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
The benefits of GPUs. Growing application domain

- 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

See: https://blogs.nvidia.com/blog/2017/05/24/ai-revolution-eating-software
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 addresses these challenges by providing a unified framework for GPU programming, reducing the need for low-level coding expertise, decoupling algorithm development from implementation, shortening development lead times, and offering a common platform for optimizations.
Quasar: overview

Algorithm (quasar code)

Code analysis

Compilation

Runtime

Data

Data characteristics

Kernel characteristics

Hardware characteristics

CPU

Multi-core CPU

Many-core accelerator

GPU

Embedded device

• Programming language
• Compact code, easy to learn, easy to use
• High level of abstraction
• Hardware agnostic programming

• Kernel generation
• Loop parallelization
• Target-specific optim.
• Developer feedback

• OpenMP & SIMD
• OpenCL
• CUDA

• Memory management
• Load balancing
• Scheduling
• Kernel parameter optimization

Hardware characteristics

Kernel characteristics

Data characteristics
Quasar – High-level Scripting language

- Same abstraction level as Python and Matlab

```
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, 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

\[
\begin{align*}
A[\cdot,1] & \\
B[\cdot;\cdot,1] & \\
\text{sum} & \\
\text{sum}(B[\cdot;\cdot,1]) & 
\end{align*}
\]
Quasar – High-level to low-level translation
• Compiler intermediate results can be visualized

Optimization pipeline

- <init>
- Code intelligence
- ConvertMatrixSlicingToLoops
- TapeDriveTransform
- LoopFission
- AffineLoopTransformation
- LoopFusion
- ConvertMatrixSlicingToLoops
- Automatic for-loop parallelizer
- Automatic kernel generator
- Data transfer optimization pass
- opt__multiply$\_vD\_1

Code preview

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

This function can be further optimized using the kernel/function optimizer.
Quasar – Redshift IDE

- Code editing, debugging, variable inspection, immediate evaluation
Quasar – Redshift IDE

- Profiler: analyze CPU and GPU activity (bottleneck kernels, timeline)
Quasar – test case results

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

Work days

Execution time (ms)

CUDA Programmer Quasar Novice

4% faster

Future proof: program once, run everywhere
Quasar – benchmark results

**Execution time (s)**

- Filter 32 taps, with global memory
- 2D spatial filter 32x32 separable, with global memory
- Wavelet filter, with global memory

**Lines of Code**

- Filter 32 taps, with global memory
- 2D spatial filter 32x32 separable, with global memory
- Wavelet filter, with global memory
Video link: https://www.youtube.com/watch?v=IEwrMHXgyoU
• **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!

Thank you for your attention!

Questions?

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