In the AltaPixShare software app, when the user drags a group of images to the target location, the following code iterates over the list of images, and for each image creates the destination file:

```
{CODE brush: delphi; ruler: true;}for i := 0 to Pred(ActiveImages.Count) do
 begin
 try
 CurrentImage := TProcessedImage(ActiveImages[i]);
 except
 CurrentImage := nil;
 end;
 if Assigned(CurrentImage)
 then begin
 CurrentImage.CreateDestFile();
 end;
 end;{/CODE}
```

Given that applying a filter to a large image and then writing the output image into a JPEG file is a time-consuming task, this is a clear case for multi-threaded optimization, so that multiple images are processed at the same time on different CPU cores. And using the amazing Async Calls library , we can achieve this result with only a few lines of code. These are the steps required to turn this serial code into a multi-threaded one:

1. add the AsyncCalls unit to uses list
2. add a variable-length array that contains the handles to the various async calls: {CODE brush: delphi; ruler: false;}ThreadCallsArray : array of IAsyncCall;{/CODE}
3. set the size of the variable-length array to match the number of images to be processed in parallel: {CODE brush: delphi; ruler: false;}SetLength(ThreadCallsArray, ActiveImages.Count);{/CODE}
4. instead of directly calling the method that builds the output file for each image in the list, create an async call for each image: {CODE brush: delphi; ruler: false;}ThreadCallsArray[i] := AsyncCall(@CreateDestFileOfImage, CurrentImage);{/CODE}
5. finally, wait for all async calls to complete, and proceed with other computations: {CODE brush: delphi; ruler: false;}WaitResult := AsyncMultiSync(ThreadCallsArray, True, INFINITE);{/CODE}

The only troublesome spot in the code above is that the AsyncCall does not let you use the normal method invocation (CurrentImage.CreateDestFile()) but requires the usage of a custom-defined procedure that receives the object as a parameter and invokes the CreateDestFile on the given object:

{CODE brush: delphi; ruler: true;}procedure CreateDestFileOfImage(CurrentImage : TProcessedImage);
begin
 if Assigned(CurrentImage)
 then CurrentImage.CreateDestFile;
end;{/CODE}

So here is the code from AltaPixShare that does the parallel processing of images:

{CODE brush: delphi; ruler: true;}var
 i: integer;
 Res: TDragResult;
 CurrentImage : TProcessedImage;
 ThreadCallsArray : array of IAsyncCall;
 WaitResult : Integer;
begin
 if (ActiveImages.Count = 1)
 then begin
 /// one image only
 try
 CurrentImage := TProcessedImage(ActiveImages[0]);
 except
 CurrentImage := nil;
 end;
 if Assigned(CurrentImage)
 then CurrentImage.CreateDestFile;
 end
 else begin
 /// multiple images -> multi-threaded processing
 SetLength(ThreadCallsArray, ActiveImages.Count);
 for i := 0 to Pred(ActiveImages.Count) do
 begin
 try
 CurrentImage := TProcessedImage(ActiveImages[i]);
 except

```
CurrentImage := nil;
end;
if Assigned(CurrentImage)
then begin
ThreadCallsArray[i] := AsyncCall(@CreateDestFileOfImage, CurrentImage);
end;
end;
WaitResult := AsyncMultiSync(ThreadCallsArray, True, INFINITE);
end;{/CODE}
```

Please note that if there is only one image in the list, it is better to just process that image in the main VCL thread, as spanning a different thread and then waiting on that thread's completion would be a waste of resources.

Remember to store the interface returned by AsyncCall, as dropping it, setting it to NULL, or overwriting it with another call to AsyncCall (for example, in a loop that creates multiple threads) would force the thread to be executed in the main VCL thread, canceling the benefits we are trying to achieve by using multiple worker threads.

Before switching to a multi-threaded solution, be aware that you should check if the code is really thread-safe (if the code inside a thread tries to access the VCL, it's a recipe for troubles, as only the main VCL thread can work with it) and also check if third-party libraries are thread-safe of not: e.g. the JPEG compression library used by AltaPixShare is not thread-safe, so you get corrupted output images if you try to compress multiple JPEG files at the same time, so the JPEG function is called within a critical section that serializes access to that resource:

```
{CODE brush: delphi; ruler: true;}JPEGLibAccess.Enter;
try
SaveTo24bitJPEGFile(OutputBitmap, DestFilename, 90, false);
finally
JPEGLibAccess.Leave;
end;{/CODE}
```

A new article showing how to parallelize loops in Delphi has been published at this address .