

The GPU conondrum for rapid prototyping

For research, an ideal programming language should enable **fast development** with **compact**, yet readable **code**. However, with the increasing complexity of state of the art algorithms, **fast execution** becomes paramount as well. GPU acceleration would be a major asset. Yet researchers often refrain from GPU acceleration due to the use of low-level programming languages and the perception of a steep learning curve.

The tools

➤ A full-blown IDE:

Main target platform

GPU & CPU info

Real-time image viewers

Code editor with auto completion windows and help tooltips

Debugging windows: call stack, breakpoints

Data inspection and watches

Interactive command window

➤ Performance and profile analysis:

Kernel overview	
Total CPU Kernel time:	Total GPU Kernel time:
42.239 ms	36.687 ms
CPU Kernel calls:	GPU Kernel calls:
11	13
Avg CPU Kernel time:	Avg GPU Kernel time:
3.840 ms	2.822 ms
Max CPU Kernel time:	Max GPU Kernel time:
27.512 ms	17.297 ms
CPU bottleneck kernel:	GPU bottleneck kernel:
sinc_cube_anonymous10	opt_main_opt_main_kernel_p

• Bottle neck detection

• CPU/GPU usage

➤ Code analysis and feedback:

Warning: missed optimization opportunity of im[(x-smx)..(x+smx)].(y-smx)..(y+smx)]. reason: '(x-smx)' has the wrong type '??', expected 'int const' (templateMatching.q line 9)

Missed opportunity for optimization for (im:mat,\$t1,\$t2,\$t3,\$t4) -> im[(t1..(t2), (t3..(t4))]: could not determine the type of the variables 't1', 't2', 't3', 't4'.

The Quasar Value proposition

Quasar is a new **programming language** that:

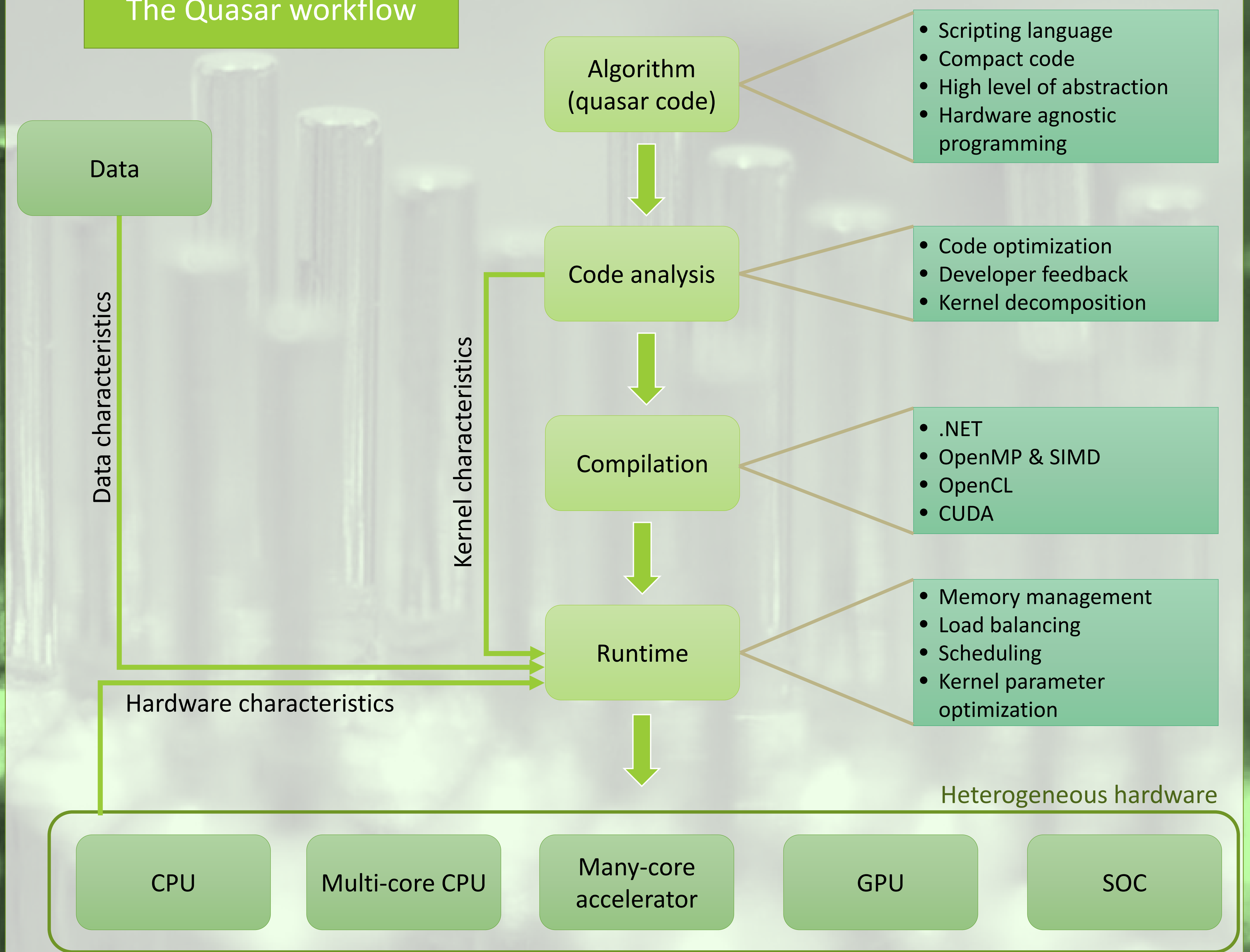
- Has a **low barrier** of entry in **GPU acceleration**
- Results in a single (**HW-agnostic**) code base
 - Ideal for sharing code
- Results in **shorter development** cycles
- Allows researchers to focus on algorithms, not on implementation
- Has differentiating **development tools**
 - e.g. interactive debugging

Check



<http://quasar.ugent.be>

The Quasar workflow



An example

Algorithm 1: Pseudo code for primal dual optimization of active contour segmentation.

```
1  $p_{k+1}[t] = \frac{p_k[t] + \delta t \nabla^T (\nabla^T p_k[t] - \theta v_k[t])}{1 + \delta t |\nabla^T (\nabla^T p_k[t] - \theta v_k[t])|}$ 
2  $u_{k+1}[t] = v_k[t] - \frac{1}{\theta} \nabla^T p[t]$ 
3  $v_{k+1} = \min \left( \max \left( u_{k+1} - \frac{\lambda}{\theta} c, 0 \right), 1 \right)$ 
```

Straightforward mapping to Quasar code

```
...iterative optimization using primal-dual
for cnt=1..maxIter
    ...
    %update u
    term = divp(p) - v*theta
    gr = gradm(term)
    norm = sqrt(gr[:, :, 0].^2 + gr[:, :, 1].^2)
    denom = 1 + dt*norm
    nom = p + dt*gr
    p[:, :, 0] = nom[:, :, 0] ./ denom
    p[:, :, 1] = nom[:, :, 1] ./ denom
    u = v - divp(p) / theta
    ...
    %update v
    v = u - lambda*c/theta
    v = max(v, 0)
    v = min(v, 1)
end
%binarise final solution
u = u > 0.5
end

function dp = divp(p:mat'circular)
    dp = uninit(size(p))
    sp = size(p)
    for cnt_x=0..sp[0]-1
        for cnt_y=0..sp[1]-1
            px = p[cnt_x, cnt_y] - p[cnt_x-1, cnt_y]
            py = p[cnt_x, cnt_y] - p[cnt_x, cnt_y-1]
            dp[cnt_x, cnt_y] = px + py
        end
    end
end
```

Matrix syntax

Boundary handling

Results



1

Fast development

- 2 weeks vs. 3 months for a CUDA implementation of an MRI reconstruction algorithm

2

Fast execution using the GPU

- 64 fps vs 2,91 fps for a template matching algorithm

3

Efficient code:

- 300 lines of Quasar code vs. 2700 lines of C++ code for a registration algorithm

R
A
P
I
D

P
R
O
T
O
T
Y
P
I
N
G