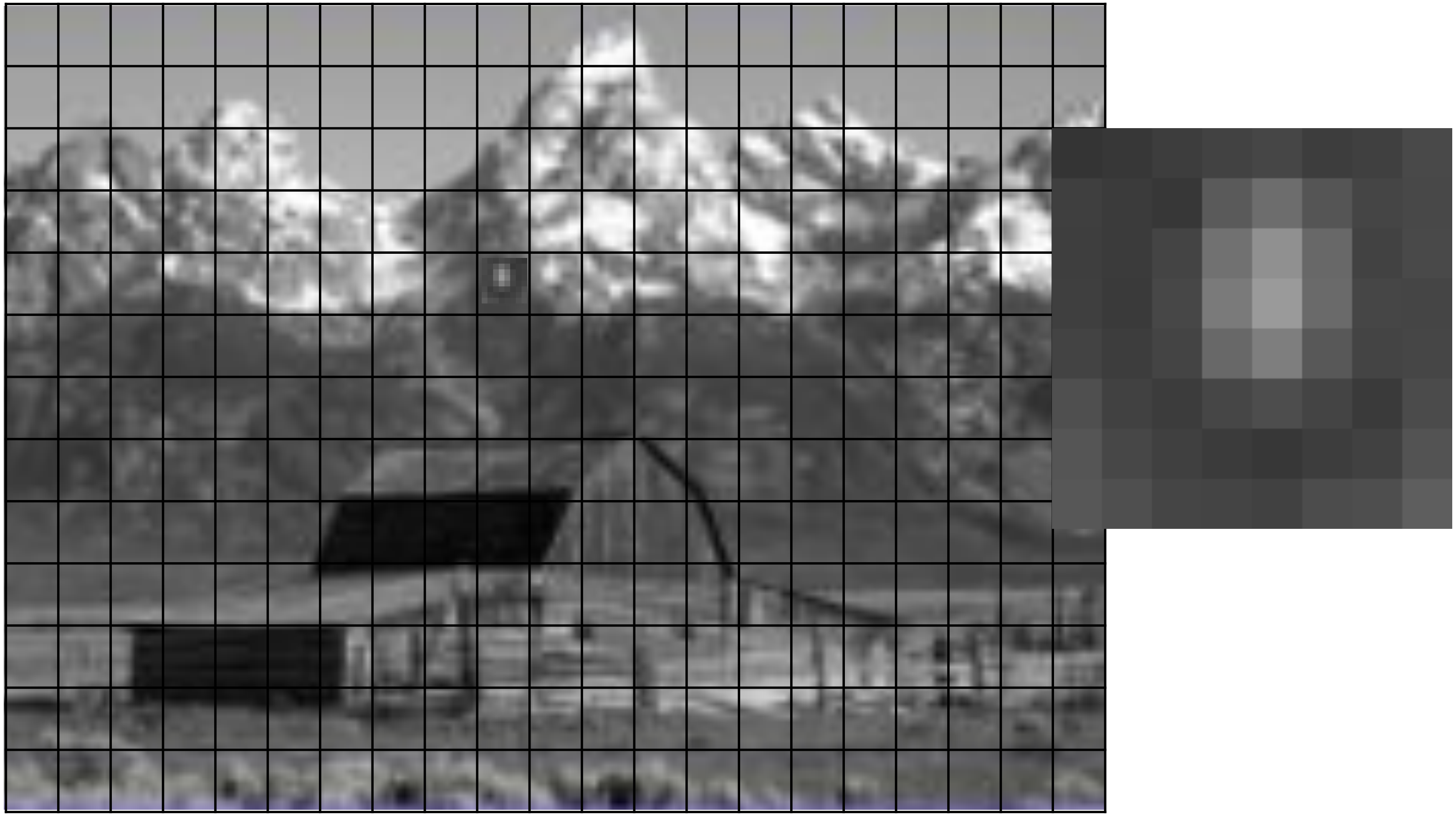
1 - Split each picture in 8x8 blocks

- **Each** sub-picture is divided into 8x8 blocks, number depends on resolution



2 - Discrete cosine transform (DCT)

- Each 8×8 block is converted to a frequency-domain representation, using a normalized, two-dimensional DCT



2 - Discrete cosine transform (DCT)

- Each 8×8 block is converted to a frequency-domain representation, using a normalized, two-dimensional DCT
  - each pixel is represented by a [0, 255]-value
  - each pixel is transformed to a [-128, 127]-value

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

$$\overset{\xrightarrow{x}}{\begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix}} \Big\downarrow y$$

2 - Discrete cosine transform (DCT)

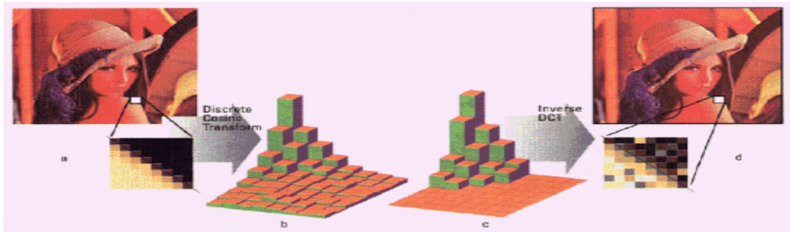
- Each 8 × 8 block is converted to a frequency-domain representation, using a normalized, two-dimensional DCT
  - two-dimensional DCT: $$G_{u,v} = \alpha(u)\alpha(v) \sum_{x=0}^{7} \sum_{y=0}^{7} g_{x,y} \cos\left[\frac{\pi}{8}\left(x + \frac{1}{2}\right) u\right] \cos\left[\frac{\pi}{8}\left(y + \frac{1}{2}\right) v\right]$$
    - $G_{u,v}$ is the DCT at coordinates *(u,v)*
    - u is the horizontal spatial frequency [0,8>
    - v is the vertical spatial frequency [0,8>
    - $g_{x,y}$ is the pixel value at coordinates (x,y)
    - α is a normalizing function: $$\alpha_p(n) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } n = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

Note the rather large value of the top-left corner (DC coefficient). The remaining 63 are AC coefficients. The advantage of the DCT is its tendency to aggregate most of the signal in one corner of the result, as may be seen above.

Compression possible: the following **quantization** step accentuates this effect while simultaneously reducing the overall size of the DCT coefficients

$$\overset{u \longrightarrow}{\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix}} \Big\downarrow v$$

3 - Quantization



- The human eye
  - is good at seeing small differences in brightness over a large area
  - not so good at distinguishing the exact strength of a high frequency brightness variation
  - can reduce the amount of information in the high frequency components

  - simply dividing each component in the frequency domain by a known constant for that component, and then rounding to the nearest integer:

$$B_{j,k} = \mathrm{round}\left(\frac{G_{j,k}}{Q_{j,k}}\right) \text{ for } j = 0,1,2,\cdots,N_1-1; k = 0,1,2,\cdots,N_2-1$$

where $Q_{j,k}$ is a quantization matrix, e.g., for JPEG

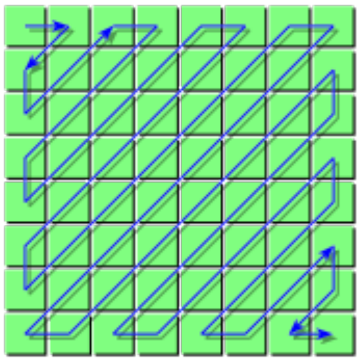
$$\frac{G_{0,0} = -415}{Q_{0,0} = 16} = -25.9375000000 \approx -26$$

16	[illegible]	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

−26	[illegible]3	−6	2	2	−1	0	0
0	−2	−4	1	1	0	0	0
−3	1	5	−1	−1	0	0	0
−4	1	2	−1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

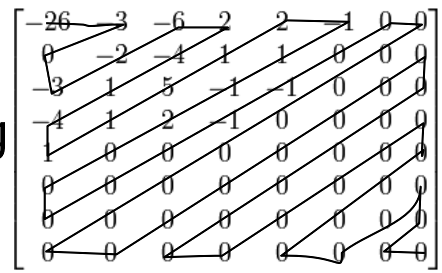
4 - Lossless compression

- The resulting data for all 8 × 8 blocks is further compressed with a loss-less algorithm:
  - organizing numbers in a **zigzag pattern**:

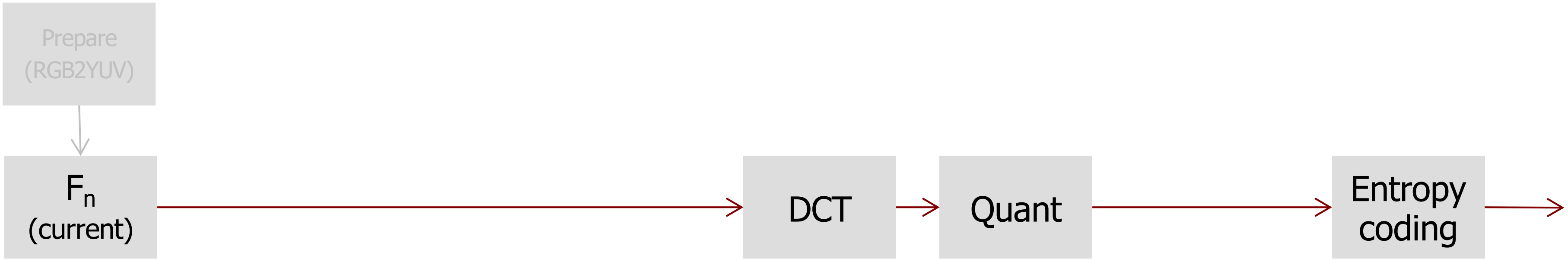


-26, -3, 0, -3, -2, -6, 2, -4, 1, -4, 1, 1, 5, 1, 2, -1, 1, -1, 2,
0, 0, 0, 0, 0, -1, -1, 0 , 0, 0, 0, 0, 0, 0, 0, 0, … , 0, 0

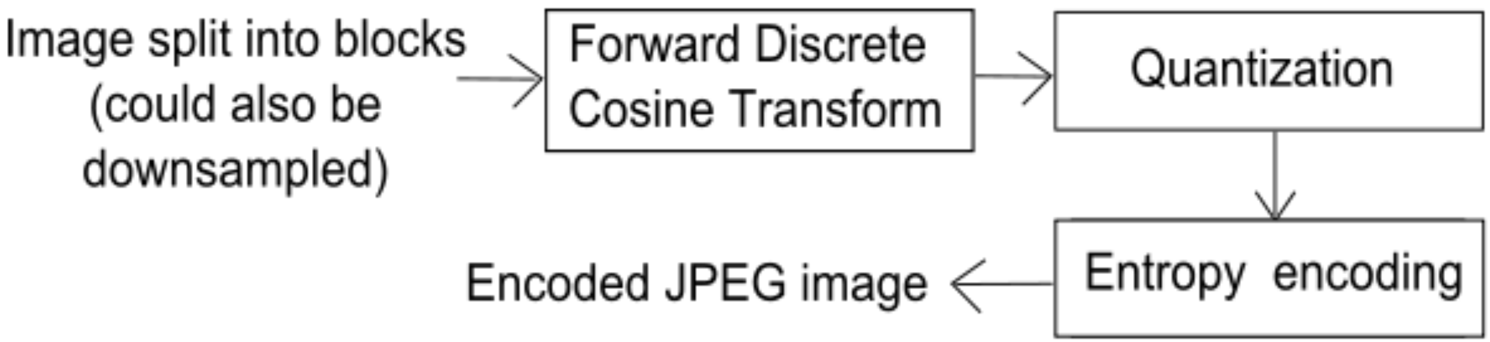
  - Compress using for example run-length or Huffman coding



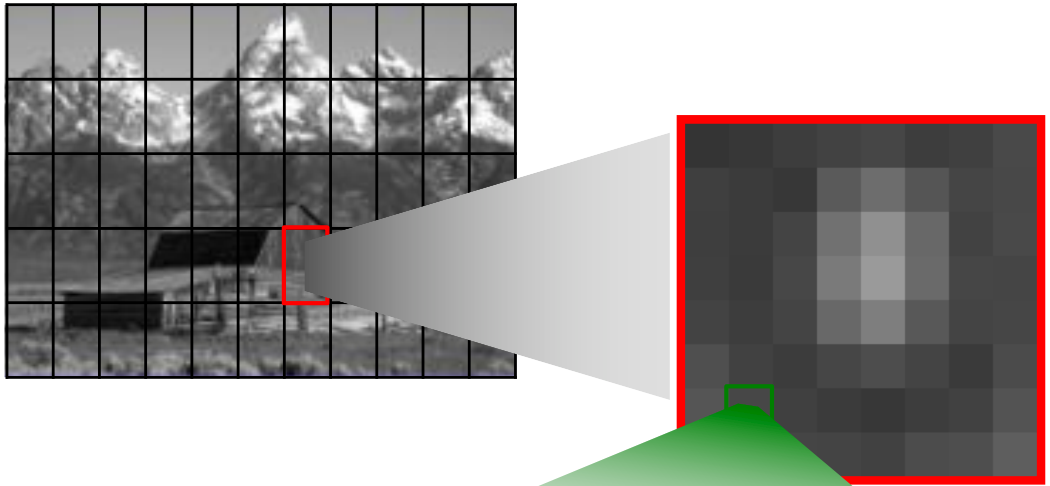
JPEG Encoder Overview



Encoding:



DCT / Quantization



```
// Make 8x8 block of the entire picture
for(y = 0; y < height; y += 8)
  {
     for(x = 0; x < width; x += 8)
     {
        ...
        //Loop through all elements of the block
        for(u = 0; u < 8; ++u)
        {
           for(v = 0; v < 8; ++v)
           {



           }
        }
     }
  }
```

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } u = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

$$G_{u,v} = \alpha(u)\alpha(v) \sum_{x=0}^{7} \sum_{y=0}^{7} g_{x,y} \cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

$g_{x,y} = pixel(x,y) - 128;$

$$B_{j,k} = \text{round}\left(\frac{G_{j,k}}{Q_{j,k}}\right) \quad \text{for } j = 0,1,2,\cdots,N_1-1;\ k = 0,1,2,\cdots,N_2-1$$

(M-)JPEG

SSE / AVX examples

Optimizing DCT


```
// Make 8x8 block of the entire picture
for(y = 0; y < height; y += 8)
  {
    for(x = 0; x < width; x += 8)
    {
      …
      //Loop through all elements of the block
      for(u = 0; u < 8; ++u)
      {
        for(v = 0; v < 8; ++v)
        {
          for(j = 0; j < jj; ++j)  // Inner DCT
            for(i = 0; i < ii; ++i)
            {   // Inner sum of DCT
              float coeff = in_data[(y+j)*width+(x+i)] - 128.0f;
              dct += coeff * (float) (cos(…) * cos(…));
            }


          float a1 = !u ? ISQRT2 : 1.0f;  float a2 = !v ? ISQRT2 : 1.0f;
          /* Scale according to normalizing function */
          dct *= a1*a2/4.0f;

          /* Quantization */

        }
      }
    }
  }
```

DCT – Approach 1 – normalization table

v-ph-dct

	√1/8	√2/8	√2/8	√2/8	√2/8	√2/8	√2/8	√2/8
√1/8	ISQRT2 * ISQRT2 / 4	1 * ISQRT2 / 4	1 * ISQRT2 / 4	1 * ISQRT2 / 4	1 * ISQRT2 / 4	1 * ISQRT2 / 4	1 * ISQRT2 / 4	1 * ISQRT2 / 4
√2/8	ISQRT2 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4
√2/8	ISQRT2 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4
√2/8	ISQRT2 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4
√2/8	ISQRT2 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4
√2/8	ISQRT2 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4
√2/8	ISQRT2 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4
√2/8	ISQRT2 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4	1 * 1 / 4

$$\alpha_p(n) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } n = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

```
// Make 8x8 block of the entire picture
for(y = 0; y < height; y += 8)
  {
    for(x = 0; x < width; x += 8)
 {
   ...
   //Loop through all elements of the block
   for(u = 0; u < 8; ++u)
   {
      for(v = 0; v < 8; ++v)
      {
         for(j = 0; j < jj; ++j)  // Inner DCT
            for(i = 0; i < ii; ++i)
            {  // Inner sum of DCT
               float coeff = in_data[(y+j)*width+(x+i)] - 128.0f;
               dct += coeff * (float) (cos(...) * cos(...));
            }

          /* Scale according to normalizing function */
         dct *= dct_norm_table[u][v];

         /* Quantization */

      }
   }
 }
}
```

DCT – Approach 2 – cosine table

v-ph-dct

	0	1	2	3	4	5	6	7
0	C(0, 0)	C(0, 1)	C(0, 2)	C(0, 3)	C(0, 4)	C(0, 5)	C(0, 6)	C(0, 6)
1	C(1, 0)	C(1, 1)	C(1, 2)	C(1, 3)	C(1, 4)	C(1, 5)	C(1, 6)	C(1, 7)
2	C(2, 0)	C(2, 1)	C(2, 2)	…	…	…	…	…
3	C(3, 0)	C(3, 1)	…	C(3, 3)	…	…	…	…
4	C(4, 0)	C(4, 1)	…	…	C(4, 4)	…	…	…
5	C(5, 0)	C(5, 1)	…	…	…	C(5, 5)	…	…
6	C(6, 0)	C(6, 1)	…	…	…	…	C(6, 6)	…
7	C(7, 0)	C(7, 1)	…	…	…	…	…	C(7, 7)

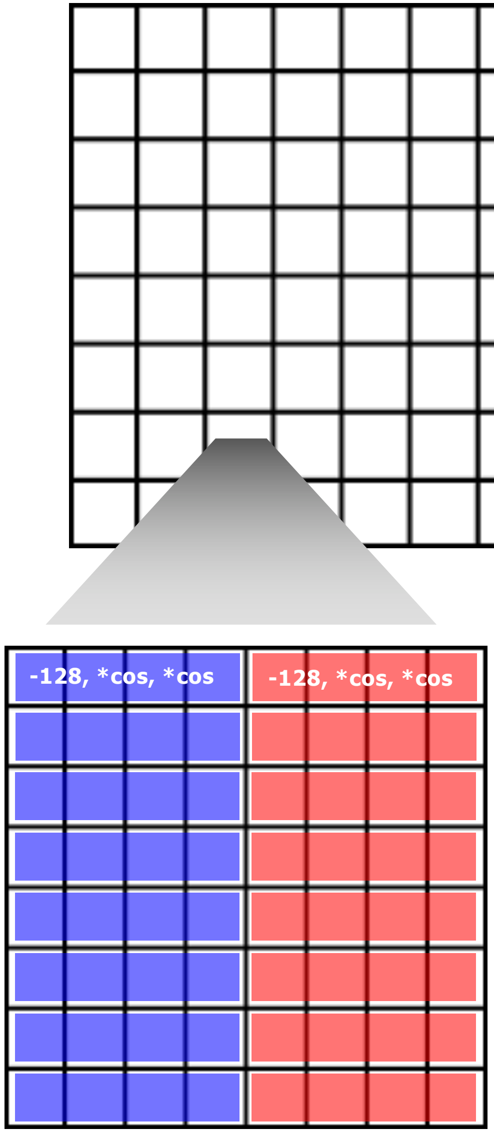
$$G_{u,v} = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

C(**x**,**u**) = cos((2***x**+1)***u***PI/16.0f);

```
// Make 8x8 block of the entire picture
for(y = 0; y < height; y += 8)
  {
    for(x = 0; x < width; x += 8)
  {
      ...
      //Loop through all elements of the block
      for(u = 0; u < 8; ++u)
      {
        for(v = 0; v < 8; ++v)
        {
          for(j = 0; j < jj; ++j)  // Inner DCT
            for(i = 0; i < ii; ++i)
            {  // Inner sum of DCT
              float coeff = in_data[(y+j)*width+(x+i)] - 128.0f;
              dct += coeff * costable[i][u] * costable[j][v];
            }

          /* Scale according to normalizing function */
          dct *= dct_norm_table[u][v];

          /* Quantization */

        }
      }
    }
  }
```

DCT – Approach 3 – SSE entire row

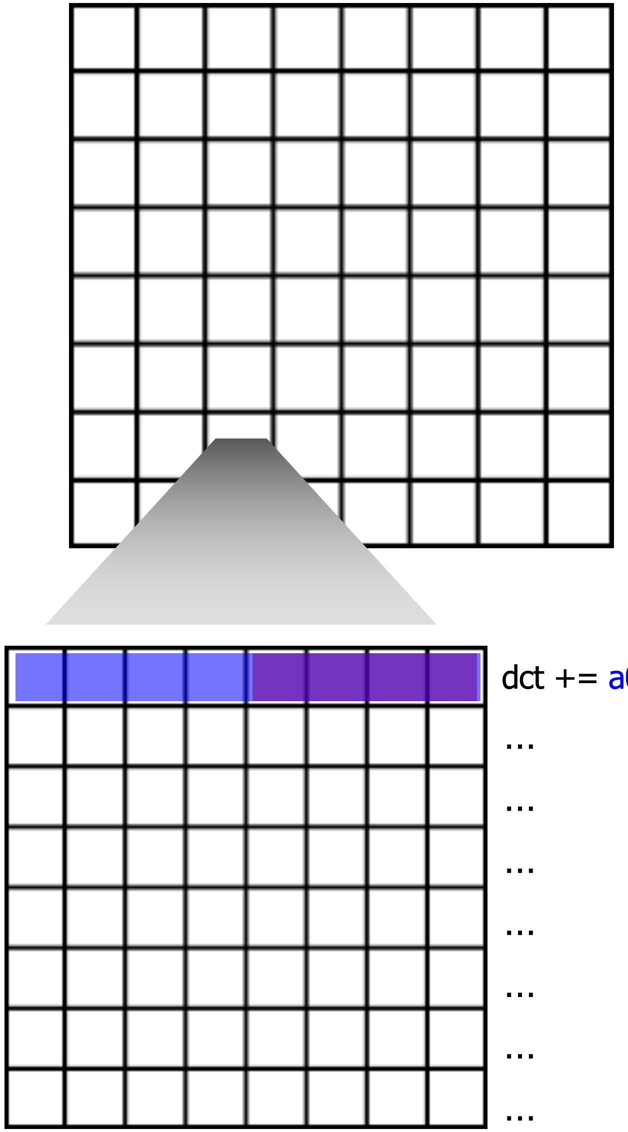
v-ph-dct




```
dct += a0+a1+a2+a3+b0+b1+b2+b3
dct += a0+a1+a2+a3+b0+b1+b2+b3
dct += a0+a1+a2+a3+b0+b1+b2+b3
dct += a0+a1+a2+a3+b0+b1+b2+b3
dct += a0+a1+a2+a3+b0+b1+b2+b3
dct += a0+a1+a2+a3+b0+b1+b2+b3
dct += a0+a1+a2+a3+b0+b1+b2+b3
dct += a0+a1+a2+a3+b0+b1+b2+b3
```

```
// Make 8x8 block of the entire picture
for(y = 0; y < height; y += 8)
  {
    for(x = 0; x < width; x += 8)
    {
      ...
      //Loop through all elements of the block
      for(u = 0; u < 8; ++u)
      {
        for(v = 0; v < 8; ++v)
        {
          for(j = 0; j < jj; ++j)  // Inner DCT
            for(i = 0; i < ii; ++i)
            {   // Inner sum of DCT
              float coeff = in_data[(y+j)*width+(x+i)] - 128.0f;
              dct += coeff * costable[i][u] * costable[j][v];
            }

          /* Scale according to normalizing function */
          dct *= dct_norm_table[u][v];

          /* Quantization */

        }
      }
    }
  }
```

DCT – Approach 4 – AVX entire row



dct += a0+a1+a2+a3+a4+a5+a6+a7

...

...

...

...

...

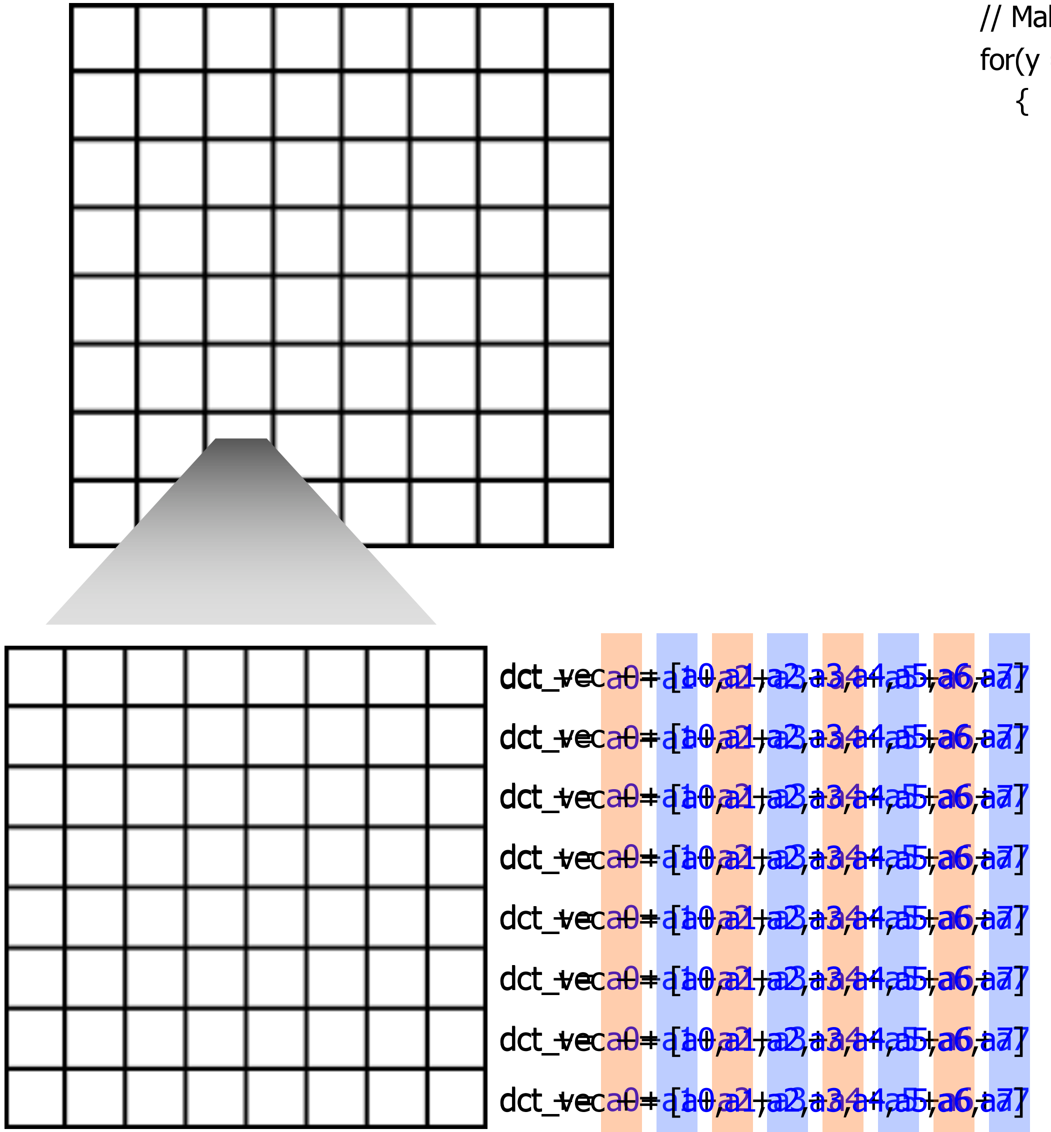
...

...

```
// Make 8x8 block of the entire picture
for(y = 0; y < height; y += 8)
  {
    for(x = 0; x < width; x += 8)
    {
      ...
      //Loop through all elements of the block
      for(u = 0; u < 8; ++u)
      {
        for(v = 0; v < 8; ++v)
        {
          for(j = 0; j < jj; ++j)  // Inner DCT
            for(i = 0; i < ii; ++i)
            {   // Inner sum of DCT
              float coeff = in_data[(y+j)*width+(x+i)] - 128.0f;
              dct += coeff * costable[i][u] * costable[j][v];
            }


          /* Scale according to normalizing function */
          dct *= dct_norm_table[u][v];


          /* Quantization */

        }
      }
    }
  }
```

DCT – Approach 5 – AVX entire row, add

v-ph-dct2-avx



```
// Make 8x8 block of the entire picture
for(y = 0; y < height; y += 8)
  {
    for(x = 0; x < width; x += 8)
    {
      …
      //Loop through all elements of the block
      for(u = 0; u < 8; ++u)
      {
        for(v = 0; v < 8; ++v)
        {
          for(j = 0; j < jj; ++j)  // Inner DCT
            for(i = 0; i < ii; ++i)
            {  // Inner sum of DCT
              float coeff = in_data[(y+j)*width+(x+i)] - 128.0f;
              dct += coeff * costable[i][u] * costable[j][v];
            }


          /* Scale according to normalizing function */
          dct *= dct_norm_table[u][v];


          /* Quantization */

        }
      }
    }
  }
```

dct += dct_vec[0]+dct_vec[0]+…+dct_vec[7]


That's all Folks!
kallak