ESE532, Final review

Fall 2019, Fall 2018

Fall 2019 Final exam review

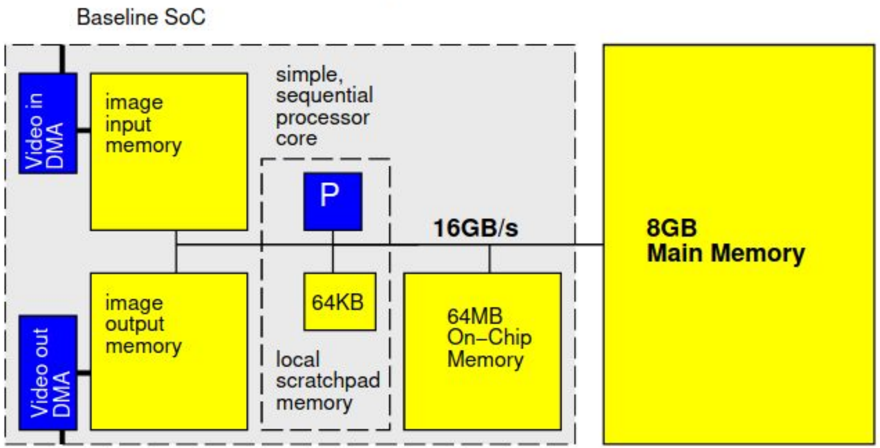
Read the assumptions!

We had/have/will have similar assumptions for all the exams: e.g. ignore array indices computations

- For simplicity throughout, we will treat non-memory indexing adds (subtracts count as adds), compares, abs, shifts, and multplies as the only compute operations. We'll assume the other operations take negligible time or can be run in parallel (ILP) with the adds, abs, shift, multiplies, and memory operations. (Some consequences: You may ignore loop and conditional overheads in processor runtime estimates; you may ignore computations in array indecies.)
- Baseline (simple, sequential) processor can execute one multiply, compare, shift, abs, or add per cycle and runs at 1 GHz.
- Data can be transfered between pairs of memory (including main memory) at 16 GB/s when streamed in chunks of at least 2048B. Assume `for` loops that only copy data can be auto converted into streaming operations.
- Non-streamed access to the main memory takes 100 cycles and can move 8B.
- Non-streamed access to image and 64 MB on-chip memories takes 10 cycles and can move 8B.
- Baseline processor has a local scratchpad memory that holds 64KB of data. Data can be streamed into the local scratchpad memory at 16 GB/s. Non-streamed accesses to the local scratchpad memory take 1 cycle.
- Baseline processor is 1 mm^2 of silicon including its 64KB local scratchpad.
- By default, all arrays live in the 8 GB main memory.
- `image_in` and `image_out` live in the respective image input and image output memories.
- Arrays for `sintable`, `costable` and viewpoints (`old_viewpoint`, `viewpoint`) live in local scratchpad memory.
- Assume scalar (non-array) variables can live in registers.
- Assume all additions are associative.
- Assume comparisons, adds, and multiplies take 1 ns when implemented in hardware accelerator, so fully pipelined accelerators also run at 1 GHz. A compare-mux operation can also be implemented in 1 ns. Consider abs and shift free in hardware.
- Data can be transfered to accelerator local memory at the same 16 GB/s when streamed in chunks of at least 2048B.

Fall 2019 Final exam review

System overview




```
void augment_frame() {
  uint16_t raw[HEIGHT][WIDTH][COLORS]; // uint16_t for 16b (2 byte) color per pixel
  uint16_t augment[HEIGHT][WIDTH][COLORS];
  uint16_t augmented[HEIGHT][WIDTH][COLORS];
  uint16_t old_viewpoint[VPARAMS];
  uint16_t viewpoint[VPARAMS];
  uint16_t *tmp_viewpoint;
  get_image(raw);
  tmp_viewpoint=old_viewpoint;
  old_viewpoint=viewpoint;
  viewpoint=tmp_viewpoint;
  compute_viewpoint(raw,reference,old_viewpoint,viewpoint);
  render_augmentation(viewpoint,overlay,augment);
  merge_frames(reference,viewpoint,raw,augment,augmented);
  send_image(augmented);
}
```

```c
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t  *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
      for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
        for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
          for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
          {
            uint64_t score=0;
            for (int iy=0;i<HEIGHT;iy++) // loop F
              for (int ix=0;i<WIDTH;ix++) // loop G
              {
                uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
                uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, ysca
                if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
                  for (int c=0;c<COLORS;c++) // loop H
                    score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
              }
            if (score<best_score)
              {
                best_score=score;
                current[VP_ROT]=rot;
                current[VP_X]=x;
                current[VP_Y]=y;
                current[VP_XS]=xs;
                current[VP_YS]=ys;
              }
          }
  }
}
```

```c
void render_augmentation(int16_t  *current, uint16_t ***overlay, uint16_t ***image)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;i<HEIGHT;iy++) // loop I
    for (int ix=0;i<WIDTH;ix++) // loop J
      image[iy][ix]=UNMAPPED; // assume this runs like streaming data copy
  for (int iy=0;i<HEIGHT;iy++) // loop K
    for (int ix=0;i<WIDTH;ix++) // loop L
      {
        uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
        uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscale
        if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
            && (overlay[ty][tx][MASK]>0))
             for (int c=0;c<COLORS;c++) // loop M
                image[iy][ix][c]=overlay[ty][tx][c];
      }

}
```

```
void merge_frames(uint16_t ***reference, int16_t  *current,
                  uint16_t ***image, uint16_t ***augment, uint16_t ***augmented)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;i<HEIGHT;iy++) // loop N
      for (int ix=0;i<WIDTH;ix++) // loop O
        {
          uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
          uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y;// +8 for xscale, yscale
          if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
              && (augment[iy][ix]!=UNMAPPED))
            {
              uint32_t diff=0;
              for (int c=0;c<COLORS;c++) // loop P
                diff+=abs(image[iy][ix][c]-reference[ty][tx][c]);
              if (diff<THRESH)
                for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=augment[iy][ix][c];
              else
                for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
            }
          else
            for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
        }
}

void get_image(uint16_t ***image)
{
  for (int iy=0;i<HEIGHT;iy++)
    for (int ix=0;i<WIDTH;ix++)
      for (int c=0;c<COLORS;c++)
        image[iy][ix][c]=image_in[iy][ix][c];
}

void send_image(uint16_t ***image)
{
  for (int iy=0;i<HEIGHT;iy++)
    for (int ix=0;i<WIDTH;ix++)
      for (int c=0;c<COLORS;c++)
        image_out[iy][ix][c]=image[iy][ix][c];

}
```

For Q1,
Red: explanation for compute ops
Orange: explanation for memory ops

Question 1

(a) Based only on the resource bound for compute operations, what throughput can a simple, single processor system achieve [answer in frames/second, or equivalently, `augment_frame` calls per second]?

get_image	all DMA	0
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (12 + 3 \times 3)$ cycles	5.6 s
render_augmentation	$4096 \times 2048 \times 12$ cycles	0.10 s
merge_frames	$4096 \times 2048 \times (12 + 3 \times 3)$ cycles	0.18 s
send_image	all DMA	0
Total		5.9 s

0.17 frames/second

(b) Based only on the resource bound for memory operations, what throughput can a simple, single processor system achieve [answer in frames/second]?

get_image	$\frac{4096\times2048\times3\times2}{16\times10^9}$	0.0031 s
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (2 \times 100)$ cycles image[iy][ix] and reference[ty][tx] is single read ignore small terms sin/costable, current update	54 s
render_augmentation	$\frac{4096\times2048\times3\times2}{16\times10^9}+$ $4096 \times 2048 \times 2 \times 100$ cycles overlay[ty][tx] including mask is single read image[iy][ix] is single write	1.7s
merge_frames	$4096 \times 2048 \times 4 \times 100$ cycles	3.4 s
send_image	$\frac{4096\times2048\times3\times2}{16\times10^9}$	0.0031
Total		59 s

0.017 frames/second

16bit, 2B

```
void get_image(uint16_t ***image)
{
  for (int iy=0;i<HEIGHT;iy++)
    for (int ix=0;i<WIDTH;ix++)
      for (int c=0;c<COLORS;c++)
        image[iy][ix][c]=image_in[iy][ix][c];
}
```

For Q1,
Red: explanation for compute ops
Orange: explanation for memory ops

Question 1

(a) Based only on the resource bound for compute operations, what throughput can a simple, single processor system achieve [answer in frames/second, or equivalently, `augment_frame` calls per second]?

get_image	all DMA	0
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (12 + 3 \times 3)$ cycles	5.6 s
render_augmentation	$4096 \times 2048 \times 12$ cycles	0.10 s
merge_frames	$4096 \times 2048 \times (12 + 3 \times 3)$ cycles	0.18 s
send_image	all DMA	0
Total		5.9 s

0.17 frames/second

(b) Based only on the resource bound for memory operations, what throughput can a simple, single processor system achieve [answer in frames/second]?

get_image	$\frac{4096 \times 2048 \times 3 \times 2}{16 \times 10^9}$	0.0031 s
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (2 \times 100)$ cycles image[iy][ix] and reference[ty][tx] is single read ignore small terms sin/costable, current update	54 s
render_augmentation	$\frac{4096 \times 2048 \times 3 \times 2}{16 \times 10^9} +$ $4096 \times 2048 \times 2 \times 100$ cycles overlay[ty][tx] including mask is single read image[iy][ix] is single write	1.7s
merge_frames	$4096 \times 2048 \times 4 \times 100$ cycles	3.4 s
send_image	$\frac{4096 \times 2048 \times 3 \times 2}{16 \times 10^9}$	0.0031
Total		59 s

0.017 frames/second

```c
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
      for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
        for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
          for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
          {
            uint64_t score=0;
            for (int iy=0;i<HEIGHT;iy++) // loop F
              for (int ix=0;i<WIDTH;ix++) // loop G
              {
                uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
                uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
                if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
                  for (int c=0;c<COLORS;c++) // loop H
                    score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
              }
            if (score<best_score)
              {
                best_score=score;
                current[VP_ROT]=rot;
                current[VP_X]=x;
                current[VP_Y]=y;
                current[VP_XS]=xs;
                current[VP_YS]=ys;
              }
          }
  }
}
```

2^5

4096*2048

6+6

3*3

Read from main memory
2^5*4096*2048*(2*100)

Note that 3*16b=6B,(image, reference are each 16b data) and the processor allows 8B transfer for non-streaming

Question 1

(a) Based only on the resource bound for compute operations, what throughput can a simple, single processor system achieve [answer in frames/second, or equivalently, `augment_frame` calls per second]?

get_image	all DMA	0
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (12 + 3 \times 3)$ cycles	5.6 s
render_augmentation	$4096 \times 2048 \times 12$ cycles	0.10 s
merge_frames	$4096 \times 2048 \times (12 + 3 \times 3)$ cycles	0.18 s
send_image	all DMA	0
Total		5.9 s

0.17 frames/second

(b) Based only on the resource bound for memory operations, what throughput can a simple, single processor system achieve [answer in frames/second]?

get_image	$\frac{4096\times2048\times3\times2}{16\times10^9}$	0.0031 s
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (2 \times 100)$ cycles image[iy][ix] and reference[ty][tx] is single read ignore small terms sin/costable, current update	54 s
render_augmentation	$\frac{4096\times2048\times3\times2}{16\times10^9}+$ $4096 \times 2048 \times 2 \times 100$ cycles overlay[ty][tx] including mask is single read image[iy][ix] is single write	1.7s
merge_frames	$4096 \times 2048 \times 4 \times 100$ cycles	3.4 s
send_image	$\frac{4096\times2048\times3\times2}{16\times10^9}$	0.0031
Total		59 s

0.017 frames/second

```
void render_augmentation(int16_t  *current, uint16_t ***overlay, uint16_t ***image)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;i<HEIGHT;iy++) // loop I
    for (int ix=0;i<WIDTH;ix++) // loop J
      image[iy][ix]=UNMAPPED; // assume this runs like streaming data copy
  for (int iy=0;i<HEIGHT;iy++) // loop K
    for (int ix=0;i<WIDTH;ix++) // loop L
      {
        uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
        uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscale
        if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
            && (overlay[ty][tx][MASK]>0))
          for (int c=0;c<COLORS;c++) // loop M
            image[iy][ix][c]=overlay[ty][tx][c];
      }

}
```

4096*2048*3*2/16*10^9

6+6

4096*2048*2*100

Note that 3*16b=6B, and the processor allows 8B transfer for non-streaming

Question 1

(a) Based only on the resource bound for compute operations, what throughput can a simple, single processor system achieve [answer in frames/second, or equivalently, `augment_frame` calls per second]?

get_image	all DMA	0
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (12 + 3 \times 3)$ cycles	5.6 s
render_augmentation	$4096 \times 2048 \times 12$ cycles	0.10 s
merge_frames	$4096 \times 2048 \times (12 + 3 \times 3)$ cycles	0.18 s
send_image	all DMA	0
Total		5.9 s

0.17 frames/second

(b) Based only on the resource bound for memory operations, what throughput can a simple, single processor system achieve [answer in frames/second]?

get_image	$\frac{4096 \times 2048 \times 3 \times 2}{16 \times 10^9}$	0.0031 s
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (2 \times 100)$ cycles image[iy][ix] and reference[ty][tx] is single read ignore small terms sin/costable, current update	54 s
render_augmentation	$\frac{4096 \times 2048 \times 3 \times 2}{16 \times 10^9} +$ $4096 \times 2048 \times 2 \times 100$ cycles overlay[ty][tx] including mask is single read image[iy][ix] is single write	1.7s
merge_frames	$4096 \times 2048 \times 4 \times 100$ cycles	3.4 s
send_image	$\frac{4096 \times 2048 \times 3 \times 2}{16 \times 10^9}$	0.0031
Total		59 s

0.017 frames/second

```c
void merge_frames(uint16_t ***reference, int16_t  *current,
                  uint16_t ***image, uint16_t ***augment, uint16_t ***augmented)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;i<HEIGHT;iy++) // loop N
    for (int ix=0;i<WIDTH;ix++) // loop O
      {
        uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
        uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y;// +8 for xscale, yscale
        if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
            && (augment[iy][ix]!=UNMAPPED))
          {
            uint32_t diff=0;
            for (int c=0;c<COLORS;c++) // loop P
              diff+=abs(image[iy][ix][c]-reference[ty][tx][c]);
            if (diff<THRESH)
              for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=augment[iy][ix][c];
            else
              for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
          }
        else
          for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
      }
}
```

6+6

3*3

4096*2048*4*100

The processor allows 8B transfer for non-streaming

Question 1

(a) Based only on the resource bound for compute operations, what throughput can a simple, single processor system achieve [answer in frames/second, or equivalently, `augment_frame` calls per second]?

get_image	all DMA	0
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (12+3\times 3)$ cycles	5.6 s
render_augmentation	$4096 \times 2048 \times 12$ cycles	0.10 s
merge_frames	$4096 \times 2048 \times (12+3\times 3)$ cycles	0.18 s
send_image	all DMA	0
Total		5.9 s

0.17 frames/second

(b) Based only on the resource bound for memory operations, what throughput can a simple, single processor system achieve [answer in frames/second]?

get_image	$\frac{4096\times2048\times3\times2}{16\times10^9}$	0.0031 s
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (2\times 100)$ cycles image[iy][ix] and reference[ty][tx] is single read ignore small terms sin/costable, current update	54 s
render_augmentation	$\frac{4096\times2048\times3\times2}{16\times10^9}+$ $4096 \times 2048 \times 2 \times 100$ cycles overlay[ty][tx] including mask is single read image[iy][ix] is single write	1.7s
merge_frames	$4096 \times 2048 \times 4 \times 100$ cycles	3.4 s
send_image	$\frac{4096\times2048\times3\times2}{16\times10^9}$	0.0031
Total		59 s

0.017 frames/second

```
void send_image(uint16_t ***image)
{
  for (int iy=0;i<HEIGHT;iy++)
    for (int ix=0;i<WIDTH;ix++)
      for (int c=0;c<COLORS;c++)
        image_out[iy][ix][c]=image[iy][ix][c];

}
```

16bit, 2B

Question 2

2. Based on the simple, single processor mapping from Question 1:

(a) What function is the bottleneck? (circle one)

`get_image`
`compute_viewpoint`
`render_augmentation`
`merge_frames`
`send_image`

(b) What is the Amdahl's Law speedup if you only accelerate the identified function?

$\frac{64.9}{5.4} = 11$

(a) Based only on the resource bound for compute operations, what throughput can a simple, single processor system achieve [answer in frames/second, or equivalently, `augment_frame` calls per second]?

get_image	all DMA	0
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (12 + 3 \times 3)$ cycles	5.6 s
render_augmentation	$4096 \times 2048 \times 12$ cycles	0.10 s
merge_frames	$4096 \times 2048 \times (12 + 3 \times 3)$ cycles	0.18 s
send_image	all DMA	0
Total		5.9 s

0.17 frames/second

(b) Based only on the resource bound for memory operations, what throughput can a simple, single processor system achieve [answer in frames/second]?

get_image	$\frac{4096 \times 2048 \times 3 \times 2}{16 \times 10^9}$	0.0031 s
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (2 \times 100)$ cycles image[iy][ix] and reference[ty][tx] is single read ignore small terms sin/costable, current update	54 s
render_augmentation	$\frac{4096 \times 2048 \times 3 \times 2}{16 \times 10^9} +$ $4096 \times 2048 \times 2 \times 100$ cycles overlay[ty][tx] including mask is single read image[iy][ix] is single write	1.7s
merge_frames	$4096 \times 2048 \times 4 \times 100$ cycles	3.4 s
send_image	$\frac{4096 \times 2048 \times 3 \times 2}{16 \times 10^9}$	0.0031
Total		59 s

0.017 frames/second

Total = 5.9 + 59 = 64.9

`compute_viewpoint` = 59.6

64.9/5.3 = 11

Question 3

3. Data Parallel and Reduce: Classify Loops

Loop	Data Parallel?	Associative Reduce?	Must be Sequential?
A		X	
F		X	
K	X		
N	X		
Z (main)			X

Z must compute new viewpoint from one iteration/image before starting computation on next image.
A is a min-reduce on best_score.
F is a sum-reduce for score.
Computation for image[iy][ix] (K) and augmented[iy][ix] (N) are each independent of other elements of the respective arrays.

For 2^5 cases, `score` can be computed in parallel. Once there are 2^5 `score` values available, we need to find min value in tree-fashion(associative reduce).

2^5

```
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t  *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
      for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
        for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
          for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
          {
            uint64_t score=0;
            for (int iy=0;i<HEIGHT;iy++) // loop F
              for (int ix=0;i<WIDTH;ix++) // loop G
              {
                uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
                uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
                if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
                  for (int c=0;c<COLORS;c++) // loop H
                    score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
              }
            if (score<best_score)
              {
                best_score=score;
                current[VP_ROT]=rot;
                current[VP_X]=x;
                current[VP_Y]=y;
                current[VP_XS]=xs;
                current[VP_YS]=ys;
              }
          }
  }
}
```

Question 3

3. Data Parallel and Reduce: Classify Loops

Loop	Data Parallel?	Associative Reduce?	Must be Sequential?
A		X	
F		X	
K	X		
N	X		
Z (main)			X

Z must compute new viewpoint from one iteration/image before starting computation on next image.

A is a min-reduce on best_score.

F is a sum-reduce for score.

Computation for image[iy][ix] (K) and augmented[iy][ix] (N) are each independent of other elements of the respective arrays.

score, is accumulated for 4096*2048*3 times, and this can be done in tree-fashion(associative reduce).

```
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                        int16_t  *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
      for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
        for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
          for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
          {
            uint64_t score=0;
            for (int iy=0;i<HEIGHT;iy++) // loop F
              for (int ix=0;i<WIDTH;ix++) // loop G
                {
                  uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
                  uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
                  if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
                    for (int c=0;c<COLORS;c++) // loop H
                      score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
                }
            if (score<best_score)
              {
                best_score=score;
                current[VP_ROT]=rot;
                current[VP_X]=x;
                current[VP_Y]=y;
                current[VP_XS]=xs;
                current[VP_YS]=ys;
              }
          }
  }
}
```

4096*2048*3

Question 3

3. Data Parallel and Reduce: Classify Loops

Loop	Data Parallel?	Associative Reduce?	Must be Sequential?
A		X	
F		X	
K	X		
N	X		
Z (main)			X

Z must compute new viewpoint from one iteration/image before starting computation on next image.

A is a min-reduce on best_score.

F is a sum-reduce for score.

Computation for image[iy][ix] (K) and augmented[iy][ix] (N) are each independent of other elements of the respective arrays.

```
void render_augmentation(int16_t  *current, uint16_t ***overlay, uint16_t ***image)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;i<HEIGHT;iy++) // loop I
    for (int ix=0;i<WIDTH;ix++) // loop J
      image[iy][ix]=UNMAPPED; // assume this runs like streaming data copy
  for (int iy=0;i<HEIGHT;iy++) // loop K
    for (int ix=0;i<WIDTH;ix++) // loop L
        {
          uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
          uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscale
          if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
              && (overlay[ty][tx][MASK]>0))
               for (int c=0;c<COLORS;c++) // loop M
                  image[iy][ix][c]=overlay[ty][tx][c];
        }

}
```

Question 3

3. Data Parallel and Reduce: Classify Loops

Loop	Data Parallel?	Associative Reduce?	Must be Sequential?
A		X	
F		X	
K	X		
N	X		
Z (main)			X

Z must compute new viewpoint from one iteration/image before starting computation on next image.

A is a min-reduce on best_score.

F is a sum-reduce for score.

Computation for image[iy][ix] (K) and augmented[iy][ix] (N) are each independent of other elements of the respective arrays.

```
void merge_frames(uint16_t ***reference, int16_t  *current,
                  uint16_t ***image, uint16_t ***augment, uint16_t ***augmented)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;i<HEIGHT;iy++) // loop N
      for (int ix=0;i<WIDTH;ix++) // loop O
        {
          uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
          uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y;// +8 for xscale, yscale
          if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
              && (augment[iy][ix]!=UNMAPPED))
            {
              uint32_t diff=0;
              for (int c=0;c<COLORS;c++) // loop P
                diff+=abs(image[iy][ix][c]-reference[ty][tx][c]);
              if (diff<THRESH)
                for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=augment[iy][ix][c];
              else
                for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
            }
          else
            for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
        }
}
```

Question 3

```
void main() {
  while (true) { // loop Z
    augment_frame();
  }
}
```

3. Data Parallel and Reduce: Classify Loops

Loop	Data Parallel?	Associative Reduce?	Must be Sequential?
A		X	
F		X	
K	X		
N	X		
Z (main)			X

Z must compute new viewpoint from one iteration/image before starting computation on next image.

A is a min-reduce on best_score.

F is a sum-reduce for score.

Computation for image[iy][ix] (K) and augmented[iy][ix] (N) are each independent of other elements of the respective arrays.

```
void augment_frame() {
  uint16_t raw[HEIGHT][WIDTH][COLORS]; // uint16_t for 16b (2 byte) color per pixel
  uint16_t augment[HEIGHT][WIDTH][COLORS];
  uint16_t augmented[HEIGHT][WIDTH][COLORS];
  uint16_t old_viewpoint[VPARAMS];
  uint16_t viewpoint[VPARAMS];
  uint16_t *tmp_viewpoint;
  get_image(raw);
  tmp_viewpoint=old_viewpoint;
  old_viewpoint=viewpoint;
  viewpoint=tmp_viewpoint;
  compute_viewpoint(raw,reference,old_viewpoint,viewpoint);
  render_augmentation(viewpoint,overlay,augment);
  merge_frames(reference,viewpoint,raw,augment,augmented);
  send_image(augmented);
}
```

Previous loop's viewpoint becomes old_viewpoint

Question 4

```c
void get_image(uint16_t ***image)
{
  for (int iy=0;i<HEIGHT;iy++)
    for (int ix=0;i<WIDTH;ix++)
      for (int c=0;c<COLORS;c++)
        image[iy][ix][c]=image_in[iy][ix][c];
}
```

4. Data Streaming:

   (a) Can the producer and consumer operate concurrently on the same input image? or must the consumer work on a different (earlier) input image? ("Same Image?" column)

   (b) How big (minimum size) does the buffer (or other data storage space) need to be between the identified loops in order to allow the loops to profitably execute concurrently?

   (Hint: Based on data dependencies, under what scenarios and granularity can the identified loops act as a producer-consumer pair in a pipeline.)

Loop Pair	(a) Same Image?	(b) Size (bytes)
get_image → compute_viewpoint	N	48 MB
compute_viewpoint→render_augmentation	N	10 B
render_augmentation → merge_frames	Y	6 B
merge_frames → send_image	Y	6 B

image, 4096*2048*3*2B

image has to be ready for compute_viewpoint to start!

```c
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t  *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
      for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
        for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
          for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
            {
              uint64_t score=0;
              for (int iy=0;i<HEIGHT;iy++) // loop F
                for (int ix=0;i<WIDTH;ix++) // loop G
                  {
                    uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
                    uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
                    if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
                      for (int c=0;c<COLORS;c++) // loop H
                        score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
                  }
              if (score<best_score)
                {
                  best_score=score;
                  current[VP_ROT]=rot;
                  current[VP_X]=x;
                  current[VP_Y]=y;
                  current[VP_XS]=xs;
                  current[VP_YS]=ys;
                }
            }
  }
}
```

Question 4

4. Data Streaming:

   (a) Can the producer and consumer operate concurrently on the same input image? or must the consumer work on a different (earlier) input image? ("Same Image?" column)

   (b) How big (minimum size) does the buffer (or other data storage space) need to be between the identified loops in order to allow the loops to profitably execute concurrently?

   (Hint: Based on data dependencies, under what scenarios and granularity can the identified loops act as a producer-consumer pair in a pipeline.)

Loop Pair	(a) Same Image?	(b) Size (bytes)
get_image → compute_viewpoint	N	48 MB
compute_viewpoint→render_augmentation	N	10 B
render_augmentation → merge_frames	Y	6 B
merge_frames → send_image	Y	6 B

current, 5*2B

current has to be ready for render_augmentation to start!

```
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t  *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
      for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
        for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
          for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
            {
              uint64_t score=0;
              for (int iy=0;i<HEIGHT;iy++) // loop F
                for (int ix=0;i<WIDTH;ix++) // loop G
                  {
                    uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
                    uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
                    if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
                      for (int c=0;c<COLORS;c++) // loop H
                        score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
                  }
              if (score<best_score)
                {
                  best_score=score;
                  current[VP_ROT]=rot;
                  current[VP_X]=x;
                  current[VP_Y]=y;
                  current[VP_XS]=xs;
                  current[VP_YS]=ys;
                }
            }
  }
}
```

```
void render_augmentation(int16_t  *current, uint16_t ***overlay, uint16_t ***image)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;i<HEIGHT;iy++) // loop I
    for (int ix=0;i<WIDTH;ix++) // loop J
      image[iy][ix]=UNMAPPED; // assume this runs like streaming data copy
  for (int iy=0;i<HEIGHT;iy++) // loop K
    for (int ix=0;i<WIDTH;ix++) // loop L
      {
        uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
        uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscale
        if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
            && (overlay[ty][tx][MASK]>0))
          for (int c=0;c<COLORS;c++) // loop M
            image[iy][ix][c]=overlay[ty][tx][c];
      }

}
```

Question 4

4. Data Streaming:

   (a) Can the producer and consumer operate concurrently on the same input image? or must the consumer work on a different (earlier) input image? ("Same Image?" column)

   (b) How big (minimum size) does the buffer (or other data storage space) need to be between the identified loops in order to allow the loops to profitably execute concurrently?

   (Hint: Based on data dependencies, under what scenarios and granularity can the identified loops act as a producer-consumer pair in a pipeline.)

Loop Pair	(a) Same Image?	(b) Size (bytes)
get_image → compute_viewpoint	N	48 MB
compute_viewpoint→render_augmentation	N	10 B
render_augmentation → merge_frames	Y	6 B
merge_frames → send_image	Y	6 B

augment(1 pixel), 3*2B

Whenever image[iy][ix] is ready, send it to merge_frames! (dataflow)

```
void render_augmentation(int16_t  *current, uint16_t ***overlay, uint16_t ***image)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;i<HEIGHT;iy++) // loop I
    for (int ix=0;i<WIDTH;ix++) // loop J
      image[iy][ix]=UNMAPPED; // assume this runs like streaming data copy
  for (int iy=0;i<HEIGHT;iy++) // loop K
    for (int ix=0;i<WIDTH;ix++) // loop L
        {
         uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
         uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscale
         if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
             && (overlay[ty][tx][MASK]>0))
              for (int c=0;c<COLORS;c++) // loop M
                image[iy][ix][c]=overlay[ty][tx][c];
        }

}
```

```
void merge_frames(uint16_t ***reference, int16_t  *current,
                  uint16_t ***image, uint16_t ***augment, uint16_t ***augmented)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;i<HEIGHT;iy++) // loop N
      for (int ix=0;i<WIDTH;ix++) // loop O
        {
         uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
         uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y;// +8 for xscale, yscale
         if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
             && (augment[iy][ix]!=UNMAPPED))
           {
             uint32_t diff=0;
             for (int c=0;c<COLORS;c++) // loop P
               diff+=abs(image[iy][ix][c]-reference[ty][tx][c]);
             if (diff<THRESH)
               for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=augment[iy][ix][c];
             else
               for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
           }
         else
           for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
        }
}
```

Question 4

4. Data Streaming:

   (a) Can the producer and consumer operate concurrently on the same input image? or must the consumer work on a different (earlier) input image? ("Same Image?" column)

   (b) How big (minimum size) does the buffer (or other data storage space) need to be between the identified loops in order to allow the loops to profitably execute concurrently?

   (Hint: Based on data dependencies, under what scenarios and granularity can the identified loops act as a producer-consumer pair in a pipeline.)

Loop Pair	(a) Same Image?	(b) Size (bytes)
`get_image` → `compute_viewpoint`	N	48 MB
`compute_viewpoint`→`render_augmentation`	N	10 B
`render_augmentation` → `merge_frames`	Y	6 B
`merge_frames` → `send_image`	Y	6 B

augmented(1 pixel), 3*2B

```
void merge_frames(uint16_t ***reference, int16_t  *current,
                  uint16_t ***image, uint16_t ***augment, uint16_t ***augmented)
{
  uint16_t rot=current[VP_ROT];
  uint16_t x=current[VP_X];
  uint16_t y=current[VP_Y];
  uint16_t xs=current[VP_XS];
  uint16_t ys=current[VP_YS];
  int16_t sr=sintable[rot]; // result is a fraction
  int16_t cr=costable[rot];
  for (int iy=0;i<HEIGHT;iy++) // loop N
      for (int ix=0;i<WIDTH;ix++) // loop O
        {
         uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
         uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y;// +8 for xscale, yscale
         if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT)
             && (augment[iy][ix]!=UNMAPPED))
           {
             uint32_t diff=0;
             for (int c=0;c<COLORS;c++) // loop P
               diff+=abs(image[iy][ix][c]-reference[ty][tx][c]);
             if (diff<THRESH)
               for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=augment[iy][ix][c];
             else
               for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
           }
         else
           for (int c=0;c<COLORS;c++) augmented[iy][ix][c]=image[iy][ix][c];
        }
}
```

```
void send_image(uint16_t ***image)
{
  for (int iy=0;i<HEIGHT;iy++)
    for (int ix=0;i<WIDTH;ix++)
      for (int c=0;c<COLORS;c++)
        image_out[iy][ix][c]=image[iy][ix][c];

}
```

In `merge_frames`, it's sequentially accessing `augmented` from `iy=0, ix=0` to `iy=HEIGHT-1, ix=WIDTH-1`
Whenever `augmented[iy][ix]` is ready, send it to `send_image`! (dataflow)

Question 5

5. Latency Bound

(a) What is the critical path (latency bound) for the entire computation as captured in the augment_frame function?

compute_viewpoint	read sintable, costable	1
	multiply by sine, cos	1
	add sin/cos terms	1
	scale	1
	(shifts for free)	0
	add offset	1
	read image and reference	100
	subtract	1
	(abs for free)	0
	sum reduce	$\log_2(4096 \times 2048 \times 3) = 25$
	min reduce	$\log_2(2^5) = 5$
render_augmentation	read sintable, costable	1
	multiply by sine, cos	1
	add sin/cos terms	1
	scale	1
	(shifts for free)	0
	add offset	1
	read overlay	100
	(don't write image, just use it below)	0
merge_frames	compute tx, ty with above	0
	(reads, if needed, happen with overlay above)	0
	subtract	1
	(abs for free)	0
	sum reduce	$\log_2(3) = 2$
	(don't write image, just use for output)	0
Total		244

From assumption in p.5. (they are simply wiring in HW)

```c
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t  *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
      for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
        for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
          for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
            {
              uint64_t score=0;
              for (int iy=0;i<HEIGHT;iy++) // loop F
                for (int ix=0;i<WIDTH;ix++) // loop G
                  {
                    uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
                    uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
                    if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
                      for (int c=0;c<COLORS;c++) // loop H
                        score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
                  }
              if (score<best_score)
                {
                  best_score=score;
                  current[VP_ROT]=rot;
                  current[VP_X]=x;
                  current[VP_Y]=y;
                  current[VP_XS]=xs;
                  current[VP_YS]=ys;
                }
            }
  }
}
```

Question 5

5. Latency Bound

(a) What is the critical path (latency bound) for the entire computation as captured in the `augment_frame` function?

`compute_viewpoint`	read sintable, costable	1
	multiply by sine, cos	1
	add sin/cos terms	1
	scale	1
	(shifts for free)	0
	add offset	1
	read image and reference	100
	subtract	1
	(abs for free)	0
	sum reduce	$\log_2(4096 \times 2048 \times 3) = 25$
	min reduce	$\log_2(2^5) = 5$
`render_augmentation`	read sintable, costable	1
	multiply by sine, cos	1
	add sin/cos terms	1
	scale	1
	(shifts for free)	0
	add offset	1
	read overlay	100
	(don't write image, just use it below)	0
`merge_frames`	compute tx, ty with above	0
	(reads, if needed, happen with overlay above)	0
	subtract	1
	(abs for free)	0
	sum reduce	$\log_2(3) = 2$
	(don't write image, just use for output)	0
Total		244

(b) What is the latency bound Iteration Internal (II) for the `main` computation? (Hint: builds on part (a).)
136
Only need to compute new viewpoint.

In order to start the next iteration, we need a new `viewpoint`

Question 6

6. Consider rewriting the body of `compute_viewpoint` to minimize the memory resource bound by exploiting the scratchpad memory and the 64 MB on-chip memory and streaming data tranfers.

(a) Identify new temporary arrays allocated to scratchpad memory or 64MB on-chip memory (and specify which memory each new array is in).

```
uint16_t image_line[WIDTH][COLORS]; // scratchpad
uint16_t ref_copy[HEIGHT][WIDTH][COLORS]; // in 64MB on-chip memory
```

24KB = 4096*3*2B

48MB = 4096*2048*3*2B

(b) Describe how you use these arrays.

Copy reference image into 64MB on-chip memory at beginning of function and operate on it from there.

Copy each line ($4096 \times 3 \times 2$B) into image_line in the body of F before starting G. All references to image[iy][ix] now go to image_line.

Common Problem: `reference` is accessed randomly. A line buffer will not work for it.

(c) Account for total memory usage in the local scratchpad.

24KB in image_line; 1440B in sintable and costable; 20B in old and current. Less than 26KB

(d) Estimate the new memory resource bound for your optimized `compute_viewpoint`.

$$\frac{4096\times2048\times3\times2}{16\times10^9} + 2^5 \times 2048 \times \frac{4096\times3\times2}{16\times10^9} + \frac{2^5\times4096\times2048\times(10+1)}{10^9}$$

3.1 seconds

```
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t  *old, int16_t  *current)
{
  int64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
     for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
      for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
       for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
       {
         uint64_t score=0;
         for (int iy=0;i<HEIGHT;iy++) // loop F
          for (int ix=0;i<WIDTH;ix++) // loop G
           {
             uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
             uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
             if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
              for (int c=0;c<COLORS;c++) // loop H
               score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
           }
         if (score<best_score)
           {
             best_score=score;
             current[VP_ROT]=rot;
             current[VP_X]=x;
             current[VP_Y]=y;
             current[VP_XS]=xs;
             current[VP_YS]=ys;
           }
       }
  }
}
```

(b) Based only on the resource bound for memory operations, what throughput can a simple, single processor system achieve [answer in frames/second]?

get_image	$\frac{4096\times2048\times3\times2}{16\times10^9}$	0.0031 s
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (2 \times 100)$ cycles image[iy][ix] and reference[ty][tx] is single read ignore small terms sin/costable, current update	54 s

Old memory bound

Question 6

6. Consider rewriting the body of `compute_viewpoint` to minimize the memory resource bound by exploiting the scratchpad memory and the 64 MB on-chip memory and streaming data tranfers.

(a) Identify new temporary arrays allocated to scratchpad memory or 64MB on-chip memory (and specify which memory each new array is in).

```
uint16_t image_line[WIDTH][COLORS]; // scratchpad
uint16_t ref_copy[HEIGHT][WIDTH][COLORS]; // in 64MB on-chip memory
```

(b) Describe how you use these arrays.

Copy reference image into 64MB on-chip memory at beginning of function and operate on it from there.

Copy each line ($4096 \times 3 \times 2B$) into image_line in the body of F before starting G. All references to image[iy][ix] now go to image_line.

Common Problem: `reference` is accessed randomly. A line buffer will not work for it.

(c) Account for total memory usage in the local scratchpad.

24KB in image_line; 1440B in sintable and costable; 20B in old and current. Less than 26KB

(d) Estimate the new memory resource bound for your optimized `compute_viewpoint`.

$$\frac{4096\times2048\times3\times2}{16\times10^9} + 2^5 \times 2048 \times \frac{4096\times3\times2}{16\times10^9} + \frac{2^5\times4096\times2048\times(10+1)}{10^9}$$

3.1 seconds

`reference`, streamed into OCM, 2 for 2B(16bits)

`image_line`, streamed into local scratchpad mem

10 cycles for `reference`(OCM), 1 cycle for `image_line`(scratchpad mem)

```
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t  *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
     for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
      for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
       for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
        {
          uint64_t score=0;
          for (int iy=0;i<HEIGHT;iy++) // loop F
           for (int ix=0;i<WIDTH;ix++) // loop G
            {
              uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
              uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
              if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
               for (int c=0;c<COLORS;c++) // loop H
                score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
            }
          if (score<best_score)
            {
              best_score=score;
              current[VP_ROT]=rot;
              current[VP_X]=x;
              current[VP_Y]=y;
              current[VP_XS]=xs;
              current[VP_YS]=ys;
            }
        }
}
```

`image_line` + previously located in local scratchpad

(b) Based only on the resource bound for memory operations, what throughput can a simple, single processor system achieve [answer in frames/second]?

get_image	$\frac{4096\times2048\times3\times2}{16\times10^9}$	0.0031 s
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (2 \times 100)$ cycles image[iy][ix] and reference[ty][tx] is single read ignore small terms sin/costable, current update	54 s

Old memory bound

Question 7

7. Consider a multiprocessor design that included N copies of a vector processor with 16 vector lanes, each operating on 16b data. This is a single-issue vector processor that can either issue one vector or one scalar operation on each cycle. Assume the loops you identifed as data parallel or reduce operation in Question 3 are perfectly vectorizable. Each vector processor requires 2 mm^2 including a local 64KB data scratchpad.

   (a) Based on computational requirements alone, how many vector processors do you need to achieve a 30 frame per second frame rate? [for this problem, ignore memory and communication]
   82

   (b) Identify how the processors are used.
   Everything that takes significant time is data parallel or an associative reduce.

compute_viewpoint	81
render_augmentation	1
merge_frames	(shared)

```c
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t  *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
      for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
        for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
          for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
            {
             uint64_t score=0;
             for (int iy=0;i<HEIGHT;iy++) // loop F
               for (int ix=0;i<WIDTH;ix++) // loop G
                 {
                  uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
                  uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
                  if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
                    for (int c=0;c<COLORS;c++) // loop H
                      score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
                 }
             if (score<best_score)
               {
                best_score=score;
                current[VP_ROT]=rot;
                current[VP_X]=x;
                current[VP_Y]=y;
                current[VP_XS]=xs;
                current[VP_YS]=ys;
               }
            }
  }
}
```

(a) Based only on the resource bound for compute operations, what throughput can a simple, single processor system achieve [answer in frames/second, or equivalently, augment_frame calls per second]?

get_image	all DMA	0
compute_viewpoint	$2^5 \times 4096 \times 2048 \times (12+3\times 3)$ cycles	5.6 s
render_augmentation	$4096 \times 2048 \times 12$ cycles	0.10 s
merge_frames	$4096 \times 2048 \times (12+3\times 3)$ cycles	0.18 s
send_image	all DMA	0
Total		5.9 s

0.17 frames/second

30 frames/s => 0.03 s per a frame

Originally 5.6s for compute_viewpoint, 0.28s for render_augmentation, merge_frames

If one vector processor is allocated for render_augmentation and merge_frames,
0.28s/16=0.0175s
0.03 - 0.0175 = 0.0125 = 5.6s/(16N)
=> N = 28

Question 8

8. Considering a custom hardware accelerator implementation for `compute_viewpoint` where you are designing both the compute operators and the associated memory architecture. How would you use loop unrolling and array partitioning to achieve guaranteed throughput of 30 frames per second of throughput while minimizing area? Use the following area model in units of mm^2:

- n-bit counters: $n \times 10^{-5}$
- n-bit adder: $n \times 10^{-5}$
- 16×16 multiplier: 2.5×10^{-3}
- p-port, w-bit wide memory holding d words: $w(1+p)(d+6)\times 10^{-7}$

Make the (probably unreasonable) assumption that reads from these memories can be completed in one cycle.

Start by assuming we unroll H; we need to understand how much unrolling of the rest of the loops is required. Since the loops are associative reduce, the inner loop can be pipelined to II=1. $\frac{2^5\times4096\times2048}{A\times10^9} \leq \frac{1}{30}$, giving us A a little over 8. This suggests unrolling about a factor of 16 beyond H will be sufficient.

1GHz, accelerator

Common Problem: Not accounting for the operations that can be pipelined.

Greater than 8, divisor of WIDTH

(a) Unrolling for each loop?

Loop	Unroll Factor
A	1
B	1
C	1
D	1
E	1
F	1
G	16
H	3

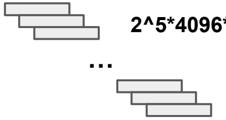
(b) For the unrolling, how many multipliers and adders?

Multipliers	$6\times16=96$
Adders	$16\times(4+3\times2)=160$

(note: for upcoming area calculation, you will need to break down adders by size.)

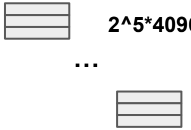
64b: 3×16=48, 16b: $(3+4)\times7=112$

Without unroll



2^5*4096*2048/10^9

With unroll



2^5*4096*2048/(A*10^9)

```
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t  *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
      for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
        for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
          for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
            {
              uint64_t score=0;
              for (int iy=0;i<HEIGHT;iy++) // loop F
                for (int ix=0;i<WIDTH;ix++) // loop G
                  {
                    uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
                    uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
                    if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
                      for (int c=0;c<COLORS;c++) // loop H
                        score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
                  }
              if (score<best_score)
                {
                  best_score=score;
                  current[VP_ROT]=rot;
                  current[VP_X]=x;
                  current[VP_Y]=y;
                  current[VP_XS]=xs;
                  current[VP_YS]=ys;
                }
            }
  }
}
```

Question 8

8. Considering a custom hardware accelerator implementation for `compute_viewpoint` where you are designing both the compute operators and the associated memory architecture. How would you use loop unrolling and array partitioning to achieve guaranteed throughput of 30 frames per second of throughput while minimizing area? Use the following area model in units of mm^2:

- n-bit counters: $n \times 10^{-5}$
- n-bit adder: $n \times 10^{-5}$
- 16×16 multiplier: 2.5×10^{-3}
- p-port, w-bit wide memory holding d words: $w(1+p)(d+6) \times 10^{-7}$

Make the (probably unreasonable) assumption that reads from these memories can be completed in one cycle.

Start by assuming we unroll H; we need to understand how much unrolling of the rest of the loops is required. Since the loops are associative reduce, the inner loop can be pipelined to II=1. $\frac{2^5 \times 4096 \times 2048}{A \times 10^9} \leq \frac{1}{30}$, giving us A a little over 8. This suggests unrolling about a factor of 16 beyond H will be sufficient.

Common Problem: Not accounting for the operations that can be pipelined.

(a) Unrolling for each loop?

Loop	Unroll Factor
A	1
B	1
C	1
D	1
E	1
F	1
G	16
H	3

(b) For the unrolling, how many multipliers and adders?

Multipliers	$6 \times 16 = 96$
Adders	$16 \times (4 + 3 \times 2) = 160$

(note: for upcoming area calculation, you will need to break down adders by size.)

64b: $3 \times 16 = 48$, 16b: $(3+4) \times \cancel{7}$ 16 $= 112$

score is 64bit => 16*3

Typo: Rest is 16 bit => 16*(4+3)

```
void compute_viewpoint(uint16_t ***image, uint16_t ***reference,
                       int16_t  *old, int16_t  *current)
{
  uint64_t best_score=MAXINT; // maximum representable integer

  for (int rot=old[VP_ROT]-ROT;rot<old[VP_ROT]+ROT;rot+=1) { // loop A
    int16_t sr=sintable[rot]; // result is a fraction
    int16_t cr=costable[rot];
    for (int x=old[VP_X]-XOFF;x<old[VP_X]+XOFF;x++) // loop B
     for (int y=old[VP_Y]-YOFF;y<old[VP_Y]+YOFF;y++) // loop C
      for (int xs=old[VP_XS]/XSCALE;xs<old[VP_XS]*XSCALE;xs*=XSFACT) // loop D
       for (int ys=old[VP_YS]/YSCALE;ys<old[VP_YS]*YSCALE;ys*=YSFACT) // loop E
       {
         uint64_t score=0;
         for (int iy=0;i<HEIGHT;iy++) // loop F
          for (int ix=0;i<WIDTH;ix++) // loop G
          {
            uint16_t tx=((ix*cr+iy*sr)*xs)>>(14+8)+x; // 14 to scale sr, cr
            uint16_t ty=((ix*sr+iy*cr)*ys)>>(14+8)+y; // +8 for xscale, yscal
            if ((tx>=0) && (tx<WIDTH) && (ty>=0) && (ty<HEIGHT))
             for (int c=0;c<COLORS;c++) // loop H
              score+=abs(image[iy][ix][c]-reference[ty][tx][c]);
          }
         if (score<best_score)
          {
            best_score=score;
            current[VP_ROT]=rot;
            current[VP_X]=x;
            current[VP_Y]=y;
            current[VP_XS]=xs;
            current[VP_YS]=ys;
          }
       }
  }
}
```

6 multipliers, 4 adders

3*2 adders

(a) Identify new temporary arrays allocated to scratchpad memory or 64MB on-chip memory (and specify which memory each new array is in).

```
uint16_t image_line[WIDTH][COLORS]; // scratchpad
uint16_t ref_copy[HEIGHT][WIDTH][COLORS]; // in 64MB on-chip memory
```

Question 8

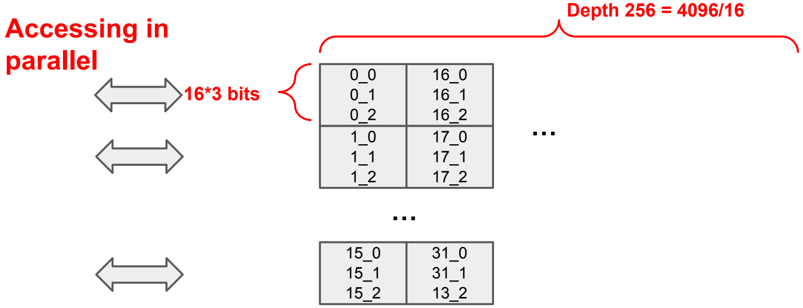
(c) Array partitioning for each array?

Note: blank rows left for local arrays you may have added when optimizing memory in Question 6.

Array	Array Partition	Ports	Width	Depth/partition
old[]	none	1	16	~~10~~ 5
current[]	none	1	16	~~10~~ 5
sintable[]	none	1	16	360
costable[]	none	1	16	360
image[]	n/a			
reference[]	n/a			
image_line[]	cyclic 16 dim 1, x complete dim 2 (and pack), c	1	48	256
ref_tmp[]	none	16	48	8,388,608

2048*4096

Common Problem: `reference` needs ports rather than partitioning since it is accessed randomly.



Partitioned array for `image_line`

Question 8

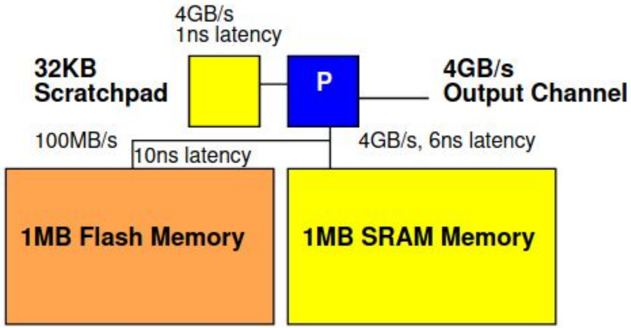
(d) Estimate the area for the accelerator.

Resource	Count	Area/resource	Area
16×16 multipliers	96	2.5×10^{-3}	0.24
64b-adders	48	64×10^{-5}	0.03072
16b-adders	112	16×10^{-5}	0.01792
3b-counters	5	3×10^{-5}	15×10^{-5}
8b-counters	1	8×10^{-5}	8×10^{-5}
12b-counters	1	12×10^{-5}	12×10^{-5}
memory (1,16,10)	2	5.1×10^{-5}	1.0×10^{-4}
memory (1,16,360)	2	1.2×10^{-3}	2.4×10^{-3}
memory (1,48,256)	16	2.5×10^{-3}	0.0402
memory(16,48,8388608)	1	680	680
Total			680

ref_tmp

Fall 2018 Final exam review

System overview




```
// You will be determining a value for FREQBYTES
#define WINDOW 1024
#define MAXBITLEN 11
#define LOG_MAXBITLEN 4
#define MAX_FREQS 255
#define MASKLOOKUP ((1<<MAXBITLEN)-1)
#define MASKLEN ((1<<LOG_MAXBITLEN)-1)
#define AMPLEN 14
#define FREQLEN 14
#define MASKAMP ((1<<AMPLEN)-1)
#define MASKFREQ ((1<<FREQLEN)-1)
uint8_t in[FREQBYTES];
uint32_t fa[FREQS];
uint32_t lookup[1<<MAXBITLEN];
uint16_t s[MAX_FREQS][WINDOW];
while(1) { // Outer while loop
    uint32_t ts[WINDOW];
    for (j=0;j<WINDOW;j++) ts[j]=0; // Loop A
    uint8_t freqs=read_flash_byte(); // max rate 100MB/s
    for(int i=0;i<FREQBYTES;i++) // Loop B
        in[i]=read_flash_byte();
    uint11_t top11=((int *)in)[0]>>21;
    uint11_t next11=(((int *)in)[0]>>10)&MASKLOOKUP;
    int next11bitpos=11;
    for(i=0;i<freqs;i++) { // freqs<MAX_FREQS // Loop C
      uint32_t res=lookup[top11];
      uint32_t tfa=res>>LOG_MAXBITLEN; fa[i]=tfa;
      uint4_t len=MASKLEN & res;
      uint32_t t1=(top11<<len); uint4_t t2=(MAXBITLEN-len); uint32_t t3=(next11>>t2);
      top11= t1|t3;
      next11bitpos+=len;
      uint32_t bytepos=next11bitpos>>3; uint3_t bitoffset=next11bitpos%8;
      uint32_t wordval=(*((int *)(&in[bytepos]))); // treat as 1 cycle
      uint4_t t4=(21-bitoffset); uint32_t t5=(wordval>>t4);
      next11=MASKLOOKUP & t5;
      }
   for (i=0;i<freqs;i++) { // Loop D
       uint16_t freq=(fa[i]>>AMPLEN) & MASKFREQ;
       uint16_t amp=fa[i] & MASKAMP;
       for (j=0;j<WINDOW;j++) // Loop E
           ts[j]+=s[freq][j]*amp;
       }
   for (j=0;j<WINDOW;j++) // Loop F
       output(ts[j]); // max rate 4GB/s
}
```

Question 1

Processor runs in 1GHz, 1ns per cycle

1. For sequential evaluation and assuming FREQBYTES is 256.

   (a) Worst-case cycles to compute one iteration of the outer while loop? (show cycles per loop for partial credit consideration.)

A	WINDOW	1024
B	FREQBYTES×10 100 MB/s bandwidth=10 cycles/byte	2560
between	5	5
C	15 × MAX_FREQS 12 ops, 3 scratchpad memory accesses	3825
D, E	MAX_FREQS × (5 + WINDOW × 10) E 10: 6 for read from s[][] + read and write ts[] + multiply, add	2,612,475
F	WINDOW	1024
Total		2,620,913

   2.6 million cycles

   (b) Which outer loop is the bottleneck?

   Circle One:

A	B	C	(D)	F

   (c) What is the Amdhal's Law maximum speedup for accelerating the identified loop?

$$\frac{A+B+C+D+F}{A+B+C+F} = \frac{2{,}620{,}913}{2{,}620{,}913-2{,}612{,}475} = 310$$

```
// You will be determining a value for FREQBYTES
#define WINDOW 1024
#define MAXBITLEN 11
#define LOG_MAXBITLEN 4
#define MAX_FREQS 255
#define MASKLOOKUP ((1<<MAXBITLEN)-1)
#define MASKLEN ((1<<LOG_MAXBITLEN)-1)
#define AMPLEN 14
#define FREQLEN 14
#define MASKAMP ((1<<AMPLEN)-1)
#define MASKFREQ ((1<<FREQLEN)-1)
uint8_t in[FREQBYTES];
uint32_t fa[FREQS];
uint32_t lookup[1<<MAXBITLEN];
uint16_t s[MAX_FREQS][WINDOW];
while(1) { // Outer while loop
    uint32_t ts[WINDOW];
    for (j=0;j<WINDOW;j++) ts[j]=0; // Loop A
    uint8_t freqs=read_flash_byte(); // max rate 100MB/s
    for(int i=0;i<FREQBYTES;i++) // Loop B
        in[i]=read_flash_byte();
    uint11_t top11=((int *)in)[0]>>21;
    uint11_t next11=(((int *)in)[0]>>10)&MASKLOOKUP;
    int next11bitpos=11;
    for(i=0;i<freqs;i++) { // freqs<MAX_FREQS // Loop C
      uint32_t res=lookup[top11];
      uint32_t tfa=res>>LOG_MAXBITLEN; fa[i]=tfa;
      uint4_t len=MASKLEN & res;
      uint32_t t1=(top11<<len); uint4_t t2=(MAXBITLEN-len); uint32_t t3=(next11>>t2);
      top11= t1|t3;
      next11bitpos+=len;
      uint32_t bytepos=next11bitpos>>3; uint3_t bitoffset=next11bitpos%8;
      uint32_t wordval=(*((int *)(&in[bytepos]))); // treat as 1 cycle
      uint4_t t4=(21-bitoffset); uint32_t t5=(wordval>>t4);
      next11=MASKLOOKUP & t5;
    }
    for (i=0;i<freqs;i++) { // Loop D
       uint16_t freq=(fa[i]>>AMPLEN) & MASKFREQ;
       uint16_t amp=fa[i] & MASKAMP;
       for (j=0;j<WINDOW;j++) // Loop E
           ts[j]+=s[freq][j]*amp;
       }
    for (j=0;j<WINDOW;j++) // Loop F
       output(ts[j]); // max rate 4GB/s
}
```

Assume that this can be hidden..

Question 2

2. Loop C

(a) How many memory operations does one instance of the loop perform?

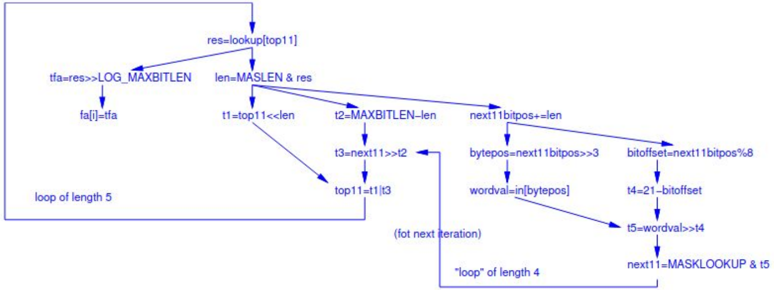
3 – lookup[], fa[], in[]

(b) How many compute operations (of the set identified) does the loop perform?

12

(c) Assuming unlimited compute operators and memory ports, what is the minimum achievable Initiation Interval (II) for this loop?
Draw dataflow graph and identify any data-dependent loops for full credit.

II=5



Note: Critical path is 7. The key loop is the one around top11, which is of length 5. We must also be able to update next 11, and that is in a loop of length 4. Strictly, it's not a loop itself, but we do need to be able to compute the next11 within one II, and this does fit.

```
// You will be determining a value for FREQBYTES
#define WINDOW 1024
#define MAXBITLEN 11
#define LOG_MAXBITLEN 4
#define MAX_FREQS 255
#define MASKLOOKUP ((1<<MAXBITLEN)-1)
#define MASKLEN ((1<<LOG_MAXBITLEN)-1)
#define AMPLEN 14
#define FREQLEN 14
#define MASKAMP ((1<<AMPLEN)-1)
#define MASKFREQ ((1<<FREQLEN)-1)
uint8_t in[FREQBYTES];
uint32_t fa[FREQS];
uint32_t lookup[1<<MAXBITLEN];
uint16_t s[MAX_FREQS][WINDOW];
while(1) { // Outer while loop
    uint32_t ts[WINDOW];
    for (j=0;j<WINDOW;j++) ts[j]=0; // Loop A
    uint8_t freqs=read_flash_byte(); // max rate 100MB/s
    for(int i=0;i<FREQBYTES;i++) // Loop B
        in[i]=read_flash_byte();
    uint11_t top11=((int *)in)[0]>>21;
    uint11_t next11=(((int *)in)[0]>>10)&MASKLOOKUP;
    int next11bitpos=11;
    for(i=0;i<freqs;i++) { // freqs<MAX_FREQS // Loop C
      uint32_t res=lookup[top11];
      uint32_t tfa=res>>LOG_MAXBITLEN; fa[i]=tfa;
      uint4_t len=MASKLEN & res;
      uint32_t t1=(top11<<len); uint4_t t2=(MAXBITLEN-len); uint32_t t3=(next11>>t2);
      top11= t1|t3;
      next11bitpos+=len;
      uint32_t bytepos=next11bitpos>>3; uint3_t bitoffset=next11bitpos%8;
      uint32_t wordval=(*((int *)(&in[bytepos]))); // treat as 1 cycle
      uint4_t t4=(21-bitoffset); uint32_t t5=(wordval>>t4);
      next11=MASKLOOKUP & t5;
      }
  for (i=0;i<freqs;i++) { // Loop D
     uint16_t freq=(fa[i]>>AMPLEN) & MASKFREQ;
     uint16_t amp=fa[i] & MASKAMP;
     for (j=0;j<WINDOW;j++) // Loop E
         ts[j]+=s[freq][j]*amp;
     }
  for (j=0;j<WINDOW;j++) // Loop F
     output(ts[j]); // max rate 4GB/s
}
```

Question 3

3. Data Parallel: Classify Loops C, D, and E:

Loop	Data Parallel?	Associative Reduce?	Must be Sequential?
C			Yes
D		Yes	
E	Yes		

C: The dependent loop for top11 identified in Problem2c forces sequentialization of the loop.

E: operations are independent for each j. Can perform the entire multiplication and add concurrently. This vectorizable.

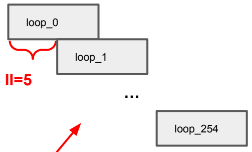
D: If you think about unrolling E into a vector, then unrolling D as well, the only dependency is the add chain for each `freq` into ts[j]. The add is associative, so this is an associative reduce operation.

Associative add for Loop D

freq_0 = (fa[0]>>	**freq _1 = (fa[1]>>**	**freq_2 = (fa[2]>>**
amp_0 = fa[0] & MASKAMP	**Amp_1 = fa[1] & MASKAMP**	**amp_2 = fa[2] & MASKAMP**
ts[0] = ts[0] + s[freq_0][0] * amp_0	**ts[0] = ts[0] + s[freq_1][0] * amp_1**	**ts[0] = ts[0] + s[freq_2][0] * amp_2**
ts[1] = ts[1] + s[freq_0][1] * amp_0	**ts[1] = ts[1] + s[freq_1][1] * amp_1**	**ts[1] = ts[1] + s[freq_2][1] * amp_2**
ts[2] = ts[2] + s[freq_0][2] * amp_0	**ts[2] = ts[2] + s[freq_1][2] * amp_1**	**ts[2] = ts[2] + s[freq_2][2] * amp_2**
ts[3] = ts[3] + s[freq_0][3]* amp_0	**ts[3] = ts[3] + s[freq_1][3]* amp_1**	**ts[3] = ts[3] + s[freq_2][3]* amp_2**
…	**…**	**…**

```
// You will be determining a value for FREQBYTES
#define WINDOW 1024
#define MAXBITLEN 11
#define LOG_MAXBITLEN 4
#define MAX_FREQS 255
#define MASKLOOKUP ((1<<MAXBITLEN)-1)
#define MASKLEN ((1<<LOG_MAXBITLEN)-1)
#define AMPLEN 14
#define FREQLEN 14
#define MASKAMP ((1<<AMPLEN)-1)
#define MASKFREQ ((1<<FREQLEN)-1)
uint8_t in[FREQBYTES];
uint32_t fa[FREQS];
uint32_t lookup[1<<MAXBITLEN];
uint16_t s[MAX_FREQS][WINDOW];
while(1) { // Outer while loop
    uint32_t ts[WINDOW];
    for (j=0;j<WINDOW;j++) ts[j]=0; // Loop A
    uint8_t freqs=read_flash_byte(); // max rate 100MB/s
    for(int i=0;i<FREQBYTES;i++) // Loop B
        in[i]=read_flash_byte();
    uint11_t top11=((int *)in)[0]>>21;
    uint11_t next11=(((int *)in)[0]>>10)&MASKLOOKUP;
    int next11bitpos=11;
    for(i=0;i<freqs;i++) { // freqs<MAX_FREQS // Loop C
      uint32_t res=lookup[top11];
      uint32_t tfa=res>>LOG_MAXBITLEN; fa[i]=tfa;
      uint4_t len=MASKLEN & res;
      uint32_t t1=(top11<<len); uint4_t t2=(MAXBITLEN-len); uint32_t t3=(next11>>t2);
      top11= t1|t3;
      next11bitpos+=len;
      uint32_t bytepos=next11bitpos>>3; uint3_t bitoffset=next11bitpos%8;
      uint32_t wordval=(*((int *)(&in[bytepos]))); // treat as 1 cycle
      uint4_t t4=(21-bitoffset); uint32_t t5=(wordval>>t4);
      next11=MASKLOOKUP & t5;
    }
   for (i=0;i<freqs;i++) { // Loop D
      uint16_t freq=(fa[i]>>AMPLEN) & MASKFREQ;
      uint16_t amp=fa[i] & MASKAMP;
      for (j=0;j<WINDOW;j++) // Loop E
          ts[j]+=s[freq][j]*amp;
    }
   for (j=0;j<WINDOW;j++) // Loop F
      output(ts[j]); // max rate 4GB/s
}
```

Question 4



=> 5*255

4. What is the latency bound for executing Loops C and D (from the beginning of C to the end of D)?

- assume memories of unbounded width (no bandwidth limits)
- respect latencies for memory access

Loop C: From Problems 2 and 3, we know this loop is sequentially dependent with an II of 5. So, it will take: MAX_FREQS × II=255×5=1275 cycles.

Loop E: This is data parallel. Fully unrolled this takes 6 (read s[][])+1=7 cycles to get to the products.

Loop D: This is a reduce add across MAX_FREQS values to produce each ts[j]. That can be done in $\log_2(MAX\_FREQS) =$ 8 cycles. There's a final write into ts[j] at the end.

Together, this gives us 1275+7+8+1=1291 cycles or 1.3μs.

We can do slightly better observing that we can overlap some (or most) of the additions in D-E with C. So, even if we sequentially perform the E vector adds, we can complete one per cycle and match pace with C. So, after finishing C, we only need to perform the 7 cycles for the E lookup and multiply, then a final add and store So, we can perform this is 1275+7+1+1=1284. To two significant figures, this is also 1.3μs.

Assume that freq, amp are hidden

```
// You will be determining a value for FREQBYTES
#define WINDOW 1024
#define MAXBITLEN 11
#define LOG_MAXBITLEN 4
#define MAX_FREQS 255
#define MASKLOOKUP ((1<<MAXBITLEN)-1)
#define MASKLEN ((1<<LOG_MAXBITLEN)-1)
#define AMPLEN 14
#define FREQLEN 14
#define MASKAMP ((1<<AMPLEN)-1)
#define MASKFREQ ((1<<FREQLEN)-1)
uint8_t in[FREQBYTES];
uint32_t fa[FREQS];
uint32_t lookup[1<<MAXBITLEN];
uint16_t s[MAX_FREQS][WINDOW];
while(1) { // Outer while loop
    uint32_t ts[WINDOW];
    for (j=0;j<WINDOW;j++) ts[j]=0; // Loop A
    uint8_t freqs=read_flash_byte(); // max rate 100MB/s
    for(int i=0;i<FREQBYTES;i++) // Loop B
        in[i]=read_flash_byte();
    uint11_t top11=((int *)in)[0]>>21;
    uint11_t next11=(((int *)in)[0]>>10)&MASKLOOKUP;
    int next11bitpos=11;
    for(i=0;i<freqs;i++) { // freqs<MAX_FREQS // Loop C
      uint32_t res=lookup[top11];
      uint32_t tfa=res>>LOG_MAXBITLEN; fa[i]=tfa;
      uint4_t len=MASKLEN & res;
      uint32_t t1=(top11<<len); uint4_t t2=(MAXBITLEN-len); uint32_t t3=(next11>>t2);
      top11= t1|t3;
      next11bitpos+=len;
      uint32_t bytepos=next11bitpos>>3; uint3_t bitoffset=next11bitpos%8;
      uint32_t wordval=(*((int *)(&in[bytepos]))); // treat as 1 cycle
      uint4_t t4=(21-bitoffset); uint32_t t5=(wordval>>t4);
      next11=MASKLOOKUP & t5;
      }
   for (i=0;i<freqs;i++) { // Loop D
      uint16_t freq=(fa[i]>>AMPLEN) & MASKFREQ;
      uint16_t amp=fa[i] & MASKAMP;
      for (j=0;j<WINDOW;j++) // Loop E
          ts[j]+=s[freq][j]*amp;
      }
   for (j=0;j<WINDOW;j++) // Loop F
      output(ts[j]); // max rate 4GB/s
}
```

Question 5

5. Data Streaming: How big (minimum size) does the buffer need to be between the identified loops in order to allow the loops to profitably execute concurrently. (Hint: Based on data dependencies, under what scenarios and granularity can the identified loops act as a producer-consumer pair in a pipeline.)

Explain size choices for partial credit consideration.

Loop Pair	Size (bytes)
B→C	1 or 4
C→D	4
D→F	4096

B→C: Each byte read can be passed directly to C, and C can perform a lookup. Technically, C may read a whole 32b word. However, depending on length, it may consume less than a byte on each iteration. If C is consuming less than a byte, it can use each byte as it shows up. If C is consuming a whole 32b word, then it will need to get a full word (4 bytes) to be able to perform each operation.

C→D: As each fa[i] is produced, C can pass it to D, allowing D to perform one loop body on that fa[i]. fa is produced by C and consumed by D in order.

D→F: ts[] is updated on every invocation of D. The final value of ts[] is not known until the D completes the final iteration. As such, D cannot pass ts[] to F until it completes its execution. Then the whole ts[] (WINDOW×4=4096 bytes) can be given to F. F can write ts[] out while D is operating on the next iteration of the outer while loop.

So, the whole B→C→D body can operate as a pipeline. B and C can operate on data in the same outer while iteration, passing data in bytes or words as they are produced, while F must operate on data from an earlier outer while iteration than B and C.

```
// You will be determining a value for FREQBYTES
#define WINDOW 1024
#define MAXBITLEN 11
#define LOG_MAXBITLEN 4
#define MAX_FREQS 255
#define MASKLOOKUP ((1<<MAXBITLEN)-1)
#define MASKLEN ((1<<LOG_MAXBITLEN)-1)
#define AMPLEN 14
#define FREQLEN 14
#define MASKAMP ((1<<AMPLEN)-1)
#define MASKFREQ ((1<<FREQLEN)-1)
uint8_t in[FREQBYTES];
uint32_t fa[FREQS];
uint32_t lookup[1<<MAXBITLEN];
uint16_t s[MAX_FREQS][WINDOW];
while(1) { // Outer while loop
    uint32_t ts[WINDOW];
    for (j=0;j<WINDOW;j++) ts[j]=0; // Loop A
    uint8_t freqs=read_flash_byte(); // max rate 100MB/s
    for(int i=0;i<FREQBYTES;i++) // Loop B
        in[i]=read_flash_byte();
    uint11_t top11=((int *)in)[0]>>21;
    uint11_t next11=(((int *)in)[0]>>10)&MASKLOOKUP;
    int next11bitpos=11;
    for(i=0;i<freqs;i++) { // freqs<MAX_FREQS // Loop C
      uint32_t res=lookup[top11];
      uint32_t tfa=res>>LOG_MAXBITLEN; fa[i]=tfa;
      uint4_t len=MASKLEN & res;
      uint32_t t1=(top11<<len); uint4_t t2=(MAXBITLEN-len); uint32_t t3=(next11>>t2);
      top11= t1|t3;
      next11bitpos+=len;
      uint32_t bytepos=next11bitpos>>3; uint3_t bitoffset=next11bitpos%8;
      uint32_t wordval=(*((int *)(&in[bytepos]))); // treat as 1 cycle
      uint4_t t4=(21-bitoffset); uint32_t t5=(wordval>>t4);
      next11=MASKLOOKUP & t5;
    }
   for (i=0;i<freqs;i++) { // Loop D
     uint16_t freq=(fa[i]>>AMPLEN) & MASKFREQ;
     uint16_t amp=fa[i] & MASKAMP;
     for (j=0;j<WINDOW;j++) // Loop E
         ts[j]+=s[freq][j]*amp;
    }
   for (j=0;j<WINDOW;j++) // Loop F
     output(ts[j]); // max rate 4GB/s
}
```

Question 6

6. Consider trying to achieve a real-time rate of one window output per cycle (equivalently, the II of the outer while loop is WINDOW or 1024 cycles).
Assume you exploit data streaming between loops so they can run concurrently.

(a) Given that Flash memory has a maximum throughput of 100 MB/s, what is the maximum possible value for FREQBYTES?
100MB/s throughput, means the fastest we can read each byte is once ever 10 cycles.
FREQBYTES $\times$ 10 = 1024 $\rightarrow$ FREQBYTES=102.

Within 1024 cycles, 102 loops are possible for Loop B

(b) Based on your II identified in Problem 2c, what is the maximum value for `freqs` in order fo meet this real-time throughput goal?
freqs $\times$ II = 1024 $\rightarrow$ freqs $\times$ 5 = 1024 $\rightarrow$ freqs=204.

Within 1024 cycles, 204 loops are possible for Loop C

(c) What II do you need to achieve for Loop D to meet this real-time throughput goal?
The most direct argument is that this needs to match the rate of Loop C, so also has an II=5 requirement.
Alternately, we have the same equation, now with II_D as the variable.
freqs $\times II_D$=1024 $\rightarrow$ 204$\times II_D$=1024 $\rightarrow$ $II_D = 5$.

```
// You will be determining a value for FREQBYTES
#define WINDOW 1024
#define MAXBITLEN 11
#define LOG_MAXBITLEN 4
#define MAX_FREQS 255
#define MASKLOOKUP ((1<<MAXBITLEN)-1)
#define MASKLEN ((1<<LOG_MAXBITLEN)-1)
#define AMPLEN 14
#define FREQLEN 14
#define MASKAMP ((1<<AMPLEN)-1)
#define MASKFREQ ((1<<FREQLEN)-1)
uint8_t in[FREQBYTES];
uint32_t fa[FREQS];
uint32_t lookup[1<<MAXBITLEN];
uint16_t s[MAX_FREQS][WINDOW];
while(1) { // Outer while loop
    uint32_t ts[WINDOW];
    for (j=0;j<WINDOW;j++) ts[j]=0; // Loop A
    uint8_t freqs=read_flash_byte(); // max rate 100MB/s
    for(int i=0;i<FREQBYTES;i++) // Loop B
        in[i]=read_flash_byte();
    uint11_t top11=((int *)in)[0]>>21;
    uint11_t next11=(((int *)in)[0]>>10)&MASKLOOKUP;
    int next11bitpos=11;
    for(i=0;i<freqs;i++) { // freqs<MAX_FREQS // Loop C
      uint32_t res=lookup[top11];
      uint32_t tfa=res>>LOG_MAXBITLEN; fa[i]=tfa;
      uint4_t len=MASKLEN & res;
      uint32_t t1=(top11<<len); uint4_t t2=(MAXBITLEN-len); uint32_t t3=(next11>>t2);
      top11= t1|t3;
      next11bitpos+=len;
      uint32_t bytepos=next11bitpos>>3; uint3_t bitoffset=next11bitpos%8;
      uint32_t wordval=(*((int *)(&in[bytepos]))); // treat as 1 cycle
      uint4_t t4=(21-bitoffset); uint32_t t5=(wordval>>t4);
      next11=MASKLOOKUP & t5;
    }
    for (i=0;i<freqs;i++) { // Loop D
      uint16_t freq=(fa[i]>>AMPLEN) & MASKFREQ;
      uint16_t amp=fa[i] & MASKAMP;
      for (j=0;j<WINDOW;j++) // Loop E
          ts[j]+=s[freq][j]*amp;
    }
    for (j=0;j<WINDOW;j++) // Loop F
      output(ts[j]); // max rate 4GB/s
}
```

Question 7

7. Define the composition of a custom VLIW datapath for loop C that can achieve the identified II in Problem 2c.
   For full credit, minimize area of your implementation.
   Assume:

   - Design includes at least one write port to a scratchpad memory containing fa[] and one read port to a scratchpad memory containing in[]
   - Assume a crossbar interconnect between operator (and memory port) outputs and operator (and memory address, data) inputs.

   (a) How many operators of each type? Give both Resource Bound (RB) and number for which you can schedule.

Operator	Inputs	Outputs	Number RB	Number Schedule
shifters	2	1	$\left\lceil \frac{5}{5} \right\rceil = 1$	2
ALU (includes |, &, +, - , %-by-powers-of-2)	2	1	$\left\lceil \frac{7}{5} \right\rceil = 2$	2
scratchpad memory banks	2	1	1	1
ports to memory containing in[]	1	1	1	1
ports to memory containing fa[]	1	0	1	1
above error, should be	2	0		
branch units	1	0	1	1

   (b) How are the scratchpad memory banks used?

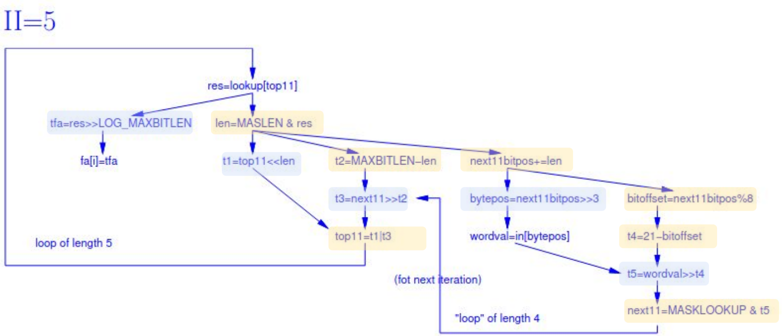
   Hold lookup[] array.

   (c) Crossbar Inputs and Outputs for your design (final column, the one you can schedule)?

Inputs	13 (or 14 with correction)
Outputs	6

14= 2*2 + 2*2 + 2*1 + 1*1 + 2*1 + 1*1

At least, how many operators do we need to achieve II=5?



For write, ports have to be 2, addr, data.
For read, port has to be 1, addr.

Question 7

7. Define the composition of a custom VLIW datapath for loop C that can achieve the identified II in Problem 2c.
For full credit, minimize area of your implementation.
Assume:

- Design includes at least one write port to a scratchpad memory containing fa[] and one read port to a scratchpad memory containing in[]
- Assume a crossbar interconnect between operator (and memory port) outputs and operator (and memory address, data) inputs.

(a) How many operators of each type? Give both Resource Bound (RB) and number for which you can schedule.

Operator	Inputs	Outputs	Number RB	Number Schedule
shifters	2	1	$\left\lceil \frac{5}{5} \right\rceil = 1$	2
ALU (includes |, &, +, - , %-by-powers-of-2)	2	1	$\left\lceil \frac{7}{5} \right\rceil = 2$	2
scratchpad memory banks	2	1	1	1
ports to memory containing in[]	1	1	1	1
ports to memory containing fa[]	1	0	1	1
above error, should be	2	0		
branch units	1	0	1	1

(b) How are the scratchpad memory banks used?
Hold lookup[] array.

(c) Crossbar Inputs and Outputs for your design (final column, the one you can schedule)?

Inputs	13 (or 14 with correction)
Outputs	6

(d) Estimate the area for your design using the following costs.

- shifters: 1024
- ALU (includes |, &, +, - , %-by-powers-of-2): 32
- Scratchpad memory banks of depth d: $60(d+6)$
- ports to memory containing in[]: 200
- ports to memory containing fa[]: 200
- branch unit: 100
- crossbar: $128 \times \mathit{Inputs} \times \mathit{Outputs} + 2400 \times \mathit{Outputs}$
(Each crossbar output includes a 4 word memory acting as a small register file for input to the associated operator or memory.)

$$2 \times 1024 + 2 \times 32 + 60(2048+6) + 200 + 200 + 100 + 128 \times 13 \times 6 + 2400 \times 6 = 150,236 \approx 150,000$$

Question 7

(e) Provide a schedule:

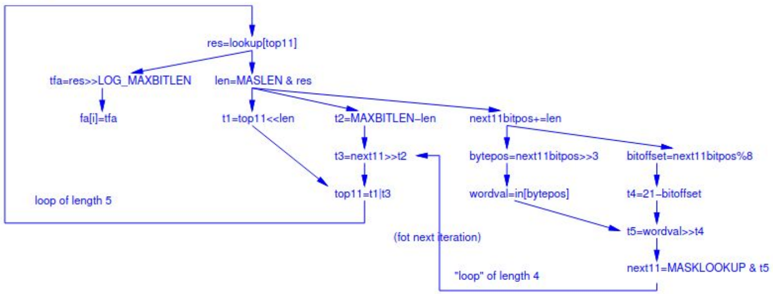
			Label with your selected operators					
Operator→ Cycle	fa[] write	in[] read	lookup[]	shift0	shift1	ALU0	ALU1	Branch
0			res	t5				
1				tfa		len	next11	
2	fa[i]			t1		t2	next11bitpos	
3				t3	bytepos	bitoffset		
4		wordval				top11	t4	branch i<freqs
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								

Label cells with the variable assigned by the operation (or array entry written).

orange instructions software pipelined from previous iteration of loop

(Note extra schedules at end. May want to use as scratch while exploring schedules and put final here.)

II=5



Question 8

8. Considering a custom hardware accelerator implementation where you are designing both the compute operators and the associated memory architecture, how would you use loop unrolling and array partitioning on Loop D to achieve the identified II in Problem 6c, while minimizing area?
Use the following area model and assume s[], ts[], and fa[] are part of this loop module:

- n-bit counters: n
- 32b adder: 32
- 16×16 multiplier: 256
- Single-port, 32b-wide memory holding d words: $38(d+6)$
- Double-port, 32b-wide memory holding d words: $60(d+6)$

(a) Unrolling for each loop (D, E)?

Loop	Unroll Factor
D	1
E	205

To meet the $II = 5$ goal, we must perform $\lceil \frac{1024}{5} \rceil = 205$ loop bodies of E on each cycle. So, we can unroll E 205 times and pipeline the computation.

(b) For the unrolling, how many multipliers and adders?

Multipliers	205
Adders	205

Note: Since E is inside D, unrolling D D_{unroll} times and E E_{unroll} times, will result in $D_{unroll} \times E_{unroll}$ adders and multipliers.

(c) Array partitioning for each array (s[], ts[], fa[])?
(each memory block has either 1 or 2 ports)

Array	Array Partition	Ports (select one)		Words/partition
s[][]	cyclic 205 dimension 1	(1)	2	1280
ts[]	cyclic 205	1	(2)	5
fa[]	1	1	(2)	256

MAX_FREQS*1024/205

Note that s[] is only read. ts[] must be both read and written on each iteration. fa[] must be written by C and read by D.

```
// You will be determining a value for FREQBYTES
#define WINDOW 1024
#define MAXBITLEN 11
#define LOG_MAXBITLEN 4
#define MAX_FREQS 255
#define MASKLOOKUP ((1<<MAXBITLEN)-1)
#define MASKLEN ((1<<LOG_MAXBITLEN)-1)
#define AMPLEN 14
#define FREQLEN 14
#define MASKAMP ((1<<AMPLEN)-1)
#define MASKFREQ ((1<<FREQLEN)-1)
uint8_t in[FREQBYTES];
uint32_t fa[FREQS];
uint32_t lookup[1<<MAXBITLEN];
uint16_t s[MAX_FREQS][WINDOW];
while(1) { // Outer while loop
    uint32_t ts[WINDOW];
    for (j=0;j<WINDOW;j++) ts[j]=0; // Loop A
    uint8_t freqs=read_flash_byte(); // max rate 100MB/s
    for(int i=0;i<FREQBYTES;i++) // Loop B
        in[i]=read_flash_byte();
    uint11_t top11=((int *)in)[0]>>21;
    uint11_t next11=(((int *)in)[0]>>10)&MASKLOOKUP;
    int next11bitpos=11;
    for(i=0;i<freqs;i++) { // freqs<MAX_FREQS // Loop C
      uint32_t res=lookup[top11];
      uint32_t tfa=res>>LOG_MAXBITLEN; fa[i]=tfa;
      uint4_t len=MASKLEN & res;
      uint32_t t1=(top11<<len); uint4_t t2=(MAXBITLEN-len); uint32_t t3=(next11>>t2);
      top11= t1|t3;
      next11bitpos+=len;
      uint32_t bytepos=next11bitpos>>3; uint3_t bitoffset=next11bitpos%8;
      uint32_t wordval=(*((int *)(&in[bytepos]))); // treat as 1 cycle
      uint4_t t4=(21-bitoffset); uint32_t t5=(wordval>>t4);
      next11=MASKLOOKUP & t5;
      }
   for (i=0;i<freqs;i++) { // Loop D
      uint16_t freq=(fa[i]>>AMPLEN) & MASKFREQ;
      uint16_t amp=fa[i] & MASKAMP;
      for (j=0;j<WINDOW;j++) // Loop E
          ts[j]+=s[freq][j]*amp;
      }
   for (j=0;j<WINDOW;j++) // Loop F
      output(ts[j]); // max rate 4GB/s
}
```

Question 8

Properly pipelined fa[] could get away with one port; when C and D are run concurrently fa[] could go away as a memory and just become a register between C and D.

(d) Identify the component(s) that consumes most (>80%) of the area? (you don't necessarily need to compute the area to fine precision, but you need to estimate where area is going well enough to answer the question above.)

Component	Calculate	Area
8-bit counter for E	8	11
3-bit counter for D	3	
Adder	32×205	6560
Multiplier	256×205	52480
s[][]	$205 \times 38(1280 + 6)$	10017940
ts[]	$205 \times 60(5 + 6)$	135300
fa[]	$60(256 + 6)$	15720
total		10228011

98% of area is the single-ported memory for s[][].

Memory (for s[][]) consumes >80% of the area.

```c
// You will be determining a value for FREQBYTES
#define WINDOW 1024
#define MAXBITLEN 11
#define LOG_MAXBITLEN 4
#define MAX_FREQS 255
#define MASKLOOKUP ((1<<MAXBITLEN)-1)
#define MASKLEN ((1<<LOG_MAXBITLEN)-1)
#define AMPLEN 14
#define FREQLEN 14
#define MASKAMP ((1<<AMPLEN)-1)
#define MASKFREQ ((1<<FREQLEN)-1)
uint8_t in[FREQBYTES];
uint32_t fa[FREQS];
uint32_t lookup[1<<MAXBITLEN];
uint16_t s[MAX_FREQS][WINDOW];
while(1) { // Outer while loop
    uint32_t ts[WINDOW];
    for (j=0;j<WINDOW;j++) ts[j]=0; // Loop A
    uint8_t freqs=read_flash_byte(); // max rate 100MB/s
    for(int i=0;i<FREQBYTES;i++) // Loop B
        in[i]=read_flash_byte();
    uint11_t top11=((int *)in)[0]>>21;
    uint11_t next11=(((int *)in)[0]>>10)&MASKLOOKUP;
    int next11bitpos=11;
    for(i=0;i<freqs;i++) { // freqs<MAX_FREQS // Loop C
      uint32_t res=lookup[top11];
      uint32_t tfa=res>>LOG_MAXBITLEN; fa[i]=tfa;
      uint4_t len=MASKLEN & res;
      uint32_t t1=(top11<<len); uint4_t t2=(MAXBITLEN-len); uint32_t t3=(next11>>t2);
      top11= t1|t3;
      next11bitpos+=len;
      uint32_t bytepos=next11bitpos>>3; uint3_t bitoffset=next11bitpos%8;
      uint32_t wordval=(*((int *)(&in[bytepos]))); // treat as 1 cycle
      uint4_t t4=(21-bitoffset); uint32_t t5=(wordval>>t4);
      next11=MASKLOOKUP & t5;
      }
   for (i=0;i<freqs;i++) { // Loop D
       uint16_t freq=(fa[i]>>AMPLEN) & MASKFREQ;
       uint16_t amp=fa[i] & MASKAMP;
       for (j=0;j<WINDOW;j++) // Loop E
           ts[j]+=s[freq][j]*amp;
       }
   for (j=0;j<WINDOW;j++) // Loop F
       output(ts[j]); // max rate 4GB/s
}
```