DESIGN, AUTOMATION & TEST IN EUROPE
DATE 18
19 – 23 March, 2018 · ICC · Dresden · Germany
The European Event for Electronic System Design & Test

GEPURA QUASAR

QUASAR
A High-level Programming Language and Development Environment for Designing Smart Vision Systems on Embedded Platforms

Bart Goossens, Hiep Luong, Jan Aelterman and Wilfried Philips

GHENT UNIVERSITY
imec

The benefits of GPUs. Growing application domain

See: https://blogs.nvidia.com/blog/2017/05/24/ai-revolution-eating-software

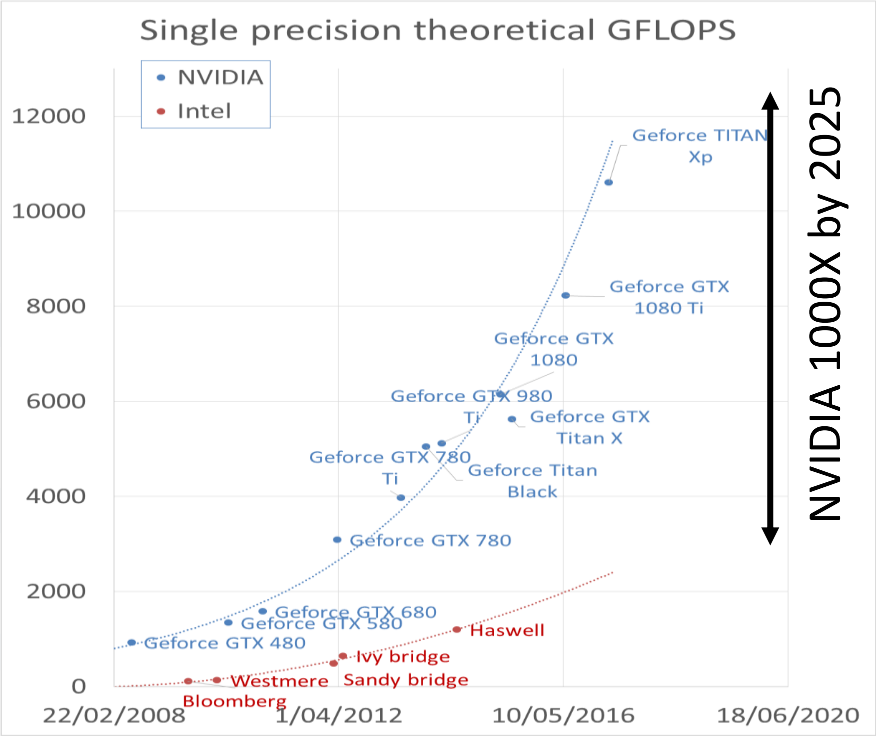
Exploits massive parallelism
- e.g. to process large amounts of data in parallel

Speed-ups of 10 to 100

Energy efficient
- calculations/Watt

Applied to many applications:
- Multimedia, finances, big data, scientific computing,…

Commodity HW
- Standard in desktops and laptops; embedded platforms



The drawbacks of GPU programming solved by Quasar

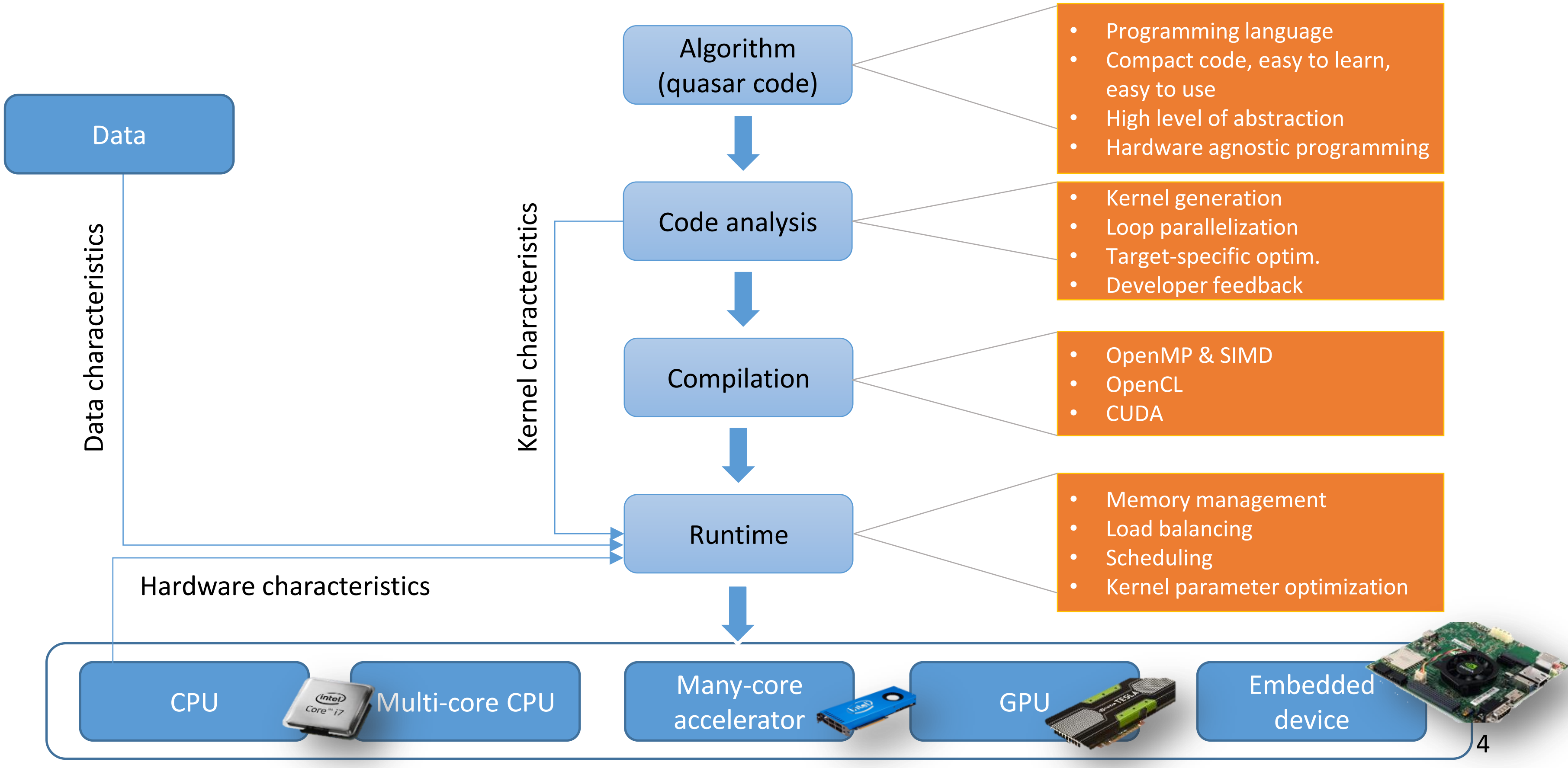
Low level coding experts needed

Strong coupling between algorithm development and implementation

Long development lead times

Each HW platform requires new optimizations

Quasar: overview



Quasar – High-level Scripting language

- Same abstraction level as Python and Matlab

beam_former ▸ beamformer_vectors Zoom: 130% Line 117 col 9

```
    P_array = transpose([arrayGeom_x, arrayGeom_y, arrayGeom_z])
    P_array_matrix = repmat(P_array,[1,1,numAz])

    v = complex(zeros(numEls,numAz,numElev,numFreqs))

    pointing_vectors = zeros(3, numAz)
    for ielev = 0..numElev-1
        pointing_vectors[0,:] = cos(az*pi/180)*cos(el[ielev]*pi/180)
        pointing_vectors[1,:] = sin(az*pi/180)*cos(el[ielev]*pi/180)
        pointing_vectors[2,:] = sin(el[ielev]*pi/180)
        focus_points = pointing_vectors * diagm(focus_range)
        focus_points_matrix : cube = reshape(kron(focus_points,ones(numEls,1)), _
                                 [numEls,3,numAz])
        delta_range = sqrt((reshape(sum((P_array_matrix - _
                    focus_points_matrix).^2,1), [numEls, numAz]))) - ones(numEls,1)*focus_range
        for ifrq = 0..numFreqs-1
            freq = freqs[ifrq]
            v[:,:,ielev,ifrq] = exp(1j*2*pi*delta_range*freq/soundSpeed)
        endfor
    endfor
```

Matrix slices!

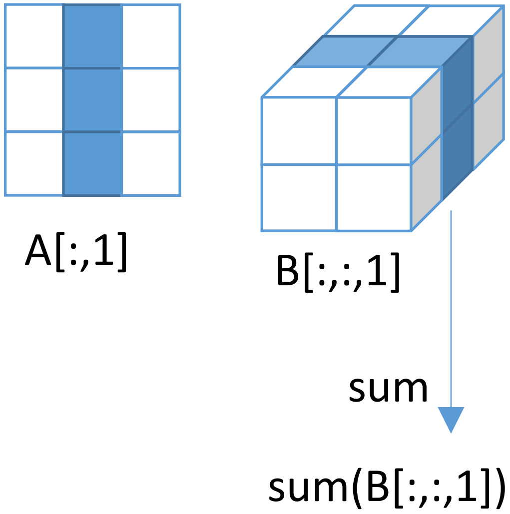
Matrix multiplication

Heterogeneous compilation: code parallelized and optimized for CPU and GPU!

Quasar – High-level to low-level translation

- Flexible algorithmic development with optimized CPU/GPU code

- High-level array programming:
  - manipulating vectors and matrices
  - concise expressions (easy to write, easy to understand)
  - no special compiler support ⇒ performance cost (poor data locality)!

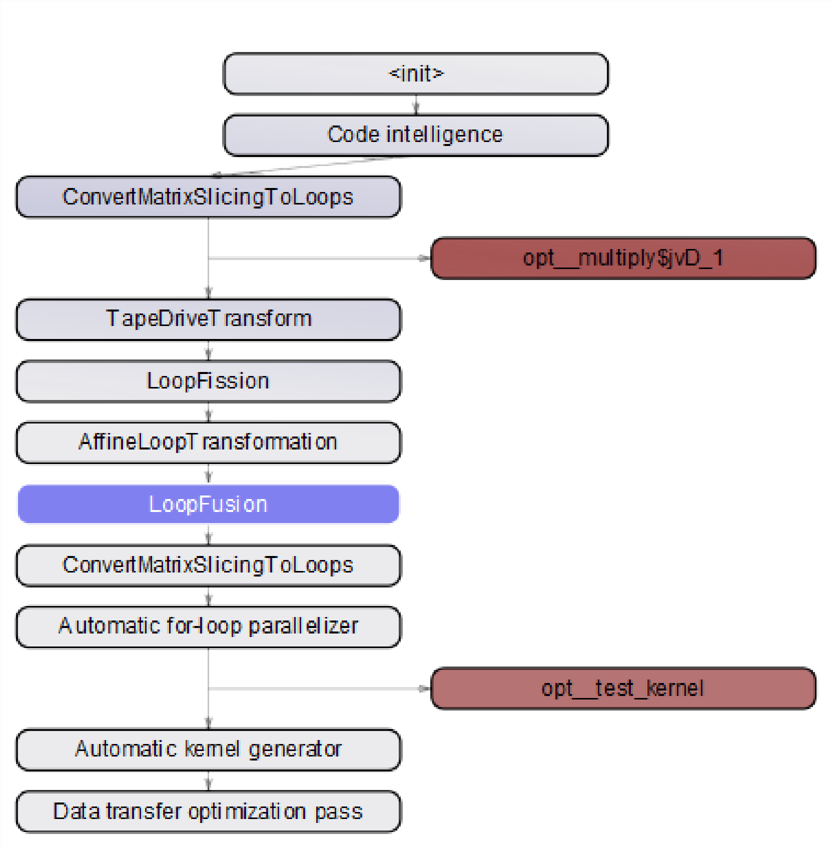


- Quasar code analysis:
  - Automatic parallelization
  - Aggregation operations (sum,mean,prod,min,max) ⇒ parallel reduction
  - Cumulative operations (cumsum, cummin, ...) ⇒ parallel prefix sum
  - Vector slices, matrix slices, vector arithmetic ⇒ automatic kernel generation

Quasar – High-level to low-level translation

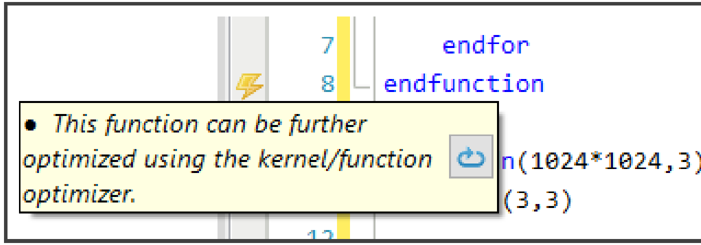
- Compiler intermediate results can be visualized

Optimization pipeline



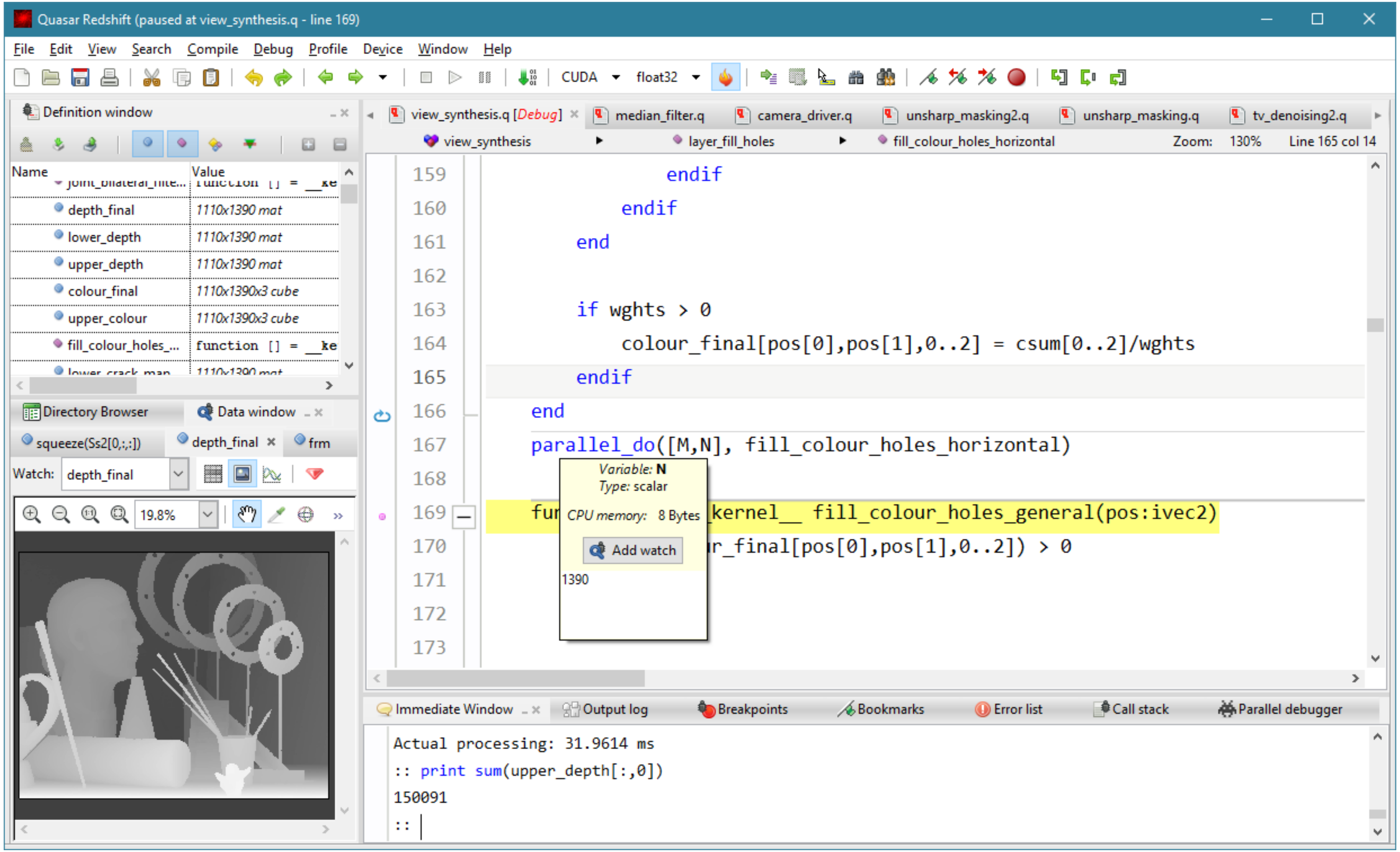
Code preview

```
  $out00=zeros((((size(x,0)-1)-0)+1))
  A=ones(3,3)
  for m=0..(size(x,0)-1)
      p=x[m,(0..2)]
      $out00[m]=0.
- endfor
- for m2=0...(size(x,0)-1)
      for $k002=0...(size(A,0)-1)
          for $k10=0..(size(A,1)-1)
-             $out00[m2]+=(p[$k002]*(A[$k002,$k10]+1)*(p[$k10]+2))
+             $out00[m]+=(p[$k002]*(A[$k002,$k10]+1)*(p[$k10]+2))
          endfor
      endfor
- endfor
- for m=0..(size(x,0)-1)
      y[m]=$out00[m]
  endfor
  return
```



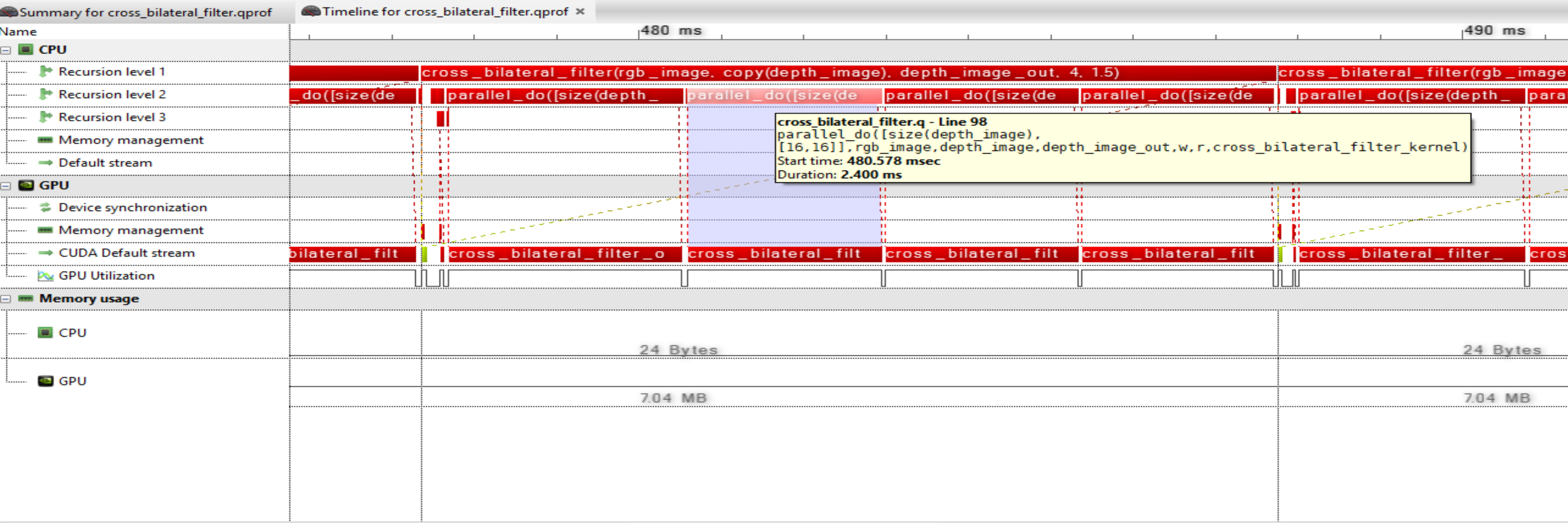
Quasar – Redshift IDE

- Code editing, debugging, variable inspection, immediate evaluation



Quasar – Redshift IDE

- Profiler: analyze CPU and GPU activity (bottleneck kernels, timeline)

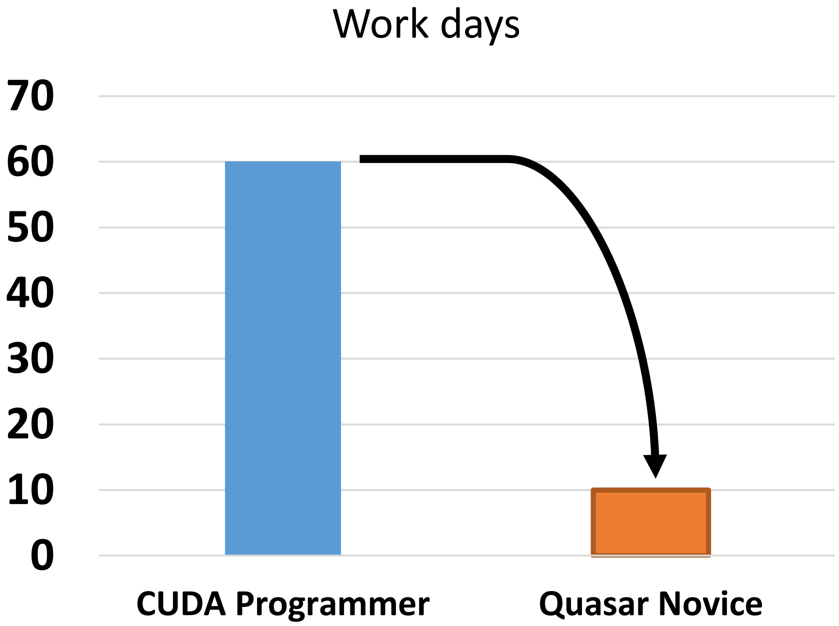


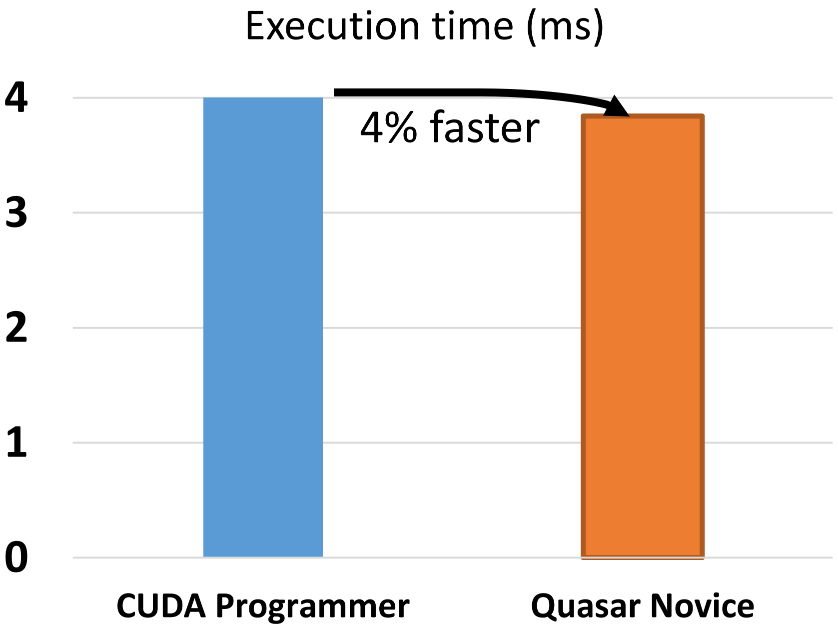
Summary for cross_bilateral_filter.qprof | Timeline for cross_bilateral_filter.qprof | Kernels for cross_bilateral_filter.qprof

Priority	Kernel	Module	Device	Launches	Total time	Max duration	Threads/block	Grid size	Block size	Occupancy	Bottleneck	Block limit	Registers
1	cross_bilateral_filter_forloop_op	cross_bilateral_filter	GPU	40	199.543 ms	5.753 ms	512	40x15x1	16x32x1	100.0%			22
2	cross_bilateral_filter_opt__cross_	cross_bilateral_filter	GPU	40	103.848 ms	4.671 ms	256	40x30x1	16x16x1	75.0%	Block size	Registers	40
3	min1	$CORELIB	GPU	5	2.352 ms	534.9 us	512	64x1x1	512x1x1	100.0%			7
4	multiplyscalar	$CORELIB	GPU	15	1.463 ms	125.5 us	512	225x8x1	4x128x1	100.0%			6
5	cube_slice_put_scalar_scalar	$CORELIB	GPU	9	912.0 us	109.1 us	512	15x40x1	32x16x1	100.0%			13
6	powscalar	$CORELIB	GPU	5	820.2 us	231.8 us	512	225x8x1	4x128x1	100.0%			15
7	opt__exp_opt__exp_kernel	cross_bilateral_filter	GPU	10	813.1 us	119.0 us	512	4x1x1	512x1x1	100.0%			13

Quasar – test case results

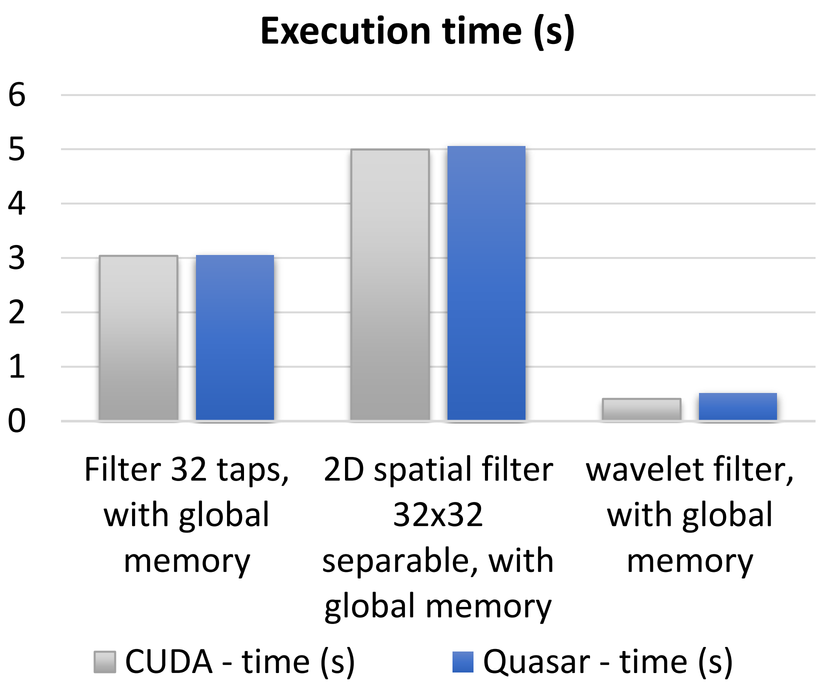
Test case: Parallel MRI reconstruction algorithm by experienced CUDA programmer and Quasar novice

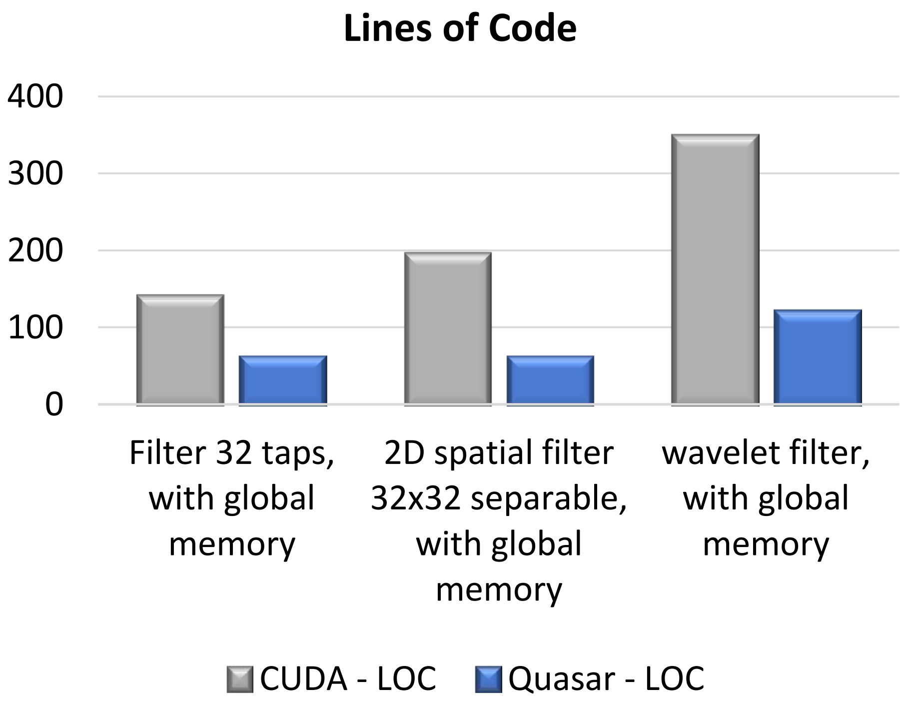




Future proof: program once, run everywhere

Quasar – benchmark results





Quasar - demo

Video link: https://www.youtube.com/watch?v=IEwrMHXgyoU

Conclusion

- GPUs: performance revolution
- High-level code: gives compiler more flexibility for code generation + optimization
- Targetting heterogeneous systems (GPUs)
- Quasar: new programming paradigm for heterogeneous systems (CPU, GPU...)
  - Low barrier of entry (Matlab-like)
  - Reduces development time
  - Future proof!
  - Redshift IDE: powerful code editing, debugging, profiling!
- Available for tryout: http://gepura.io/quasar/try-quasar/

- **Thank you for your attention!**
- **Questions?**

Try-out Quasar: http://gepura.io/quasar/try-quasar/