EPFL

Lab. On HW-SW Digital Systems Codesign
EE-390(a)

Session 8
Dynamic job scheduling across multiple accelerators

Prof. David Atienza
Dr. Denisa Constantinescu, Dr. Miguel Peón-Quirós
Mr. Rubén Rodríguez-Álvarez, Ms. Stasa Kostic, Mr. Karan Pathak

Session objectives

- Problem definition: genome sequence alignment
- SW implementation with OpenMP
- Basic HW implementation
- HW implementation with caching of sequences
- Improving performance with replication of the HW module
- Improving performance with configurable numbers of workers

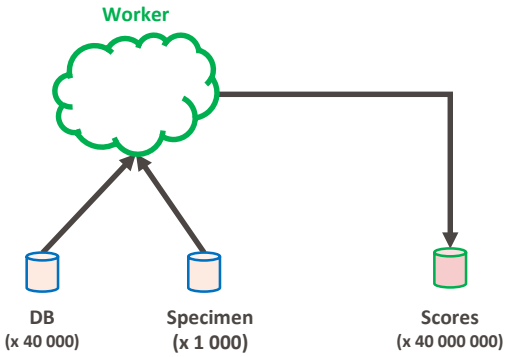
Problem definition

—Genomic sequence alignment

Genomic sequence alignment

- *In bioinformatics, a sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences.*[1]

- In essence, we have two sets of strings and we want to compare them, finding out how many edits are required to transform one sequence into the other
- The problem can be solved using dynamic programming, i.e., creating a table
- If we have *n* entries in the database file, and *m* entries in the file for the specimen under study, we want to match *n x m* strings



[1] Wikipedia. Sequence alignment. https://en.wikipedia.org/wiki/Sequence_alignment

SW Implementation

—Using OpenMP

SW implementation with OpenMP

- The SW implementation reads all the lines of the DB and the specimen in memory
  - Building auxiliary vectors with the length of each string
- Then, each sequence from the DB is compared against all the specimen sequences
- OpenMP is a C-library that allows the programmer to easily use multiple threads
  - It is based in compiler pragmas
  - Internally, it creates and destroys threads as necessary
  - In this case, we need to compute the split point in the vectors

```
119 ////////////////////////////////////////////////////////////////////////////////
120 uint32_t SeqMatcher(uint32_t numDBEntries, uint32_t numSeqsSpecimen,
121     char * seqsDB, char * seqsSpecimen, uint8_t * lengthsDB, uint8_t * lengthsSpecimen,
122     int8_t * scores, uint64_t & elapsedTime, double & cpuUtilization)
123 {
124   struct timespec start, end;
125   struct timespec startCPUTime, endCPUTime;
126   uint32_t offsetsDB[NUM_THREADS];
127   uint32_t comparisons = 0;
128
129   clock_gettime(CLOCK_PROCESS_CPUTIME_ID, & startCPUTime);
130   clock_gettime(CLOCK_MONOTONIC_RAW, &start);
131
132   // Compute the starting point for each thread.
133   uint32_t offset = 0, threadIndex = 0, tmp = 0, seqsPerThread = numDBEntries / NUM_THREADS;
134   uint8_t * pLengthsDB1 = lengthsDB;
135   for (uint32_t ii = 0; ii < numDBEntries; ++ ii) {
136     if (tmp == seqsPerThread)
137       tmp = 0;
138     if (tmp == 0)
139       offsetsDB[threadIndex++] = offset;
140     offset += *pLengthsDB1++;
141     ++ tmp;
142   }
143
```

```
144 #pragma omp parallel reduction(+: comparisons) num_threads(NUM_THREADS)
145   {
146     int threadID = omp_get_thread_num();
147     char * pDB, * pSpec;
148     int8_t * pScores;
149     uint8_t * pLengthsDB, * pLengthsSpec;
150
151     pLengthsDB = &lengthsDB[threadID*seqsPerThread];
152     pDB = seqsDB + (offsetsDB[threadID])*sizeof(char);
153     pScores = &scores[threadID*seqsPerThread*numSeqsSpecimen];
154
155     for (uint32_t iDB = 0; iDB < seqsPerThread; ++iDB) {
156       #ifdef  PRINT_PROGRESS
157       if ( (iDB % 100) == 0 ) { // Beware of this affecting the time measurement!!!!!!
158         printf("%1.0f%% of DB entries processed...\r", ((float)iDB/seqsPerThread)*100 );
159         fflush(stdout);
160       }
161       #endif
162       pLengthsSpec = lengthsSpecimen;
163       pSpec = seqsSpecimen;
164       // Compare current DB entry with all specimen entries.
165       for (uint32_t iSpec = 0; iSpec < numSeqsSpecimen; ++ iSpec) {
166         *pScores++ = CalcScore(pDB, *pLengthsDB, pSpec, *pLengthsSpec);
167         ++ comparisons;
168         pSpec += *pLengthsSpec++;
169       }
170       pDB += *pLengthsDB++; // Pass to the next sequence in the DB
171     }
172   }
173
174   clock_gettime(CLOCK_MONOTONIC_RAW, &end);
175   clock_gettime(CLOCK_PROCESS_CPUTIME_ID, & endCPUTime);
176   elapsedTime = CalcTimeDiff(end, start);
177   cpuUtilization = (double)CalcTimeDiff(endCPUTime, startCPUTime) / elapsedTime;
178
179   return comparisons;
180 }
181
```

SW implementation with OpenMP

- The SW implementation reads all the lines of the DB and the specimen in memory
  - Building auxiliary vectors with the length of each string
- Then, each sequence from the DB is compared against all the specimen sequences
- OpenMP is a C-library that allows the programmer to easily use multiple threads
  - It is based in compiler pragmas
  - Internally, it creates and destroys threads as necessary
  - In this case, we need to compute the split point in the vectors

```
119 ////////////////////////////////////////////////////////////////////////////////
120 uint32_t SeqMatcher(uint32_t numDBEntries, uint32_t numSeqsSpecimen,
121     char * seqsDB, char * seqsSpecimen, uint8_t * lengthsDB, uint8_t * lengthsSpecimen,
122     int8_t * scores, uint64_t & elapsedTime, double & cpuUtilization)
123 {
124   struct timespec start, end;
125   struct timespec startCPUTime, endCPUTime;
126   uint32_t offsetsDB[NUM_THREADS];
127   uint32_t comparisons = 0;
128
129   clock_gettime(CLOCK_PROCESS_CPUTIME_ID, & startCPUTime);
130   clock_gettime(CLOCK_MONOTONIC_RAW, &start);
131
132   // Compute the starting point for each thread.
133   uint32_t offset = 0, threadIndex = 0, tmp = 0, seqsPerThread = numDBEntries / NUM_THREADS;
134   uint8_t * pLengthsDB1 = lengthsDB;
135   for (uint32_t ii = 0; ii < numDBEntries; ++ ii) {
136     if (tmp == seqsPerThread)
137       tmp = 0;
138     if (tmp == 0)
139       offsetsDB[threadIndex++] = offset;
140     offset += *pLengthsDB1++;
141     ++ tmp;
142   }
143
```

```
144 #pragma omp parallel reduction(+: comparisons) num_threads(NUM_THREADS)
145   {
146     int threadID = omp_get_thread_num();
147     char * pDB, * pSpec;
148     int8_t * pScores;
149     uint8_t * pLengthsDB, * pLengthsSpec;
150
151     pLengthsDB = &lengthsDB[threadID*seqsPerThread];
152     pDB = seqsDB + (offsetsDB[threadID])*sizeof(char);
153     pScores = &scores[threadID*seqsPerThread*numSeqsSpecimen];
154
155     for (uint32_t iDB = 0; iDB < seqsPerThread; ++iDB) {
156       #ifdef  PRINT_PROGRESS
157       if ( (iDB % 100) == 0 ) { // Beware of this affecting the time measurement!!!!!!
158         printf("%1.0f%% of DB entries processed...\r", ((float)iDB/seqsPerThread)*100 );
159         fflush(stdout);
160       }
161       #endif
162       pLengthsSpec = lengthsSpecimen;
163       pSpec = seqsSpecimen;
164       // Compare current DB entry with all specimen entries.
165       for (uint32_t iSpec = 0; iSpec < numSeqsSpecimen; ++ iSpec) {
166         *pScores++ = CalcScore(pDB, *pLengthsDB, pSpec, *pLengthsSpec);
167         ++ comparisons;
168         pSpec += *pLengthsSpec++;
169       }
170       pDB += *pLengthsDB++; // Pass to the next sequence in the DB
171     }
172   }
173
174   clock_gettime(CLOCK_MONOTONIC_RAW, &end);
175   clock_gettime(CLOCK_PROCESS_CPUTIME_ID, & endCPUTime);
176   elapsedTime = CalcTi
177   cpuUtilization = (do
178
179   return comparisons;
180 }
181
```

NAME	Time (s)	Performance (comps/s)
00_SW (OpenMP)	321.7	124 333

HW Implementation

—Copying the SW as a HW specification in Vitis HLS

Direct HW implementation

```
68 ////////////////////////////////////////////////////////////////////////
69 ////////////////////////////////////////////////////////////////////////
70 ////////////////////////////////////////////////////////////////////////
71 uint32_t SeqMatcher_HW(uint32_t numDBEntries, uint32_t numSeqsSpecimen,
72     char * seqsDB, char * seqsSpecimen, uint8_t * lengthsDB, uint8_t * lengthsSpecimen,
73     int8_t * scores)
74 {
75 #pragma HLS INTERFACE mode=s_axilite port=numDBEntries
76 #pragma HLS INTERFACE mode=s_axilite port=numSeqsSpecimen
77 #pragma HLS INTERFACE mode=s_axilite port=return
78 #pragma HLS INTERFACE mode=m_axi depth=1024 port=seqsDB offset=slave bundle=masterPort
79 #pragma HLS INTERFACE mode=m_axi depth=1024 port=seqsSpecimen offset=slave bundle=masterPort
80 #pragma HLS INTERFACE mode=m_axi depth=1024 port=lengthsDB offset=slave bundle=masterPort
81 #pragma HLS INTERFACE mode=m_axi depth=1024 port=lengthsSpecimen offset=slave bundle=masterPort
82 #pragma HLS INTERFACE mode=m_axi depth=1024 port=scores offset=slave bundle=masterPort num_write_outstanding=64 max_write_burst_length=32 latency=30
83
84
85   uint8_t * pLengthsDB = lengthsDB, * pLengthsSpec;
86   int8_t * pScores = scores;
87   char * pDB = seqsDB, * pSpec;
88   uint32_t comparisons = 0;
89   for (uint32_t iDB = 0; iDB < numDBEntries; ++iDB) {
90     pLengthsSpec = lengthsSpecimen;
91     pSpec = seqsSpecimen;
92     // Compare current DB entry with all specimen entries.
93     for (uint32_t iSpec = 0; iSpec < numSeqsSpecimen; ++ iSpec) {
94       *pScores++ = CalcScore(pDB, *pLengthsDB, pSpec, *pLengthsSpec);
95       ++ comparisons;
96       pSpec += *pLengthsSpec++;
97     }
98     pDB += *pLengthsDB++; // Pass to the next sequence in the DB
99   }
100
101   return comparisons;
102 }
```

NAME	Time (s)	Performance (comps/s)
00_SW (OpenMP)	321.7	124 333
01_Basic_HW	5 192.0	7 705

```
1  #include <stdint.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <math.h>
5  #include <ap_int.h>
6
7  #include "seqMatcher.h"
8
9
10 ////////////////////////////////////////////////////////////////////////
11 ////////////////////////////////////////////////////////////////////////
12 ////////////////////////////////////////////////////////////////////////
13 int8_t CalcScore(char * A, uint8_t lengthA, char * B, uint8_t lengthB)
14 {
15   // tableMax <- Max score computed in the table.
16   //
17   // Matrix M has one more row and one more column than the maximum size of seq1 and seq2. Therefore,
18   // index M[i][j] corresponds to A[i-1] and B[-1] if we use the indexes of M to access A and B.
19   // In the examples, A is laid on top of M left to right, B is laid on the left, vertically.
20   // Therefore, seq1=A, seq2=B
21
22 // A = ACACACAC
23 // B = TGTGTGTG
24 //
25 //           <-i->
26 //             A
27 //
28 //          ACACACAC
29 //         000000000
30 //       T 0
31 //       G 0
32 // |     T 0
33 // j B   G 0
34 // |     T 0
35 //       G 0
36 //       T 0
37 //       G 0
38
39   TPathMatrix M;
40   int8_t tableMax;
41   uint8_t i, j;
42
43   for (j = 0; j <= lengthA; ++ j) // <= because we are adding a 0 on the top and the left
44     M[0][j] = 0;
45   for (i = 0; i <= lengthB; ++ i)
46     M[i][0] = 0;
47
48   tableMax = -128; //tableMax_i = 0; tableMax_j = 0;
49   for (i = 1; i <= lengthB; ++ i) { // Use <= because we have in M one extra zero on the top and the left.
50     for (j = 1; j <= lengthA; ++ j) {
51       int8_t hitValue, top, left, max;
52       hitValue = M[i-1][j-1] + ( (A[j-1] == B[i-1]) ? 1 : -1 ); // A and B start on
53       top = M[i-1][j] - 1;
54       left = M[i][j-1] - 1;
55       max = hitValue > top ? (hitValue > left ? hitValue : left) : (top > left ? top : left);
56       if (max < 0) max = 0;
57       if (max > tableMax) {
58         tableMax = max;
59       }
60       M[i][j] = max;
61     }
62   }
63
64   return tableMax;
65 }
```

M is stored inside the FPGA as a BRAM or registers

A and *B* are accessed directly from the external DRAM

- Why do we get such a huge reduction in performance?
  - The HW accesses every character of the strings from DRAM
- How can we improve the design?

Direct HW implementation with sequence caching

- Exercise: Modify the HW description to copy each sequence in an internal buffer
  - Call *CalcScore()* using the internal buffers
  - Copy one DB sequence and keep it in the buffer while comparing against all the specimen sequences
  - Then, repeat for each DB sequence

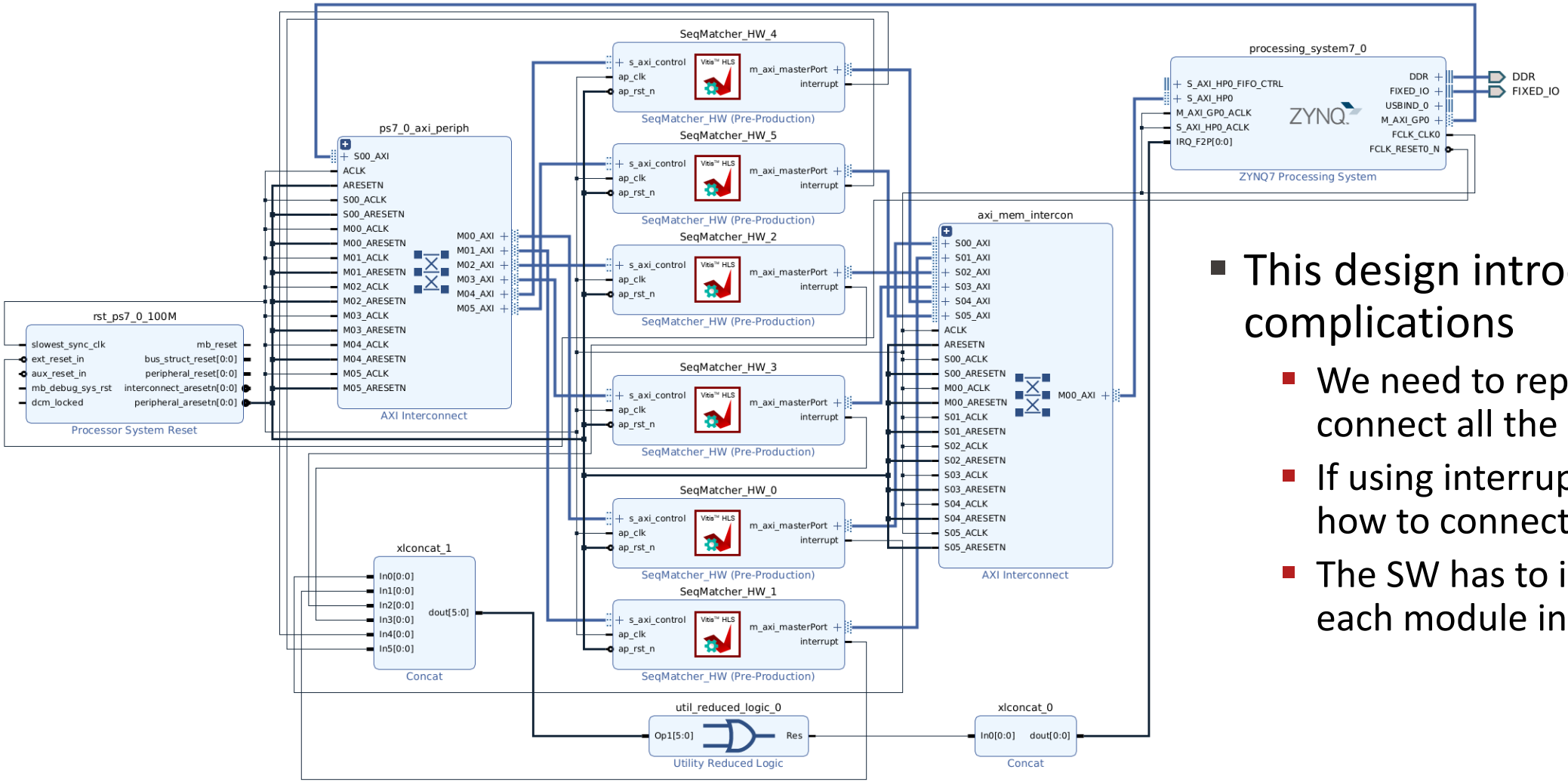
NAME	Time (s)	Performance (comps/s)
00_SW (OpenMP)	321.7	124 333
01_Basic_HW	5 192.0	7 705
03_CacheLines	379.6	105 362

- We obtain a significant improvement in performance,
  - but we are still slower than the SW version
- How can we improve performance even more?

HW module replication

HW module replication

- After replication, we obtain a system design like this:



- This design introduces many complications
  - We need to replicate and connect all the modules
  - If using interrupts, decide how to connect them
  - The SW has to interact with each module independently!

SW interaction with each HW module

- We need to configure each HW module independently
  - `CSeqMatcherDriver seqMatchers[NUM_MODULES];`

```
25 ////////////////////////////////////
26 // Address constants
27 #define NUM_MODULES 6
28 const uint32_t MAP_SIZE = 64*1024; // Size of address range mapped to the adder registers
29 const uint32_t SEQMATCHER_HW_ADDR[NUM_MODULES] = {0x40000000, 0x40010000, 0x40020000, 0x40030000, 0x40040000, 0x40050000};
30
31
32 ////////////////////////////////////////////////////////////////////////////
33 bool InitDevice(CSeqMatcherDriver * seqMatchers, uint32_t numDBEntries, uint32_t numSeqsSpecimen,
34     char * &seqsDB, char * &seqsSpecimen,
35     uint8_t * &lengthsDB, uint8_t * &lengthsSpecimen, int8_t * &scores, bool log=true)
36 {
37   printf("\n\nThis program requires that the bitstream is loaded in the FPGA.\n");
38   printf("This program has to be run with sudo.\n");
39   printf("Press ENTER to confirm that the bitstream is loaded (proceeding without it can crash the board).\n\n");
40   //getchar(); // Commented to allow execution in batch
41
42   for (uint32_t ii = 0; ii < NUM_MODULES; ++ ii) {
43     if ( seqMatchers[ii].Open(SEQMATCHER_HW_ADDR[ii], MAP_SIZE) != CAccelDriver::OK ) {
44       printf("Error mapping device at physical address 0x%08X\n", SEQMATCHER_HW_ADDR[ii]);
45       return false;
46     }
47     if (log)
48       printf("Device at physical address 0x%08X successfully mapped into the application virtual address space\n",
49              SEQMATCHER_HW_ADDR[ii]);
50   }
51
52   // Allocate DMA memory for use by the device. We receive addresses in the *virtual* address space of the application.
53   if (log)
54     printf("Allocating DMA memory...\n");
55
56   seqsDB = (char *)seqMatchers[0].AllocDMACompatible(numDBEntries*MAX_SEQ_LENGTH*sizeof(char));
57   lengthsDB = (uint8_t *)seqMatchers[0].AllocDMACompatible(numDBEntries*sizeof(uint8_t));
58   seqsSpecimen = (char *)seqMatchers[0].AllocDMACompatible(numSeqsSpecimen*MAX_SEQ_LENGTH*sizeof(char));
59   lengthsSpecimen = (uint8_t *)seqMatchers[0].AllocDMACompatible(numSeqsSpecimen*sizeof(uint8_t));
60   scores = (int8_t *)seqMatchers[0].AllocDMACompatible(numDBEntries*numSeqsSpecimen*sizeof(int8_t));
61
62   if ( (seqsDB == NULL) || (lengthsDB == NULL) || (seqsSpecimen == NULL) || (lengthsSpecimen == NULL) || (scores == NULL) ) {
63     printf("Error allocating DMA memory.\n");
64     return false;
65   }
66
67   if (log) {
68     printf("DMA memory allocated.\n");
69     printf("seqsDB: Virtual address: 0x%08X (%u)\n", (uint32_t)seqsDB, (uint32_t)seqsDB);
70     printf("lengthsDB: Virtual address: 0x%08X (%u)\n", (uint32_t)lengthsDB, (uint32_t)lengthsDB);
71     printf("seqsSpecimen: Virtual address: 0x%08X (%u)\n", (uint32_t)seqsSpecimen, (uint32_t)seqsSpecimen);
72     printf("lengthsSpecimen: Virtual address: 0x%08X (%u)\n", (uint32_t)lengthsSpecimen, (uint32_t)lengthsSpecimen);
73     printf("scores: Virtual address: 0x%08X (%u)\n", (uint32_t)scores, (uint32_t)scores);
74   }
75
76   return true;
77 }
```

```
150 ////////////////////////////////////////////////////////////////////////////
151 uint32_t SeqMatcher_HW(CSeqMatcherDriver * seqMatchers,
152     uint32_t numDBEntries, uint32_t numSeqsSpecimen,
153     char * seqsDB, char * seqsSpecimen, uint8_t * lengthsDB, uint8_t * lengthsSpecimen,
154     int8_t * scores, uint64_t & elapsedTime, uint64_t & elapsedCPU)
155 {
156   struct timespec start, end;
157   struct timespec startCPUTime, endCPUTime;
158   uint32_t numComparisons = 0;
159   uint32_t remaining = numDBEntries, indexDB = 0, indexScores = 0;
160   uint32_t offsetDB = 0;
161   uint32_t partialComparisons[NUM_MODULES];
162   uint32_t maxBatchSize = MAX_SEQS_DB;
163
164   elapsedTime = 0; elapsedCPU = 0;
165   clock_gettime(CLOCK_PROCESS_CPUTIME_ID, & startCPUTime);
166   clock_gettime(CLOCK_MONOTONIC_RAW, &start);
167
168   while (remaining > 0) {
169     uint32_t batchSize;
170
171     // Ensure that the last iteration has a similar size per module.
172     if (maxBatchSize > (remaining / NUM_MODULES))
173       maxBatchSize = (remaining / NUM_MODULES) + 1;
174
175     for (uint32_t iModule = 0; iModule < NUM_MODULES; ++ iModule) {
176       batchSize = remaining > maxBatchSize ? maxBatchSize : remaining;
177       remaining -= batchSize;
178       printf("Executing batch of size: %u, Remaining: %u Index: %u\n", batchSize, remaining, indexDB);
179
180       if ( seqMatchers[iModule].SeqMatcher_HW_Start(batchSize, numSeqsSpecimen, seqsDB, offsetDB, seqsSpecimen, 0,
181                     lengthsDB, indexDB * sizeof(lengthsDB[0]), lengthsSpecimen, 0,
182                     scores, indexScores * sizeof(scores[0])) != CAccelDriver::OK ) {
183         printf("ERROR CALLING HW.\n");
184         return 0;
185       }
186
187       // Compute offset in DB sequences, since sequences have variable length
188       for (uint32_t ii = 0; ii < batchSize; ++ ii)
189         offsetDB += lengthsDB[indexDB++];
190       indexScores += batchSize * numSeqsSpecimen;
191     }
192
193     for (uint32_t iModule = 0; iModule < NUM_MODULES; ++ iModule) {
194       seqMatchers[iModule].SeqMatcher_HW_Stop(partialComparisons[iModule]);
195       numComparisons += partialComparisons[iModule];
196     }
197   }
198
199   clock_gettime(CLOCK_MONOTONIC_RAW, &end);
200   clock_gettime(CLOCK_PROCESS_CPUTIME_ID, & endCPUTime);
201   elapsedTime += CalcTimeDiff(end, start);
202   elapsedCPU += CalcTimeDiff(endCPUTime, startCPUTime);
203
204   return numComparisons;
205 }
```

SW interaction with each HW module

```
10 uint32_t CSeqMatcherDriver::SeqMatcher_HW_Start(uint32_t numDBEntries, uint32_t numSeqsSpecimen,
11         void * seqsDB, uint32_t offsetDB, void * seqsSpecimen, uint32_t offsetSpec,
12         void * lengthsDB, uint32_t offsetLengthsDB, void * lengthsSpecimen, uint32_t offsetLengthsSpecimen,
13         void * scores, uint32_t offsetScores)
14 {
15   volatile TRegs * regs = (TRegs*)accelRegs;
16   uint32_t phySeqsDB, phySeqsSpecimen, phyLengthsDB, phyLengthsSpecimen, phyScores;
17   uint32_t status;
18
19   if (logging)
20     printf("CSeqMatcherDriver::SeqMatcher_HW(numDBEnttries=%u, numSeqsSpecimen=%u, seqsDB=0x%08X, seqsSpecimen=0x%08X, "
21           "lengtsDB=0x%08X, lengthsSpecimen=0x%08X, scores=0x%08X\n",
22           (uint32_t)numDBEntries, (uint32_t)numSeqsSpecimen, (uint32_t)seqsDB, (uint32_t)seqsSpecimen,
23           (uint32_t)lengthsDB, (uint32_t)lengthsSpecimen, (uint32_t)scores);
24
25   if (accelRegs == NULL) {
26     if (logging)
27       printf("Error: Calling SeqMatcher_HW() on a non-initialized accelerator.\n");
28     return DEVICE_NOT_INITIALIZED;
29   }
30
31   // We need to obtain the physical addresses corresponding to each of the virtual addresses passed by the application.
32   // The accelerator uses only the physical addresses (and only contiguous memory).
33   phySeqsDB = GetDMAPhysicalAddr(seqsDB);
34   if (phySeqsDB == 0) {
35     if (logging)
36       printf("Error: No physical address found for virtual address 0x%08X\n", (uint32_t)seqsDB);
37     return VIRT_ADDR_NOT_FOUND;
38   }
39   phySeqsSpecimen = GetDMAPhysicalAddr(seqsSpecimen);
40   if (phySeqsSpecimen == 0) {
41     if (logging)
42       printf("Error: No physical address found for virtual address 0x%08X\n", (uint32_t)seqsSpecimen);
43     return VIRT_ADDR_NOT_FOUND;
44   }
45   phyLengthsDB = GetDMAPhysicalAddr(lengthsDB);
46   if (phyLengthsDB == 0) {
47     if (logging)
48       printf("Error: No physical address found for virtual address 0x%08X\n", (uint32_t)lengthsDB);
49     return VIRT_ADDR_NOT_FOUND;
50   }
51   phyLengthsSpecimen = GetDMAPhysicalAddr(lengthsSpecimen);
52   if (phyLengthsSpecimen == 0) {
53     if (logging)
54       printf("Error: No physical address found for virtual address 0x%08X\n", (uint32_t)lengthsSpecimen);
55     return VIRT_ADDR_NOT_FOUND;
56   }
57   phyScores = GetDMAPhysicalAddr(scores);
58   if (phyScores == 0) {
59     if (logging)
60       printf("Error: No physical address found for virtual address 0x%08X\n", (uint32_t)scores);
61     return VIRT_ADDR_NOT_FOUND;
62   }
63
64   regs->numDBEntries = numDBEntries;
65   regs->numSeqsSpecimen = numSeqsSpecimen;
66   regs->seqsDB = phySeqsDB + offsetDB;
67   regs->seqsSpecimen = phySeqsSpecimen + offsetSpec;
68   regs->lengthsDB = phyLengthsDB + offsetLengthsDB;
69   regs->lengthsSpecimen = phyLengthsSpecimen + offsetLengthsSpecimen;
70   regs->scores = phyScores + offsetScores;
71
72   if (logging)
73     printf("\nStarting accel...\n");
74
75   status = regs->control;
76   status |= 1;  // Set to 1 ap_start
77   regs->control = status;
78
79   return OK;
80 }
```

```
82 uint32_t CSeqMatcherDriver::SeqMatcher_HW_Stop(uint32_t & numComparisons)
83 {
84   volatile TRegs * regs = (TRegs*)accelRegs;
85   uint32_t status;
86
87   do {
88     status = regs->control;
89     usleep(1000);
90   } while ( ( (status & 2) != 2) ); // wait until ap_done==1
91
92   numComparisons = regs->returnValue;
93
94   return OK;
95 }
```

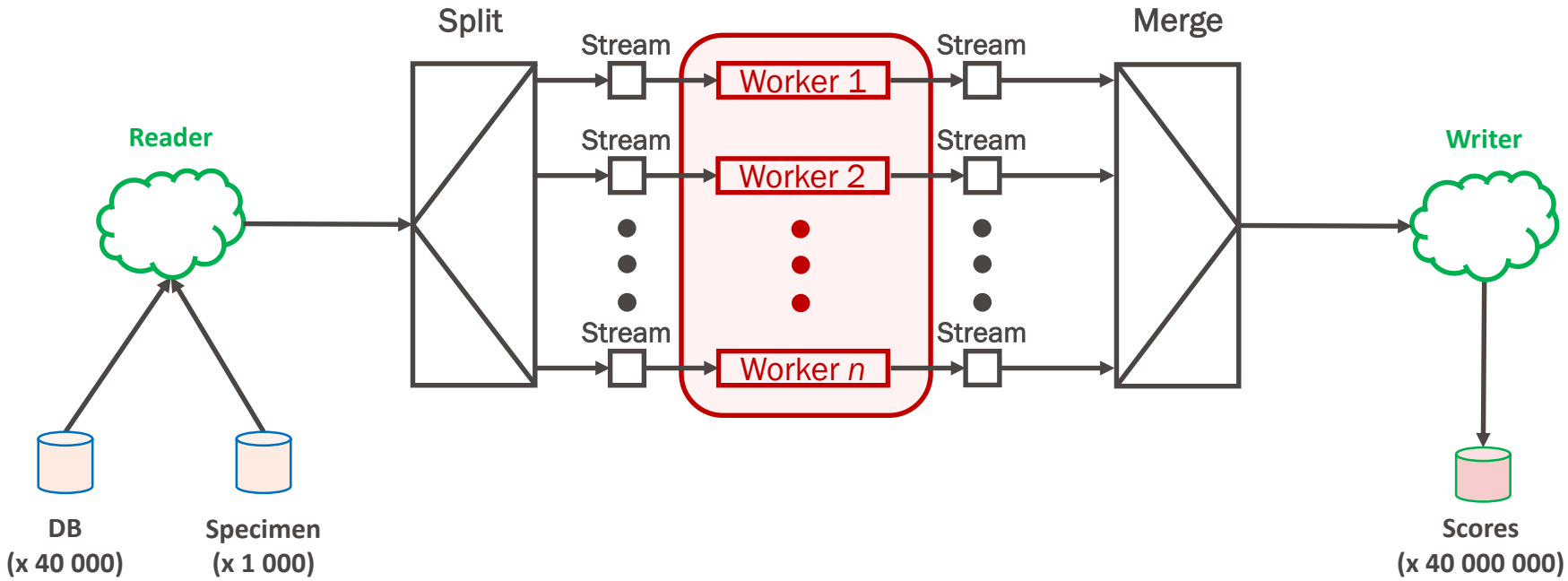
Why do we need to split the *start* and *stop* actions?

What other possibilities are there? For example, using a device driver and multiple threads.

Worker replication

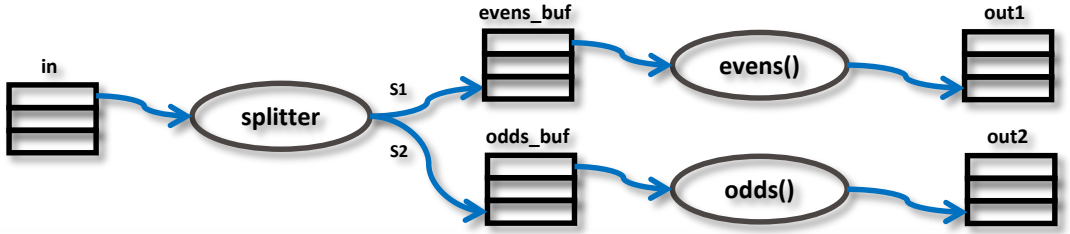
—With HLS tasks and streams

System structure with split/merge streams and task workers



Use of HLS tasks and streams

- A task is a function that is executed infinitely → HW module that processes an input as soon as it's available
  - Represented with hls::task
  - No explicit calls to the function
  - Can only read streams, not AXI ports!
- Streams provide read() and write() methods through hls::stream<T>
- In SW, equivalent to a thread executing a function, with input and output channels or FIFOs
- Useful when not interacting with SW, no explicit start/stop conditions, just flow of data

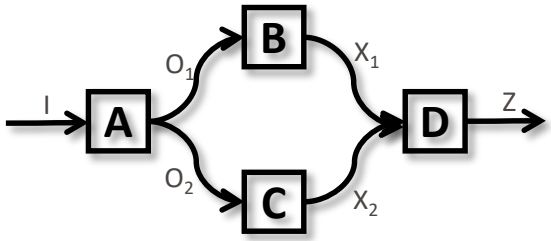



```
20  void odds_and_evens(hls::stream<int> &in, hls::stream<int> &out1,
21                      hls::stream<int> &out2) {
22    hls_thread_local hls::stream<int> s1; // channel connecting t1 and t2
23    hls_thread_local hls::stream<int> s2; // channel connecting t1 and t3
24    // t1 infinitely runs function splitter, with input in and outputs s1 and s2
25    hls_thread_local hls::task t1(splitter, in, s1, s2);
26    // t2 infinitely runs function odds, with input s1 and output out1
27    hls_thread_local hls::task t2(odds, s1, out1);
28    // t3 infinitely runs function evens, with input s2 and output out2
29    hls_thread_local hls::task t3(evens, s2, out2);
30  }
```

```
1   #include "test.h"
2
3   void splitter(hls::stream<int> &in, hls::stream<int> &odds_buf,
4                 hls::stream<int> &evens_buf) {
5     int data = in.read();
6     if (data % 2 == 0)
7     evens_buf.write(data);
8     else
9     odds_buf.write(data);
10  }
11
12  void odds(hls::stream<int> &in, hls::stream<int> &out) {
13    out.write(in.read() + 1);
14  }
15
16  void evens(hls::stream<int> &in, hls::stream<int> &out) {
17    out.write(in.read() + 2);
18  }
```

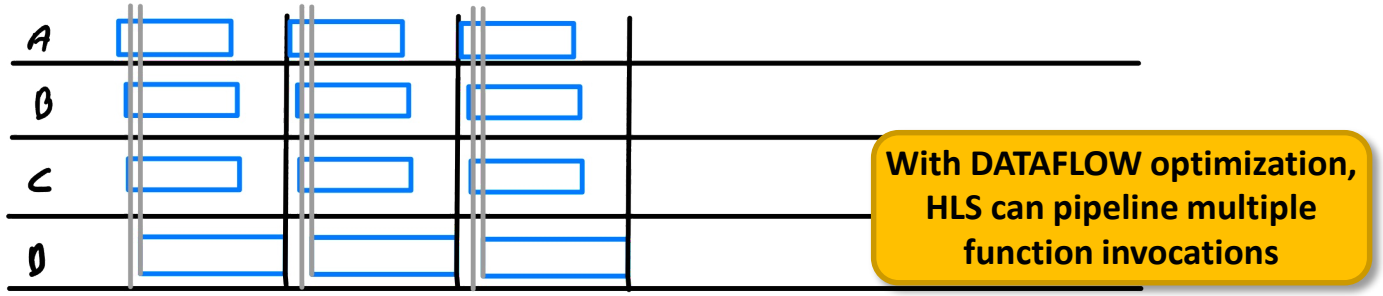
The HLS dataflow pragma

- The *dataflow* pragma defines a region in which functions and loops can overlap in their operation
  - This creates potential for higher parallelism
  - Operations in a loop or in a function can start before the previous loop or function has finished
  - Creating a task-level pipeline architecture of concurrent processes
  - Multiple sequential functions can be started simultaneously
  - HLS generates channels (e.g., FIFOs) that decouple the producers and consumers
  - Inside a dataflow region: **only variable declarations and function calls!!!**




```
A(I, O1, O2);
B(O1, X1);
C(O2, X2);
D(X1, X2, Z);
```

Let's assume that the complete process is invoked 3 times



E.g., process 3 independent images, each one analyzed row-by-row. The functions can be pipelined by rows for each image

The HLS split and merge classes

- Allow splitting and merging of jobs between multiple workers
- They work like streams that create one-to-many and many-to-one connections
- The split class has one input and multiple outputs
  - It distributes elements from its input towards its outputs, using round robin or load balancing
- The merge class has multiple inputs and one output
  - It reads elements from its inputs using round robin or load balancing, and sends all of them thru its output
- The specification of the split/merge channels is:
  - `hls::split::round_robin<DATATYPE, NUM_PORTS[, DEPTH]> name`
  - `hls::split::load_balancing<DATATYPE, NUM_PORTS[, DEPTH, N_PORT_DEPTH]> name;`
  - `hls::merge::round_robin<DATATYPE, NUM_PORTS[, DEPTH]> name`
  - `hls::merge::load_balancing<DATATYPE, NUM_PORTS[, DEPTH]> name`
- With dataflow, we can create:
  - One module to read sequences and send pairs to the workers
  - Multiple workers that receive two sequences each, and produce an output score
  - One module to receive the scores of the workers and write them back to the main memory

Putting everything together

- Define a data type to pass information through the channels
- Modify the worker definition to use channels

```
1  #ifndef SEQMATCHER_H
2  #define SEQMATCHER_H
3
4  #define MAX_SEQ_LENGTH 32
5  #define NUM_WORKERS 8
6
7  typedef int8_t TPathMatrix[MAX_SEQ_LENGTH+1][MAX_SEQ_LENGTH+1];
8
9  struct TDataIn {
10   char A
11   uint8_t lengthA;
12   char B
13   uint8_t lengthB;
14   uint32_t queryID; // To write outputs in right order
15 };
16
17 struct TDataOut {
18   uint32_t queryID;
19   int8_t score;
20 };
21
22 ////////////////////////////////
23 void CalcScore(hls::stream<TDataIn> & input, hls::stream<TDataOut> & output);
24
25 uint32_t SeqMatcher_HW(uint32_t numDBEntries, uint32_t numSeqsSpecimen,
26     char * seqsDB, char * seqsSpecimen, uint8_t * lengthsDB, uint8_t * lengthsSpecimen,
27     int8_t * scores);
28
29
30 #endif // SEQMATCHER_H
31
```

The number of workers is configurable

Putting everything together

```
65 ////////////////////////////////////////////////////////////////////////////////
66 ////////////////////////////////////////////////////////////////////////////////
67 ////////////////////////////////////////////////////////////////////////////////
68 uint32_t SeqMatcher_HW(uint32_t numDBEntries, uint32_t numSeqsSpecimen,
69     char * seqsDB, char * seqsSpecimen, uint8_t * lengthsDB, uint8_t * lengthsSpecimen,
70     int8_t * scores)
71 {
72 #pragma HLS INTERFACE mode=s_axilite port=numDBEntries
73 #pragma HLS INTERFACE mode=s_axilite port=numSeqsSpecimen
74 #pragma HLS INTERFACE mode=s_axilite port=return
75 #pragma HLS INTERFACE mode=m_axi depth=1024 port=seqsDB offset=slave bundle=masterPort
76 #pragma HLS INTERFACE mode=m_axi depth=1024 port=seqsSpecimen offset=slave bundle=masterPort
77 #pragma HLS INTERFACE mode=m_axi depth=1024 port=lengthsDB offset=slave bundle=masterPort
78 #pragma HLS INTERFACE mode=m_axi depth=1024 port=lengthsSpecimen offset=slave bundle=masterPort
79 #pragma HLS INTERFACE mode=m_axi depth=1024 port=scores offset=slave bundle=masterPort num_write_outstanding=64 max_write_burst_length=32 latency=30
80
81
82   hls_thread_local hls::split::round_robin<TDataIn, NUM_WORKERS> workerInputs;
83   hls_thread_local hls::merge::round_robin<TDataOut, NUM_WORKERS, NUM_WORKERS> workerOutputs;
84   hls_thread_local hls::task workers[NUM_WORKERS];
85
86   #pragma HLS dataflow
87   ReadInputs(numDBEntries, numSeqsSpecimen, seqsDB, seqsSpecimen, lengthsDB, lengthsSpecimen, workerInputs.in);
88
89   workers_gen: for (uint32_t ii = 0; ii < NUM_WORKERS; ++ ii) {
90   #pragma HLS unroll
91     workers[ii](CalcScore, workerInputs.out[ii], workerOutputs.in[ii]);
92   }
93
94   WriteOutputs(workerOutputs.out, scores, numDBEntries, numSeqsSpecimen);
95
96   return numDBEntries * numSeqsSpecimen;
97 }
```

Define the split/merge streams

Declare an *array* of workers

One function to read and split data

One function to gather and write outputs

And a configurable number of workers to process the data

With this technique, we can configure any number of workers without modifying the design in Vivado nor the SW that interacts with the accelerator

```
7  #include <hls_task.h>
8  #include <hls_np_channel.h>
9
10 #include "seqMatcher.h"
11
12
13 ////////////////////////////////////////////////////////////////////////////////
14 ////////////////////////////////////////////////////////////////////////////////
15 ////////////////////////////////////////////////////////////////////////////////
16 void CalcScore(hls::stream<TDataIn> & input, hls::stream<TDataOut> & output)
17 {
18   TPathMatrix M;
19   int8_t tableMax;
20   TDataOut dataOut;
21   TDataIn dataIn;
22
23   // Read from input stream
24   dataIn = input.read();
25
26   output.write(dataOut);
27 }
28
29 ////////////////////////////////////////////////////////////////////////////////
30 ////////////////////////////////////////////////////////////////////////////////
31 ////////////////////////////////////////////////////////////////////////////////
32 void ReadInputs(uint32_t numDBEntries, uint32_t numSeqsSpecimen,
33   char * seqsDB, char * seqsSpecimen, uint8_t * lengthsDB, uint8_t * lengthsSpecimen,
34   hls::stream<TDataIn> & inputStream)
35 {
36   uint8_t * pLengthsDB = lengthsDB, * pLengthsSpec;
37   char * pDB = seqsDB, * pSpec;
38   TDataIn dataIn;
39
40   for (uint32_t iDB = 0; iDB < numDBEntries; ++iDB) {
41     memcpy(dataIn.A, pDB, *pLengthsDB);
42
43     // Dispatch the comparison of current DB entry with all specimen entries.
44     for (uint32_t iSpec = 0; iSpec < numSeqsSpecimen; ++ iSpec) {
45       memcpy(dataIn.B, pSpec, *pLengthsSpec);
46
47       inputStream.write(dataIn);
48     }
49   }
50 }
51
52 ////////////////////////////////////////////////////////////////////////////////
53 ////////////////////////////////////////////////////////////////////////////////
54 ////////////////////////////////////////////////////////////////////////////////
55 void WriteOutputs(hls::stream<TDataOut> & outputStream, int8_t * scores,
56   uint32_t numDBEntries, uint32_t numSeqsSpecimen)
57 {
58   for (uint32_t ii = 0; ii < numDBEntries * numSeqsSpecimen; ++ ii) {
59     TDataOut output = outputStream.read();
60
61   }
62 }
```

Analysis of results

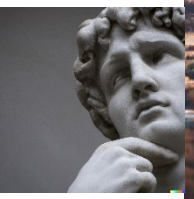
- Using the multiple workers, we obtain the following results:

NAME	Time (s)	Performance (comps/s)
00_SW (OpenMP)	321.7	124 333
01_Basic_HW	5 192.0	7 705
03_CacheLines	379.6	105 362
05_Workers_x2	179.2	223 174
05_Workers_x4	90.4	442 441
05_Workers_x8	56.7	705 471
05_Workers_x10	54.4	734 630
05_Workers_x16	54.4	735 597
05_Workers_x24	54.4	735 727

- Why does performance saturate?
  - Give four reasons
- A) Problem of output bandwidth?
- B) Is the split unit too slow as a job scheduler?
- C) All the workers share the same input port?
  - Replicate a module with 8 workers multiple times?
- D) Total memory bandwidth
  - BD file is 851 979 B
  - Specimen file is 21 366 B
  - With line caching, the system reads a total of ~850 MiB of data
  - Bandwidth is ~100 MHz @ 8-bit = 100 MiB/s
    - Check if we perform 8- or 32-bit reads!
  - Total reading time is ~8 s, without counting burst starting times
- How can we push performance more?
  - Last year, the students of final project managed to reach < 1 s!
- E) Faster workers?

Instructions for the exercise

- To complete the exercise of this session:
  - Execute version *00_SW* to get familiar with the problem and the execution options
    - Generate the scores file and verify its MD5 checksum
  - Use the provided *Makefile* to generate the bitstream for version *01_Basic_HW*
    - Synthesize the design with make bitstream
    - Transfer the bitstream and the Linux application to the Pynq board
    - Execute the application and verify the correct results (use a reduced input size)
  - Analyze and complete version *03_Basic_CacheLines*
    - Complete the code of *HLS/seqMatcher.cpp* following the **@TODO** comments
    - The HW should perform a copy of the strings into an internal buffer and call *CalcScore()* using those buffers
    - Analyze the performance of this version
  - Analyze and complete version *05_Workers*
    - Complete the code of *HLS/seqMatcher.cpp* and *HLS/seqMatcher.h*
      - Complete the configuration of the streams and tasks
      - Modify the *CalcScore()* function to operate on the data received from the streams
      - Complete the *Read()* and *Write()* functions to use the stream interface
    - Analyze the performance of this version
    - Experiment with varying numbers of workers



Questions?

Prof. David Atienza

EPFL – Embedded Systems Laboratory
david.atienza@epfl.ch