

**DATE** 18

DESIGN, AUTOMATION & TEST IN EUROPE

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**



**umec**

# 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

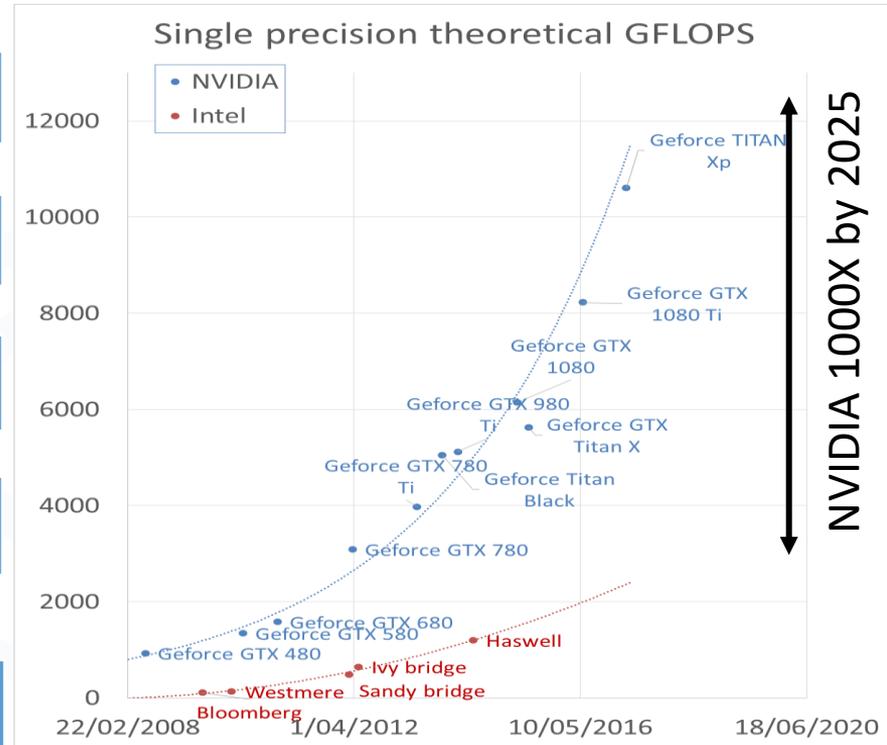
- calculations/Watt

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

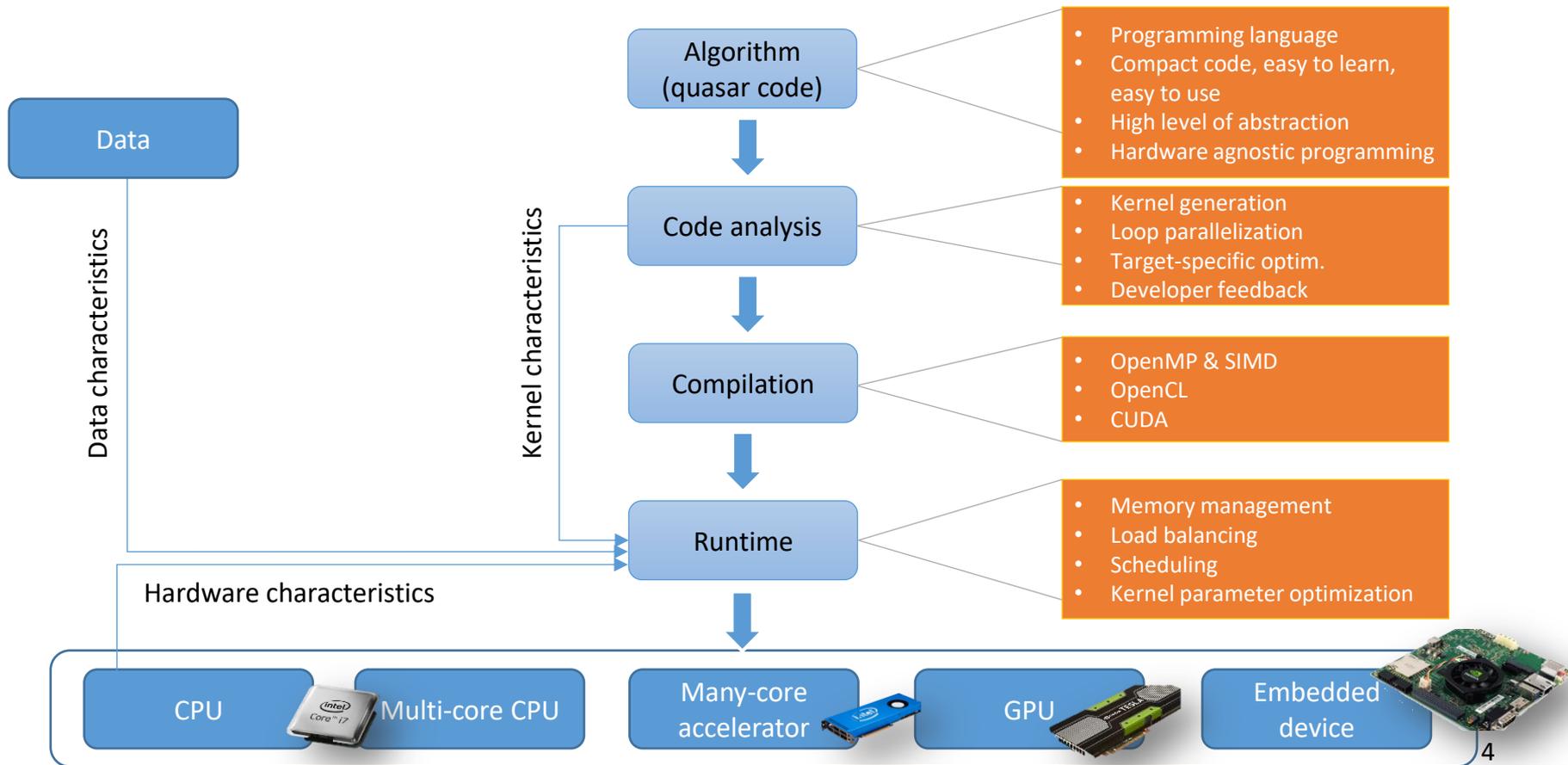
Strong coupling between algorithm development and implementation

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
110  P_array = transpose([arrayGeom_x, arrayGeom_y, arrayGeom_z])
111  P_array_matrix = repmat(P_array,[1,1,numAz])
112
113  v = complex(zeros(numEls,numAz,numElev,numFreqs))
114
115  pointing_vectors = zeros(3, numAz)
116  for ielev = 0..numElev-1
117      pointing_vectors[0,:] = cos(az*pi/180)*cos(el[ielev]*pi/180)
118      pointing_vectors[1,:] = sin(az*pi/180)*cos(el[ielev]*pi/180)
119      pointing_vectors[2,:] = sin(el[ielev]*pi/180)
120      focus_points = pointing_vectors * diagm(focus_range)
121      focus_points_matrix : cube = reshape(kron(focus_points,ones(numEls,1)), _
122          [numEls,3,numAz])
123      delta_range = sqrt((reshape(sum((P_array_matrix - _
124          focus_points_matrix).^2,1), [numEls, numAz]))) - ones(numEls,1)*focus_range
125      for ifrq = 0..numFreqs-1
126          freq = freqs[ifrq]
127          v[:, :, ielev, ifrq] = exp(1j*2*pi*delta_range*freq/soundSpeed)
128      endfor
129  endfor
```

Matrix slices!

Matrix multiplication

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

# Quasar – High-level to low-level translation

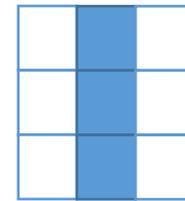
- Flexible algorithmic development with optimized CPU/GPU code

- High-level array programming:

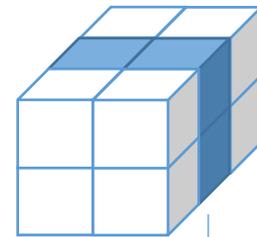
- manipulating vectors and matrices
- concise expressions (easy to write, easy to understand)
- no special compiler support  $\Rightarrow$  performance cost (poor data locality)!

- Quasar code analysis:

- Automatic parallelization
- Aggregation operations (sum,mean,prod,min,max)  $\Rightarrow$  parallel reduction
- Cumulative operations (cumsum, cummin, ...)  $\Rightarrow$  parallel prefix sum
- Vector slices, matrix slices, vector arithmetic  $\Rightarrow$  automatic kernel generation



$A[:,1]$



$B[:, :, 1]$

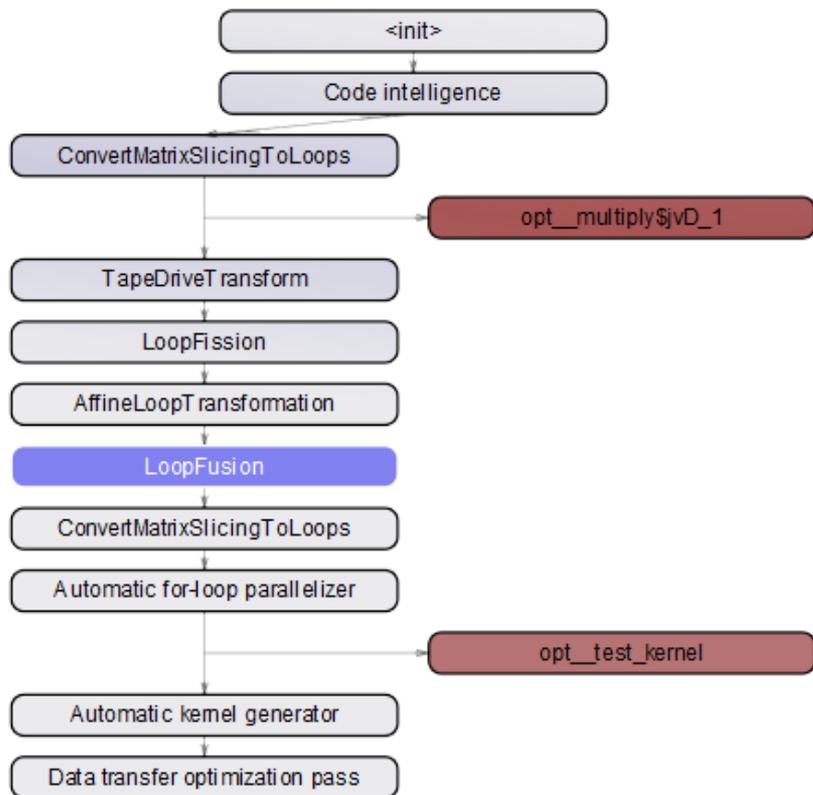
sum

$\sum(B[:, :, 1])$

# Quasar – High-level to low-level translation

- Compiler intermediate results can be visualized

## Optimization pipeline



## Code preview

```
1   $out00=zeros((((size(x,0)-1)-0)+1))
2   A=ones(3,3)
3   for m=0..(size(x,0)-1)
4     p=x[m,(0..2)]
5     $out00[m]=0.
6   -   endfor
7   -   for m2=0...(size(x,0)-1)
8     for $k002=0...(size(A,0)-1)
9       for $k10=0...(size(A,1)-1)
10      -   $out00[m2]+=(p[$k002]*(A[$k002,$k10]+1)*(p[$k10]+2))
11      +   $out00[m]+=(p[$k002]*(A[$k002,$k10]+1)*(p[$k10]+2))
12      endfor
13    endfor
14  -   endfor
15  -   for m=0..(size(x,0)-1)
16    y[m]=$out00[m]
17  endfor
18  return
```

7 endfor  
8 endfunction

• This function can be further optimized using the kernel/function optimizer.

in(1024\*1024,3)  
(3,3)

7

# Quasar – Redshift IDE

- Code editing, debugging, variable inspection, immediate evaluation

The screenshot displays the Quasar Redshift IDE interface. The main window shows a code editor with a C++ kernel snippet. The code includes conditional logic for filling holes in an image, with a loop over positions  $(M, N)$  and a call to a kernel function `kernel__fill_colour_holes_general`. A tooltip is visible over the kernel call, showing the variable `N` is a scalar with a CPU memory of 8 Bytes.

The left sidebar contains a 'Definition window' with a table of variables:

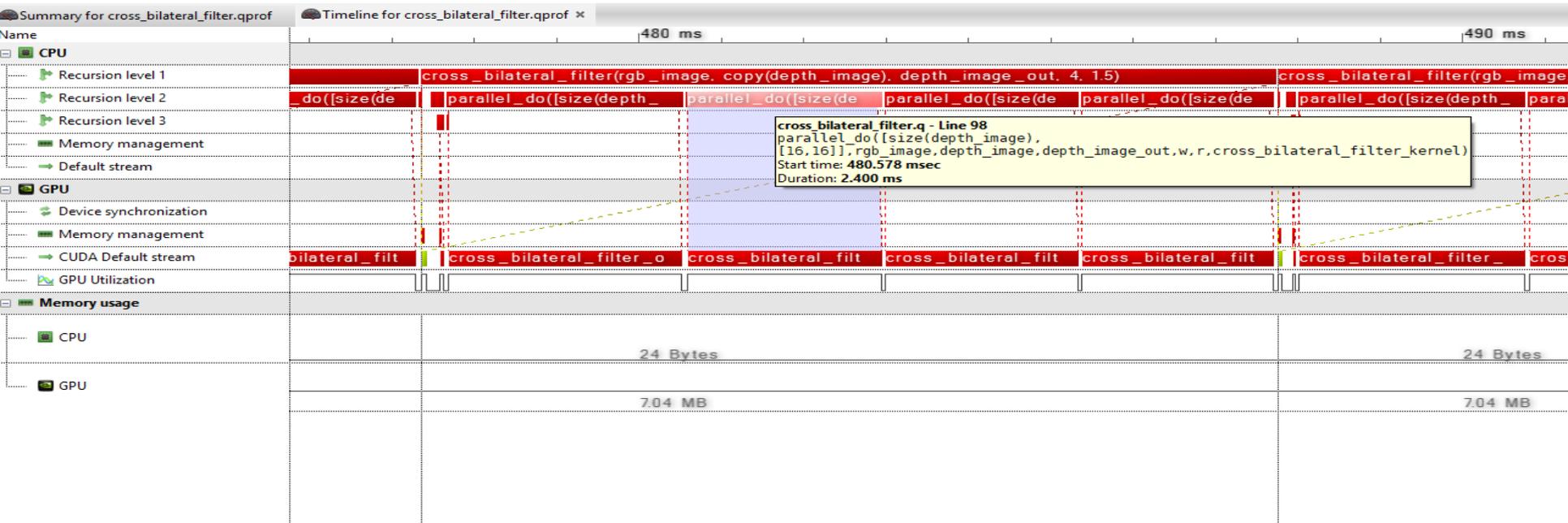
Name	Value
<code>depth_final</code>	<code>1110x1390 mat</code>
<code>lower_depth</code>	<code>1110x1390 mat</code>
<code>upper_depth</code>	<code>1110x1390 mat</code>
<code>colour_final</code>	<code>1110x1390x3 cube</code>
<code>upper_colour</code>	<code>1110x1390x3 cube</code>
<code>fill_colour_holes_...</code>	<code>function [] = __ke</code>
<code>lower_crack_ma...</code>	<code>1110x1390 mat</code>

Below the definition window is a 'Directory Browser' and a 'Data window' showing a watch list with `depth_final` selected. The bottom of the IDE features an 'Immediate Window' with the following output:

```
Actual processing: 31.9614 ms
:: print sum(upper_depth[:,0])
150091
:: |
```

# Quasar – Redshift IDE

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

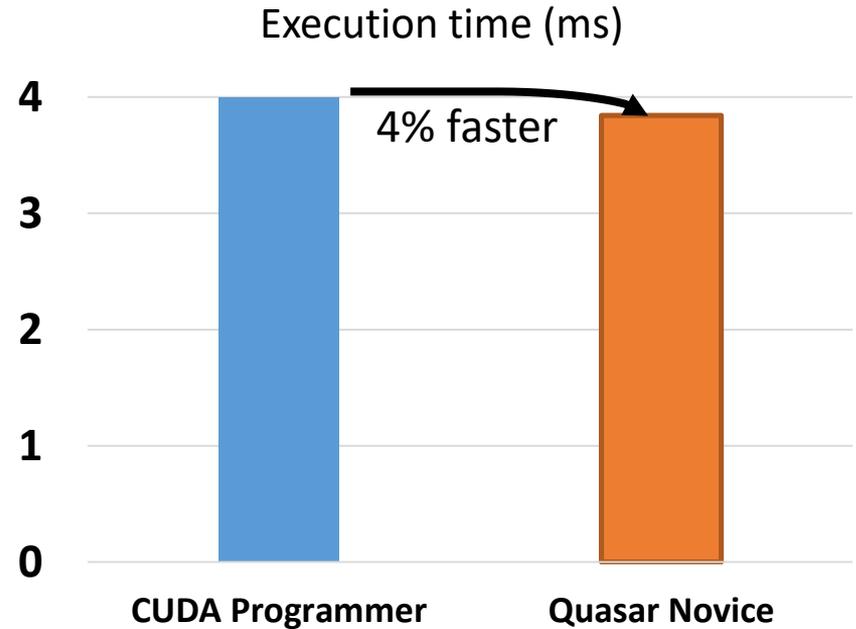
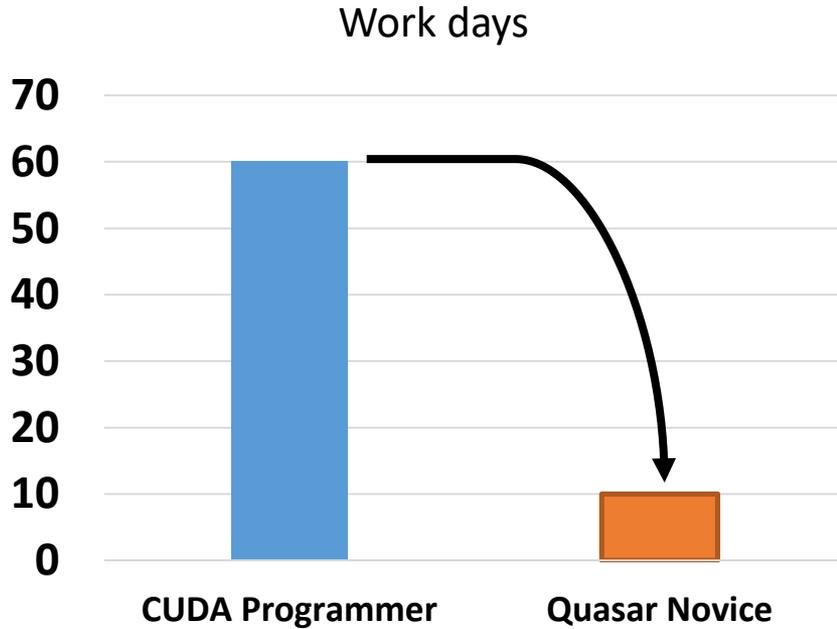


Summary for cross\_bilateral\_filter.qprof | Timeline for cross\_bilateral\_filter.qprof | Kernels for cross\_bilateral\_filter.qprof ×

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_opt	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

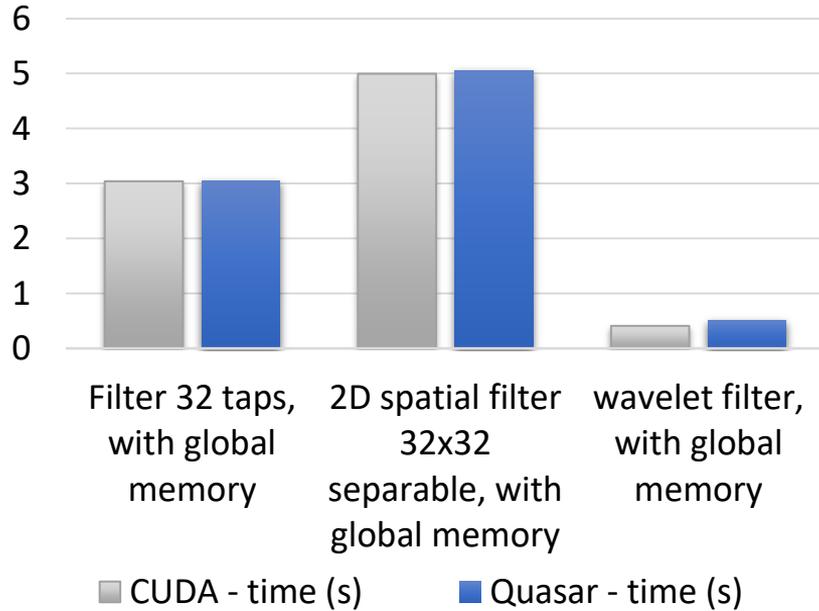
Test case: Parallel MRI reconstruction algorithm by **experienced CUDA programmer** and **Quasar novice**



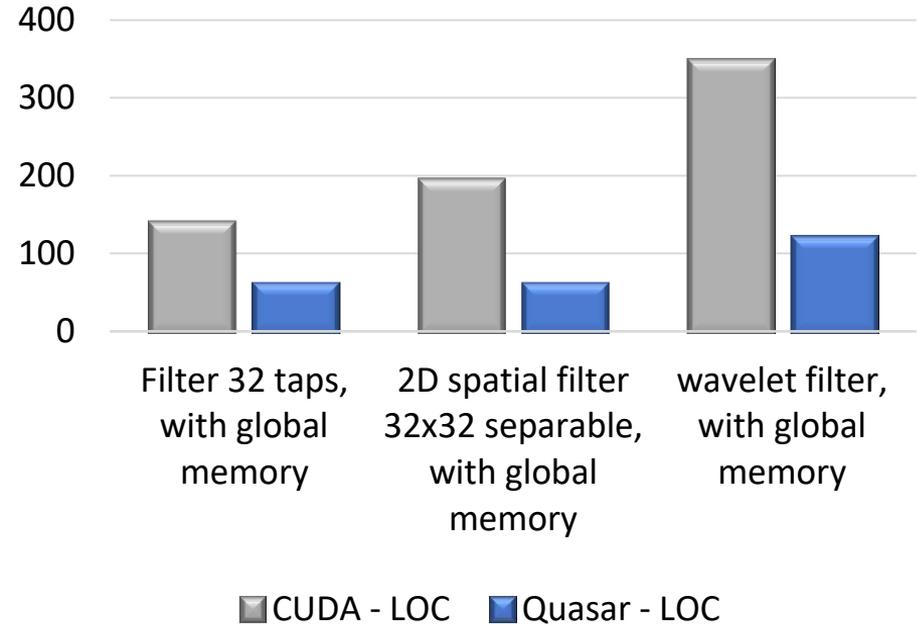
Future proof: program once, run everywhere

# Quasar – benchmark results

## Execution time (s)



## Lines of Code



# 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/>