Expressing Parallelism

Using the DKU Pattern

Sean Halle, UC Santa Cruz and INRIA Saclay

How the DKU Pattern Is Used

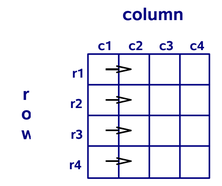
- Programmer identifies portions of the code that:
  - operate on a data structure
  - can operate on sub-portions of the data-structure independently
- Once found, such code is isolated into its own FComponent
- The internals of that FComponent consist of 4 sub FComponents:
  - A scheduler symbol that represents behavior of the built-in execution model
  - A divider that splits the data-structure into pieces, preferably in a “meta” way
  - A kernel that operates on one piece
  - An undivider that assembles the individual results coming out of the kernel into one combined result

The Code Rules

- No call-return between FComponents, only send a message *to* an FComponent
- Any FComponent can be duplicated and copy run simultaneously on a diff msg
  - A duplicate of an FComponent has no state in common with its duplicates.
  - All variables inside an FComponent are either pointers or have local scope.
  - State is only persisted in messages and data-strucs pointed to in messages.
- No threads, nor locks, nor synchronization primitives
- Threads are replaced by:
  - duplicating FComponents
  - dividing a data structure into pieces and duplicating the Kernel FComponent for each
- Programmer ensures that side-effects written by one FComponent cannot be seen by any other FComponent, including a copy, until after the writing-FComponent has completed execution of all its code then sent out a message.
- The send-receive of a message serves the purpose that synchronization used to

Example of Using Divider-Kernel-Undivider: Deblocking in SVC Decoder

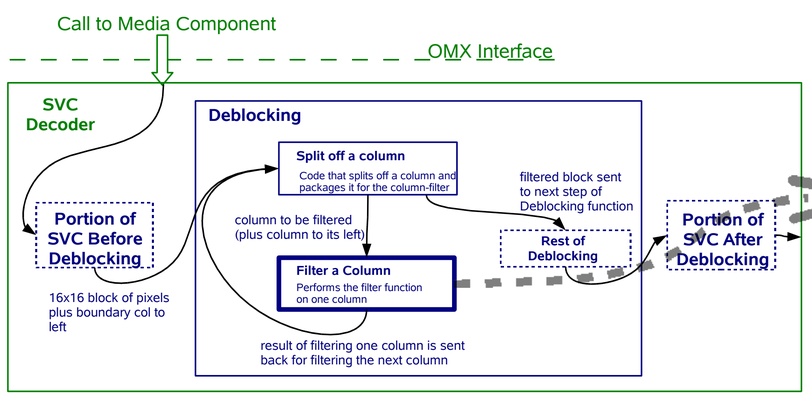
- SVC Decoder spends a large portion of its time in the “Deblocking” function
- Deblocking function receives a 16 by 16 block of pixels to which it applies a filter
- Filtering a column in the block requires the results of the column to the left
- Within a column, filtering a row is independent of filtering the other rows
- Apply Divider-Kernel-Undivider (DKU) pattern to the filtering of a single column



Column C1 is filtered,
then the result used to filter C2,
then that is used for C3 and so on.
Each row is independent of the other rows.

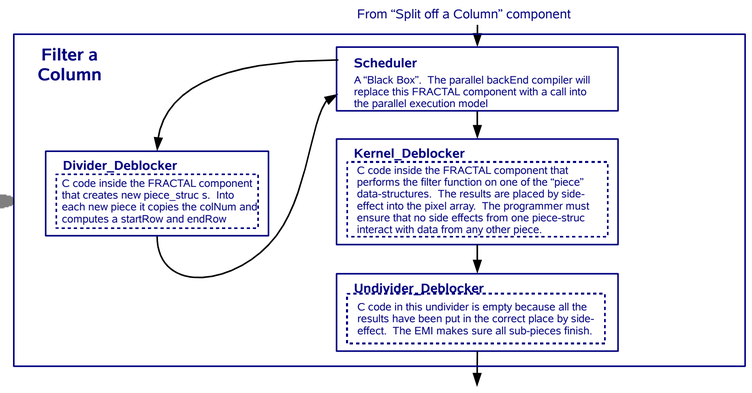
Strategy:
peel off the columns, one by one
divide each column into pieces, one or more rows to each piece
filter the pieces independently, until all done, then go on to the next column

Deblocking in terms of FComponents



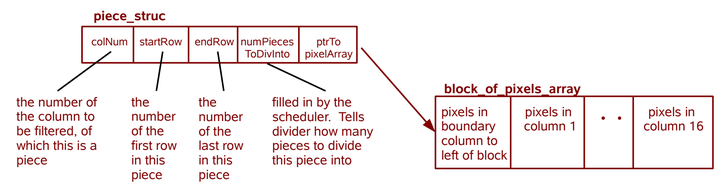
- Shows only FComponents surrounding the 'Filter a Column' portion of Deblocking
- 'Filter a Column' is separated into its own FComponent
- 'Filter a Column' receives a data-struc with info about the column-to-be-filtered

The 'Filter a Column' FComponent



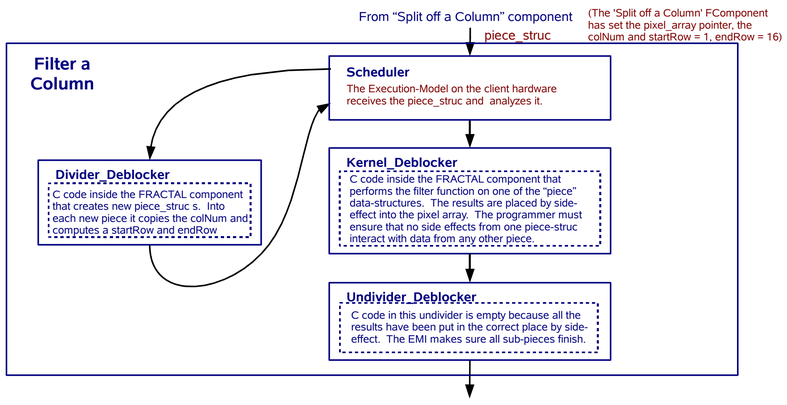
- 'Filter a Column' FComponent is parallelized with DKU pattern
- Use a naming convention: for example, a divider's name starts with “Divider_” followed by something specific chosen by the programmer.

Data Structures Used



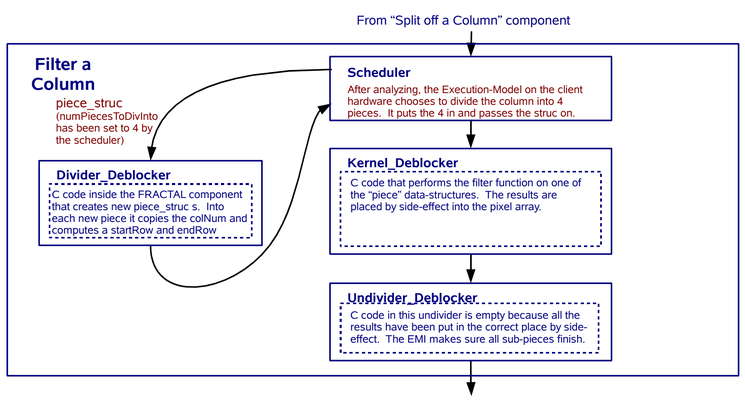
- The block_of_pixels_array holds the actual pixel values.
- The piece_struc only represents which portions of the array are part of the “piece”
  - The piece_struc is, in a sense “meta” because it doesn't contain any actual pixel data, it only has information that allows calculating the indexes of the pixels that are in the piece

Walk Through of Behavior at Run Time



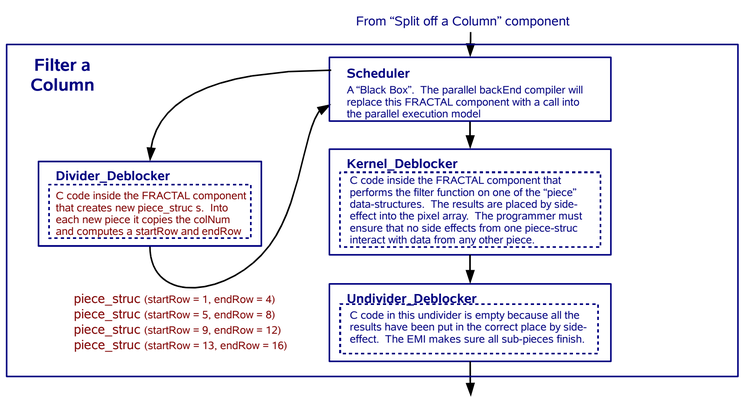
- When the code runs, the Scheduler FComponent is replaced by a portion of the execution model that was written for the hardware running the code.
- 'SplitOffAColumn' sends a piece_struc for an entire column to the scheduler

Walk Through of Behavior at Run Time



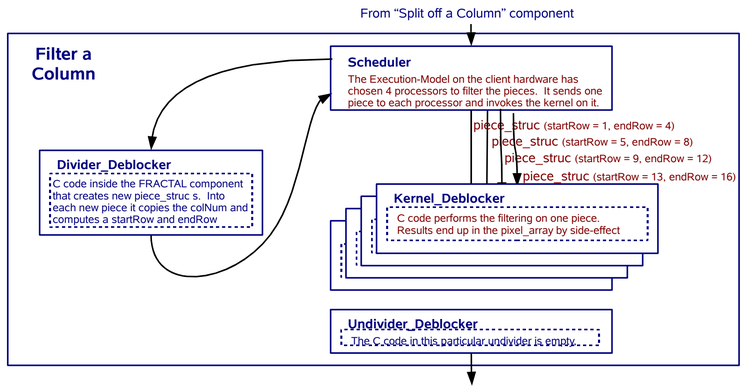
- The scheduler-implementation examines the incoming piece_struc, looks at the state of the processors, and decides how many pieces to divide the struc into
- It puts that number into the piece_struc and passes it on to the divider

Walk Through of Behavior at Run Time



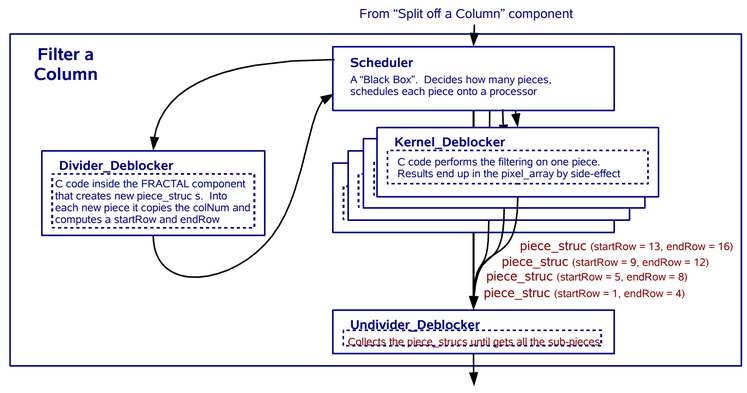
- On this hardware, the Scheduler has chosen to split a column into 4 pieces
- The divider sees the '4' in the incoming piece_struc
- Divider creates 4 new piece_strucs, each with a different range of rows, and sends them back to the scheduler

Walk Through of Behavior at Run Time



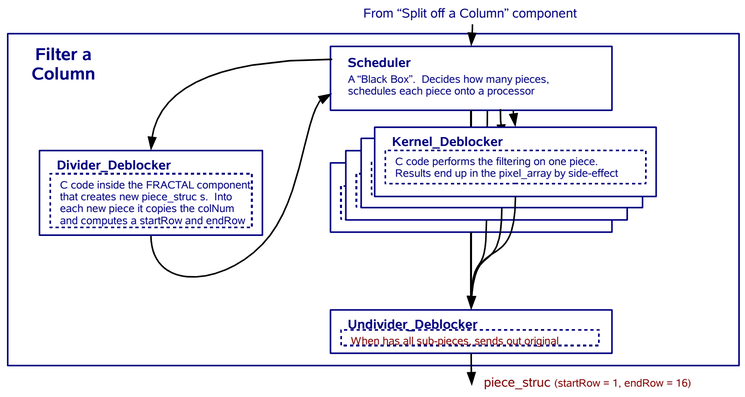
- The Scheduler chooses a processor to send each piece to
- A separate copy of the Kernel code is run on each of those processors
- Each processor performs the filter function on its piece, causing the results to appear by side-effect in the pixel-array (for this shared-memory hardware)

Walk Through of Behavior at Run Time



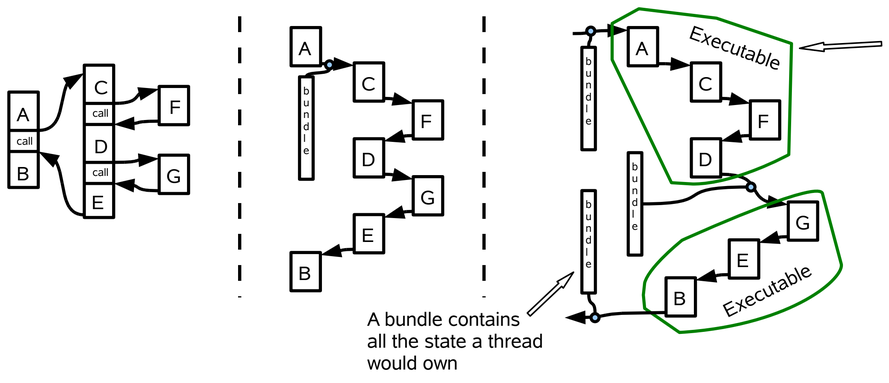
- When finished, the piece_struc is sent on to the undivider
- The undivider keeps track of which piece_strucs have come in, to know when the whole column is done

Walk Through of Behavior at Run Time



- When all the individual pieces have come in, the undivider sends the original piece_struck that came in to 'Filter a Column' on to whatever comes next

Inside DKU Compiler



Executables are packaged by the compiler. At run-time, the parallel execution model pairs an executable with a bundle, and queues each set for execution on a processor.

Each executable + bundle is the equivalent of a thread time-slice. Imagine a thread performing exactly the instructions to complete the executable during one time slice. The change in state over that time-slice is exactly the change in a bundle caused by running the e+b set.

Organizing as executables + bundles eliminates the need for synchronizations and locks in application code.

All accesses to shared data must be disjoint between any two e+b sets.

Parallel Execution Model Implementation (run-time)

Executable + Bundle