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.

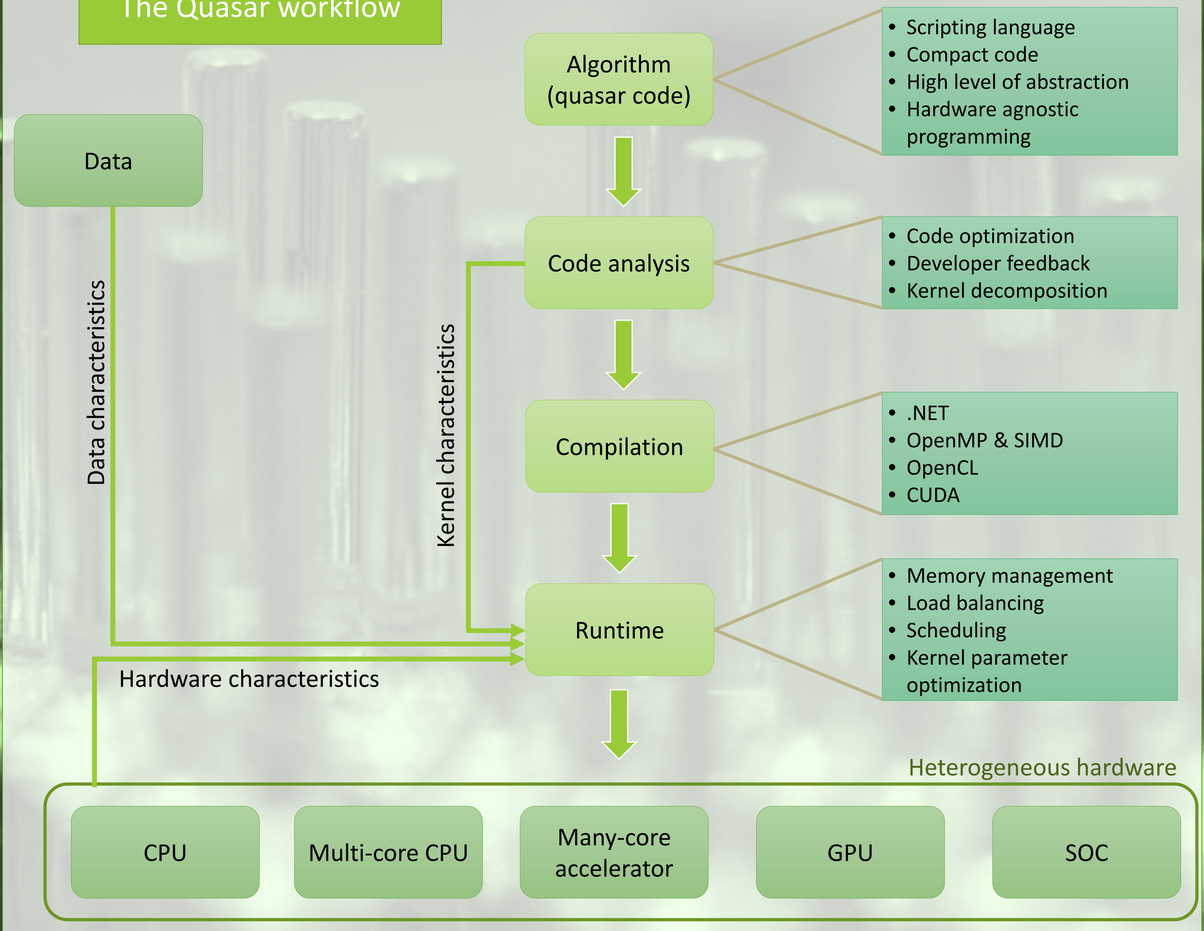
Yet researchers often refrain from GPU acceleration due to the use of low-level programming languages and the perception of a steep learning curve.

The Quasar Value proposition

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

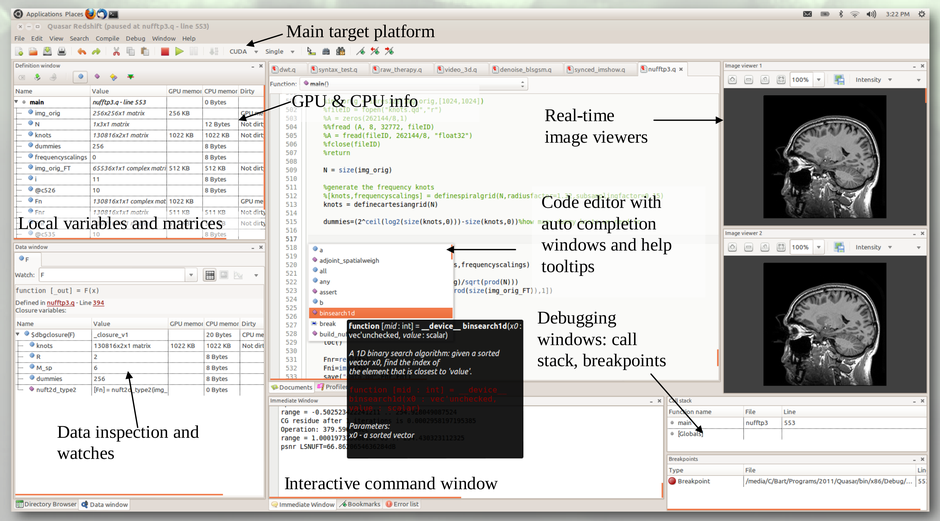
- Has a **low barrier** of entry in **GPU acceleration**
- Results in a single (**HW-agnostic**) code base
  - Ideal for sharing code
- Results in **shorter development** cycles
- Allows researchers to focus on algorithms, not on implementation
- Has differentiating **development tools**
  - e.g. interactive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

The Quasar workflow

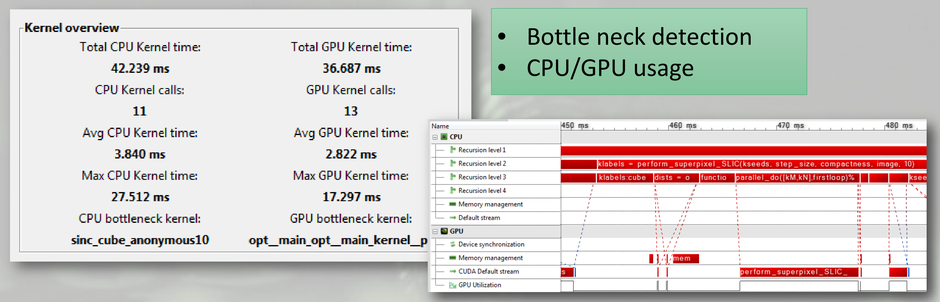


The tools

- A full-blown IDE:

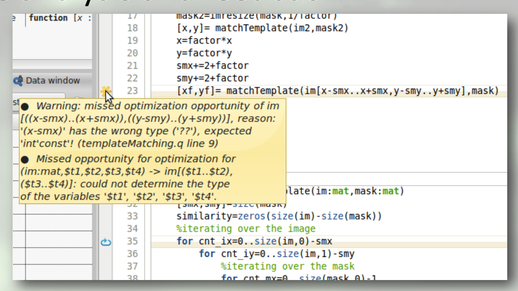


- Performance and profile analysis:



- Bottle neck detection
- CPU/GPU usage

- Code analysis and feedback:



Check

http://quasar.ugent.be

An example

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

$$\mathbf{p_{k+1}}[\mathbf{t}] = \frac{\mathbf{p_k}[\mathbf{t}]+\delta t\ \nabla^-(\nabla^+\mathbf{p_k}[\mathbf{t}]-\theta\mathbf{v_k}[t])}{1+\delta t\ |\nabla^-(\nabla^+\mathbf{p_k}[\mathbf{t}]-\theta\mathbf{v_k}[t])|}$$

$$\mathbf{u_{k+1}}[\mathbf{t}] = \mathbf{v_k}[t] - \frac{1}{\theta}\nabla^+\mathbf{p}[t]$$

$$\mathbf{v_{k+1}} = \min\left(\max\left(\mathbf{u_{k+1}} - \frac{\lambda}{\theta}\mathbf{c}, 0\right), 1\right)$$

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

RAPID PROTOTYPING