Quasar Release Notes

This text file lists the major changes in the different versions of Quasar.

22/5/2023

- Maintenance release

1/2/2022

- Maintenance release
- Added CUDA 11.5/11.6 support
- Added support for Visual Studio 2022
- Improved performance of the syncgrid() function in quasar.h
- Fixes related to multiple Python environments

12/7/2021

- Maintenance release
- Added CUDA 11.3/11.4 support

30/3/2021

- Maintenance release
- Added CUDA 11.0/11.1/11.2 support
- Added gpu_feature(“cuda_arch”) to detect the CUDA architecture

5/10/2020

- Compiler: Improved several of the built-in kernel transformations
- Compiler: Compile-time optimization of the GPU block dimensions based on symbolic calculation of the amount of shared memory used by a kernel function
- Runtime: extended the plot/scatter widgets
  - added zoom function
  - added export to .csv
  - added mouse and keyboard handlers
  - added vector layers for drawing on top of existing plots
- Redshift:
  - Directory browser pad redesign (now workspace browser)
  - Watch pad: several improvements for visualizing N-D arrays
  - Watch pad: allow editing objects and matrices on the fly

- Redshift Profiler:
  - improved profiling of CPU operations (function calls)
  - added profiling of CUDA API calls (cuPTI mode only)
    - improved GPU event viewer
    - added occupancy as function of the number of blocks
    - added synchronization/CUDA API call overview to profiler dashboard
    - added allocation type (analysis of pinned host memory transfers)
  - the profiler now gives optimization guidelines
  - performance optimizations for the memory profiler

7/7/2020: profiler upgrade

- Added support for NVIDIA PerfWorks profiling framework (to obtain kernel metrics on Turing and newer GPUs)
- Improved memory throughput visualization (kernel profiler)
- Added support for custom kernel profiling metrics
- Added ability to visualize SASS assembly code from within Redshift
- Added instruction-level SASS profiling & visualization
- Improved mixed-precision support for cuDNN tensor-core based convolutions

10/6/2020

- Give warning in kernel functions for detection of divisions with the same divisor
- Added division approximation functions rcp_rn, rcp_ru, rcp_rz and div_approx.
- Added sincos function (simultaneous calculation of sin and cos with the same argument).
- CUDA FFT max plan cache size is now configurable through option RUNTIME_FFT_PLAN_MAXCACHESIZE in Quasar.Runtime.config.xml.
- Quasar.CompMath.dll: added $hessian, $diffrmul and $functomat
- Redshift: added welcome screen with templates
- Redshift: allow placing breakpoints in autogenerated code
- Redshift: added support for Python (execution, environment selection)
- Redshift: added out-of-process and remote debugging
- Redshift: added tooltip for current debug location (containing call stack and indicating whether the location is inside autogenerated code)
- Runtime: various remoting enhancements, allowing distributed Quasar applications to be developed
- Runtime: added multi-threading and asynchronous functions async_func
- Compiler: parallel reduction transform, parallel dimension reduction transform, kernel tiling transform can now be disabled from the code through {! kernel_transform disable ="XX"} with XX= parallel_reduction / parallel_dimension_reduction / kernel_tiling .

31/1/2020

- Maintenance fixes
- Added CUDA 10.2 support
- quasar_dsl.h: added QValue::WriteHostVariable
- quasar_host.h: added IQuasarHost::GetRawSizeOfType
- SIMD processing improvements: native C++/CUDA code generator support for vecuint8 types
- SIMD processing: improved half precision floating point support (see floattypes.h)
- SIMD processing: improved hybrid mode kernel tiling algorithm
- plot/scatter: give error when too many (>128) series are plotted
- Added typename() function for inline placing of Quasar type names. In the past, a type alias was required

  ```
  type cube4 : cube{4}
  data = cube4(2,3,4,5)
  ```

  Becomes data = typename(cube{4})(2,3,4,5)
- Windows: improvement for using Redshift in a remote desktop connection
- Improved the loop invariant code motion compiler optimization
- Improved compiler optimizations for algoritmically differentiated code
- Improved compiler optimizations for algoritmically adjugated code
- Inlining of differentiated functions using $diff$(inline(func)(x), x) will expand into $inline(func_deriv).
- Warp shuffling as alternative strategy for parallel reduction in shared memory pressured kernels.
- Linux: fix for mono 6.6.0
- qdnn_conv_descriptor: renamed several parameters
- CUDA backend: support warp shuffling on fixed-length vector types
- C++ backend: C++ standard used for code generation is now configurable (Quasar.config.xml)
- Added setting “CUDA_BACKEND_COOPERATIVEGROUPS_TCCDRIVER” to generate code taking advantage of GPU drivers running in Tesla Compute Cluster (TCC) mode

27/11/2019

- Fixed issue with CUDA error reporting during profiling start
- Enable kernel fusion in algorithmically differentiated code using {!ad_support kernel_fusion=true}
- Improved kernel fusion algorithm

- Improved HLC++ backend
- Fix for FixContainerLeak workaround with Gtk versions >2.12.16
- char16_string=.c: added stscanf function

25/10/2019

- Maintenance fixes
- Fix for OpenGL gray screen issue in Windows
- Workaround to run in Mono 6.4
- Upgrade of fast_bilateral_filter.q and immedfilt.q
- boxfilter in imfilter.q: added optional normalization parameter
- C++ host API in linux: switched to wchar_t (32-bit) as the main character type
- printf() no longer prints line ending \n at the end of a line (printf("Hello") becomes printf("Hello\n"))

17/9/2019

- Maintenance fixes
- Optimizations for pinned host memory transfers (faster transfer from CPU to GPU)
- Show native modules in About Dialog in Redshift (windows only)
- Detection of sticky CUDA errors
- imshow: improved handling of matrices with singleton dimensions
- Fixes in reference counting when using the external C++ interface
- External C++ interface: updated and improved quasar_host.h and quasar_dsl.h.
- Added quasar_rt_bindings.h, quasar_dsl_class_ext .h and quasar_type_impl.h header files.
- New module system for native C++ modules, which can be imported using import "native_lib.$BITNESS$.{NATIVEEXT}"
- Added the ability to register C++ classes to Quasar.
- Improved the Visual C++ compiler detection mechanisms in Windows
- Added compiler diagnostics tool in Windows, which gives warnings when the current C++/CUDA configuration is not correct. Just open the "Program settings" dialog box in Redshift to run the tool. If there are no message boxes popping up, everything is successfully installed.

27/6/2019

- Maintenance fixes
- Redshift: global device metrics (utilization, temperature, power draw can now also be captured during profiling

30/5/2019

- Added support for the Visual C++ 2019 back-end compiler
- Added "multidimensional loops" to avoid nesting of for-loops for [p,q,m,n]=0..size(x,0..3)-1 y[p,q,m,n] = x[p,q,m+d[0],n+d[1]] endfor

- Quasar.DNN.dll: added batch normalization functions cudnn_batchnormalization_forward_inference cudnn_batchnormalization_forward_training cudnn_batchnormalization_backward
- Quasar.DNN.dll: improved support for 5D tensors
- Quasar.DNN.dll: relaxed image input/output dimension conditions for convolution
- Quasar.CompMath.dll: added gradient calculation via $diff(var, {x, y})
- Quasar.CompMath.dll: improved auto-differentiation support
- Quasar.CompMath.dll: added $diffeval() and $diffmuleval() for simultaneous derivation and function evaluation.
- Added support and optimizations for Turing GPUs (NVidia Geforce RTX)
  - warp synchronous programming via "syncthreads warp"
  - improved SIMD half float support
  - added convolution_descriptor.math_type to enable Volta tensor cores
  - kernel fusion framework (documentation will follow soon)
- Added recursive operations on nested classes suppose A and B are classes of the same type => A+B will create a new instance of the class, where each field is the sum of a field in A and a field in B. This works for all unary and binary operators in Quasar.
- Improved automatic kernel generator for patterns such as A[0][1][:,:,:]
- Redshift: added code work bench log level setting
- Redshift: added ability to disable classes of warning/optimization messages.
- Redshift: reset debugging session is now also a toolbar icon
- Redshift: added nvidia-smi as GPU performance monitoring backend
- Redshift: key binding Ctrl+L deletes the current line (previously Ctrl+D)
- Redshift: key binding Ctrl+D duplicates the current line
- Added preview function: fftn() for 4D, 5D, … fast fourier transforms. In addition, fftn(X, 1..2) performs fft along the first and second dimensions (slice FFTs). The only requirement is that the dimension indices are contiguous (for example fftn(X, [1,3]) is not supported but can be obtained using the less efficient fftn(fftn(X,1),3).
- Parallel reduction with partial dimensions now supports reductions with 2 dimensions or more.
- stdlib/linalg.q now contains dynamic memory-free specializations for det(), inv() and cholesky() for 2x2, 3x3 and 4x4 matrices
- qgldisplay: improved keyboard handling (alt+arrow key to move the camera, etc.)
- qgldisplay: added suspend_updates() and resume_updates() function to temporarily prevent the view from updating/rendering.
- qglvectorlayer3d.drawpoints() and qglvectorlayer3d.fillquads() now support offset and length parameters (useful for large vertex buffers)
- qglvectorlayer3d.fillvoxels() allows drawing large sets of voxels as individual cubes.

8/2/2019

- Maintenance release
- gldisplay: added keyboard-based navigation (arrow keys, PgUp/PgDn, minus, plus)
- C++ API: extended RuntimeSettings struct with more runtime options
- plot(), scatter(): added plot markers (for example, plot(x,y,"ro")). See documentation for the plot() function.

26/11/2018

- explicit SIMD instruction generation using C++ code generator backend (currently, SSE, SSE2, AVX and AVX2 are supported). Speeds up CPU code.
- local windowing transform {!kernel_transform enable="localwindow"} upgrade; improved performance of generated code
- improved performance of separable filter in imfilter.q
- partial support for half precision floating point formats (via floattypes.q)
- improved handling of aligned versus unaligned memory accesses.
- Redshift: display the minimum and maximum of realvalued matrices during debugging
- Redshift: added algorithmic differentiation manual

21/9/2018

- Redshift profiler: added export to PDF feature
- Redshift profiler: fixed many issues from the previous release
- Redshift: Improved background compilation to run more smoothly without interfering with the IDE
- Add parallel back-end compilation option. Considerably speeds up the back-end compilation.
- Added qvectorlayer.drawimage() optimization to keep the OpenGL buffer
- qglrenderer, qimshow: fixed issue with context stack when saving videos
- Redshift: fix for variable watch pad showing disposed matrix
- qimshow: fix for fullscreen mode of windows with multiple OpenGL displays

10/9/2018

- Integrated NVIDIA CUDA Profiling tools interface:
  - more accurate kernel timing measurements
  - less profiling overhead
  - integrated source correlation, to see instruction counts per code line
  - added detailed kernel report, using new metrics, such as sm_efficiency, achieved_occupancy, instructions_per_-clock_cycle, avg_instructions_per_warp, branch_efficiency, warp_execution_efficiency and many others.
- CUDA runtime: improved performance of fft2, ifft2 and realfft2 functions for multiple slice data (e.g. RGB color images).
- CUDA backend: added option to add line number information in PTX file
- Pascal architecture HW grid synchronization is now functional
- Added multi-GPU support for cuBLAS, cuSolver and cuDNN libraries

9/8/2018

- C++/CUDA code generator: generated code uses control structures (if, while, for etc), improving the readability of the code

- Compiler: improved the error handling and reporting routines, giving more context information about the individual errors
- Fixed sized array data types: now matrices and vectors up with to 64 elements are allocated on the stack/in the registers. Calculations with 8x8 matrices with size known at compile-time now entirely avoid dynamic kernel memory
- Added 'shared access modifier for designated shared memory (allowing to easily write kernel functions that take advantage of shared memory of the GPU)
- Shared memory caching transform is now deprecated in favor of designated shared memory (see documentation)
- Redshift: improved the error list pad, variables pad, directory browser pad and profiler
- Redshift: added shortcut keys to show/hide the different pads
- Redshift: kernel/function optimizer is now renamed to code workbench and can be activated using the F6 key
- Redshift: improved formatting of tooltips and error messages
- Debugger: various improvements (such as debugging of intermediate code in the code workbench window, access to variables in other contexts)
- Generic types now also allow division operations (for example A : vec[N/2]) as long as the result N/2 is an integer.
- First release of the algorithmic differentiation library
- Added gpu_count() function (returning the number of GPUs available in the current configuration)
- Hyperion engine: integrated a new sophisticated scheduling algorithm, that significantly improves execution time and device utilization for multi-GPU workloads.
- dwt.q, dtcwt.q: the wavelet API functions have changes and now take a wavelet_params object which contains pre-calculated filter coefficients.

6/6/2018

- imread: added 'pixelfmt' parameter to select the pixel format ('rgb' or 'rgba') im_rgb=imread("img.png", "rgb") creates an RGB image with 3 color channels, im_rgba=imread("img.png", "rgba") creates an RGBA image with 4 color channels. The type inference is adjusted automatically so that img_rgb : cube(:,:,3), img_rgba : cube(:,:,4)

24/5/2018

- Added fixed sized array data types which allow avoiding dynamic kernel memory inside kernel functions
- Built-in functions ('ones', 'eye', 'transpose', 'repmat', 'shuffledims', 'ones', 'cones', 'zeros', 'czeros) support the fixed sized array types
- Added dimensionality/size constrained array types, for cases where array dimensions are partially known
- Added dimensionality/size parametric array types, for generic programming
- Renamed the former cube{n} to cube{:} and ccube{n} to ccube{:} (cube{n} and ccube{n} can still be used but are obsolete now).
- General type inference improvements for built-in functions
- Added the 'pos2indchk' function, which is an extension of 'pos2ind' that returns -1 when the position argument is out-of-bounds.
- Added support for CUDA 9 cooperative groups and warp shuffling
- Added syncthreads(warp) (synchronization within a warp) and syncthreads(grid) (synchronization of the entire grid)

- Added opt_block_cnt() which determines the optimal number of blocks so that all blocks are active at the same time (useful for grid synchronization)
- Improved the dynamic memory performance compiler warnings to be more accurate
- nvcc_script.bat now uses the preferred compiler setting (required because CUDA 9 does not support all releases of Visual C++ 2017)
- Propagation of global constants to kernel and device functions
- Implemented a workaround solution to prevent a stack overflow in the Nvidia JIT compiler.
- Implemented context switching for the DNN library (CuDNN can now be used with multiple GPUs)
- Simplified the 'imhist' function (imhist.q) taking advantage of the shared memory caching transform; results in speedup of 10-15x.

4/12/2017

- C++ API fixes and enhancements
  - moved quasar::ref from quasar_dsl.h to quasar_host.h
  - renamed LambdaDelegate to LambdaDelegate*
  - added string_t::pad_left, string_t::pad_right and string_t::sub_string functions
- added "auto" mode to IQuasarHost::Create (allowing Quasar to choose the device automatically)
- plot layout and formatting enhancements
- added barplot() function (for displaying bar plots)
- qform: added activate(), minimize() and maximize() methods
- fixed several C++/OpenMP backend issues
- AutoKernelGeneration upgrade: added support for 'dotprod' function and also the pattern transpose(x)Ax. Both now avoid use of dynamic kernel memory.
- Compiler and runtime performance enhancements
- Improved error handling in autogenerated code
- Improved performance hints related to the use of dynamic kernel memory

8/9/2017

- Compiler: Added {! kernel memorymodel="stack"} as a means to avoid the dynamic memory system and achieve higher efficiency

- AutoKernelGeneration upgrade: added support for the built-in functions 'transpose', 'repmat', 'shuffledims', 'ones', 'cones', 'zeros', 'czeros', 'eye'

- Compiler: Added compiler-generated parallel reduction/prefix sum algorithms for dimension functions sum(x,k) and cumsum(x,k)

- Compiler: Added loop fusion and loop fission code transforms

- Compiler: Added ability to specify reduction parameters (e.g., name, priority) Example:

```
reduction {priority=high, name="sin"}, x -> f(x) = sin(x)
```

- Compiler: block dimension optimization -kernel functions are specialized based based on available block dimension information.
- Redshift: integrated inspection of the function and kernel transform pipelines
- Redshift: compile-time tuning suggestions
- Added support for the Visual C++ 2017 back-end compiler

4/4/2017

- Added lambda capturing reductions

  ```
  reduction (f : function, g : function) -> f - g = x -> f(x) - g(x)
  ```

  example: (fn1 - fn2)(1) becomes fn1(1) - fn2(1)
- Added variadic lambda capturing reductions reduction (x,y,…f, …g) -> sum([f(x) .* g(y)]) = calc(f, g, x, y) example: a1(x).*b1(y) + a2(x)*.b2(y) becomes calc({a1,a2},{b1,b2},x,y)
- Added a named parameter list feature: unpacking of objects to functions param_list = { param2 := 2, param1 := 1, other := "hello" } function [] = my_func(text, param1 = 0.2, param2 = 0.6) … endfunction my_func("param:", …param_list)
- Added support for atomic exchange and atomic compare and swap operations b = (x <- new) % reads the value of x, replaces it by 'new' and stores % the old value in b b = (x <- (old, new)) % reads the value of x, when 'x==old' sets 'x=new', % and stores the old value of x in b.
- Added basic support for enums, as a better way for passing named items (instead of passing strings) type color : enum red orange endtype my_color = color.red
- Added basic support for exception handling, as an improved error handling mechanism. Example:

  ```
  try
    error "here I fail"
  catch ex
    print ex
  endtry
  ```
- Added support for generic type aliases: type bucketlist[T] : vec[linkedlist[^T]]
- Added branch divergence reducer kernel transform {!kernel_transform enable="branchdivergencereducer"}
- Added ability to compile and run .cpp modules with main() functions directly from Quasar and Redshift
- Added various binding examples (including IronPython, C++/Quasar)

- system.q: separated diag() (extracting the diagonal from a matrix) from diagm() (constructing a diagonal matrix)
- Loop parallelizer upgrade: allow parallelization of inner loops, for example {! parallel for} for m=0..size(im,0)−1 avg = mean(im[m,:]) {! parallel for} for n=0..size(im,1)−1 im[m,n] = im[m,n] − avg endfor endfor In the past, this was only possible when explicitly writing out the inner kernel function. There is one restriction: for GPU devices, the inner loop cannot have scalar/vecX output parameters (for example accumulation patterns).
- Redshift debugger: integrated "run external process" and added support for CUDA memcheck (to obtain more information about CUDA errors).
- Redshift editor: allow to change the line height
- Library: added data structures: List[T], LinkedList[T], Stack[T], Queue[T], Hashmap[T], Concurrent_set[T]

1/2/2017

- Kernel dynamic memory system: added option to configure the parameters
  - total amount of kernel dynamic memory (in MB)
  - max allocation block size (in KB)
- Quasar.UI.dll: several new features
  - added qprogress class
  - qcombobox class now also allows vector binding, allowing the items of the combobox to be easily updated
  - added qevent.clear(), which removes all event handlers for a specific event
- Quasar.Xml.dll: updates and new features
  - renamed xmldocument to qxmlnode
  - added qxmlnode.remove, qxmlnode.get, qxmlnode.outerxml, qxmlnode.serialize and qxmlnode.deserialize

13/1/2017

- Compiler: various type inference improvements (type inference for fixed cell array types, e.g., tuples)
- Compiler: improved handling of polymorphic variables in type inference, with type widening and type class union mechanisms, taking better advantage of information available at compile-time
- Compiler: added type annotation merging: for example, to declare a constant variable while using type inference to determine its type, one can now write a : 'const = 2.0
- Added support for variadic reductions, for example reduction (a : string, ...args : vec[??]) -> op(a, args) = handler(a, ...args)
- Added support for variadic methods, for example function [] = format(self : myclass, ...args : vec[??])
- Lambda expressions can now be explicitly typed, for example: wrap : [mat -> vec -> mat] = A -> x -> x * transpose(A) is equivalent to the former wrap = (A : mat) -> (x : vec) -> x * transpose(A) but also allows the return type to be specified
- added support for lambda expressions with multiple return values swap : [(??, ??) -> (??, ??)] = (x, y) -> {y, x}
- added {!parallel for; multi_device=true} as a simplified way to parallelize over multiple devices.

- Hyperion engine: added multi-GPU memory resource optimizations
- kernel back-end: added support for cell matrix constructors {a, b}, mainly useful for handling functions with multiple return values
- various enhancements for the CIL (.exe) code generator
- Redshift memory profiler: report memory block lifetime
- Redshift: added memory info in variables window for the Hyperion engine
- External C++ API updates:
  - added support for various C++11 features (variadic arguments, lambda expressions)
  - added device selection and enumeration interface
  - added RuntimeControl class, for controlling the Quasar runtime parameters from C++
  - Added Reduction class, for simple definitions of reductions from C++
  - Added various examples of the new functionality

5/11/2016

- Compiler: improved warning message when a loop is forced to be parallelized despite detected dependencies
- Compiler: improved the type inference engine for function return value inference, allowing it to follow conditional branches in Quasar programs.
- compact_core.q: improved the implementation of the min(), max(), sum(), prod(), cumsum(), cumprod(), mindim(), maxdim() and shuffledim() functions
- added imaffinetransform.q: affine image transformations
- added guided_filter.q: guided filters
- added minsphere.q: calculation of minimum bounding spheres
- added Quasar macros
- added $target() function for enabling target-dependent optimizations
- Redshift: added auto-completion for code attributes
- reenabled the cooperative memory support based on the Quasar external C++ API
- nvcc_script.bat: added CUDA 8.0 support
- CUDA runtime: fixed auto-coalescing for 3-dimensional kernels
- External C++ API updates:
  - added exception_t class for exception handling
  - added a custom exception handling mechanism
  - added some operators for assigning used-defined objects to QValue and QValue matrices.
- Hyperion engine:
  - added support for cuBLAS and cuDNN libraries
  - improved support for higher-dimensional matrices
  - improved profiling of external library calss (cuBLAS, cuDNN)
- Improved the CIL (.exe) code generator

23/9/2016

- Added the randseed function
- Deeplearning library (deeplearning.q) has now support for cudnn

15/9/2016

- Added experimental serialization (i.e. native code generation) of while loops

9/9/2016

- Improved compiler warning message for uncoalesced access

5/7/2016

- Added specialization code attribute, to be placed inside kernel functions.

  ```
      {!specialize args=(radius==1 && channels_out==3 || _
                         radius==2 && channels_out==3 || _
                         radius==3 && channels_out==3) }
  ```

  The compiler will then generate separate specializations for each condition separated by OR (||).

- Added built-in support for cuBLAS (CUDA devices only)

- Added built-in support for cuSolver (CUDA devices only)

- Added support for cuDNN (import Quasar.DNN.dll)

- Added warning for uncoalesced matrix accesses within kernel functions

- CUDA engine: removed atomic operations in the sum() and prod() function, to have a consistent result

- Runtime: added mindim(x,dim), maxdim(x, dim) function (minimum and maximum along a given dimension)

- Runtime: added cumsum(x,dim), cumprod(x, dim), cummin(x, dim) and cummax(x, dim) functions (cumulative operation along a given dimension)

- Runtime: improved several builtin-functions to handle higher-dimensional matrices (4D and higher) more efficiently

- Added pos2ind() function, for converting positions to indices (inverse operation of the already existing ind2pos()). The function is used internally by some kernel function transforms.

- fread/fwrite: added support for “float16” and “complex16” (16-bit half-precision floating point) formats

30/5/2016

- Added sharedmemcaching transform, for caching small matrices in shared memory (both input/output)
- Redshift: display low-level CUDA kernel information in the kernel function tooltips in the left margin bar in the source code editor

11/5/2016

- Added several new runtime functions: compressfile, decompressfile, copyfile, movefile, deletefile, fileparts, fullfile, isrelativepath, makepath, uigetdir, uigetfile, uiputfile, has_display, csvread, csvwrite
- Enhancements for Redshift: source code editor performance optimization
- Redshift directory browser: added new features (filter, home button)
- Added {! cuda_launch_bounds max_threads_per_block=1024; min_blocks_per_sm=8} attribute for generating CUDA launch bounds directives (better controlling the registers of the generated kernel)
- Improved the inference for automatic boundary check removal. Some code is now 10-20% faster.
- CUDA v1 runtime: optimized sum(), prod(), min(), max() functions with 1 argument. The performance has increased significantly.
- Automatic debugger session reset after CUDA error when CUDA context is invalidated
- Redshift: added ability to import/export CSV files
- fread, fwrite: added support for reading and writing complex valued matrices

19/4/2016

- added program for the undecimated wavelet transform (udwt.q)
- automatic kernel generator can now handle aggregation functions sum, prod, min, max, resulting in a parallel reduction algorithm
- added support for the NVidia Runtime compiler (CUDA 7.5 64-bit only)
- added runtime functions mkdir(), exist(), pwd()
- redshift: improved code feedback - explicit distinction between info and optimization hints (yellow underlining)
- compiler: added simple algebraic simplification optimization
- compiler: added detection of superfluous/unused code (aka dead code).

14/3/2016

- added the feof() function
- Quasar.UI.dll: added mouse handler for qgldisplay class
- external C++ interface: added support for higher-dimensional matrices
- external c++ interface: added quasar::CoreLib class
- important: SYNTAX UPDATE!!! To improve the readability of Quasar code and for consistency reasons (at the expense of extra typing:-)), from now on end block markers should match the block start.
  - if goes with endif (as before)
  - for goes with endfor
  - function goes with endfunction
  - match goes with endmatch
  - type goes with endtype
  - while with endwhile If you don't like the new endings, you can still work in the old syntax. If you switch to the new scheme, compiler errors will be generated if you use the old scheme (end) again in the current file. This

is done on a source file level. Note: fully backward compatible, all existing code will still work. Currently in a testing phase: you are not forced to switch.

11/2/2016

- Improved the image viewer full screen mode
- CUDA runtime: improved the handling of hardware textures.
    1) Performance improvements when used in read/write mode.
    2) Texture width no longer needs to be aligned
- Quasar.Video.dll: added support for 48 bit/pixel HDR videos

1/2/2016

- Added splash screen
- Improved variable definition window in order to become a hybrid "project" explorer and watch window.
- qimshow.connect: 3D image viewers are now also connected in 3D mode
- Redshift: added argument tooltips for parallel_do/serial_do functions

17/1/2016

- Windows: 64-bit version for Quasar and Redshift. Windows users need to download and install: http://telin.ugent.be/~dvhaeren/files/ sharp-2.12.12-x64.msi

11/1/2016

- the semantics of ivecX, vecX, cvecX data types changes: values of these types are from now on passed by reference, to be consistent with the corresponding ivec, vec and cvec types. The vector length is then an additional constraint that is used by the compiler to generate efficient code.
- Added support for variadic functions function [] = my_function(... args)
- Implemented the spread operator, for unpacking vector values a[...pos, 0] is equivalent to a[pos[0], pos[1], 0] when pos is a vector of length 2
- Added support for non-coherent texture caches using the 'hwconst modifier (CUDA compute capability 3.5 or higher)
- Matrix constructors [[1,2,3],[4,5,6]] can now be used from within kernel/device functions
- Improved handling of kernel error checking, removing error checking when unnecessary (resulting in several performance improvements)
- Added support for CUDA constant memory (via the 'hwconst modifier). See documentation for usage. Results in speed ups of x2 or x3 for several kernels (e.g. convolution kernels)
- Improved the performance of the code generated by the parallel reduction transform
- Added optimization hint when the LocalWindowingTransform can be applied
- Improvements to the CUDA/CPU memory allocator (especially when reaching the maximum available memory bounds)
- Added (experimental) support for CUDA 6.0 unified memory

- Added 'nocopy modifier, to avoid unnecessary copying of data to the target device. The modifier is generated internally by the compiler.
- Implemented CUDA surface writes (for writing to matrices in texture memory via the 'hwtex_nearest modifier).
- Added support for ivecX, vecX and cvecX kernel function return values.
- The hardware texturing units can now be used in 64-bit float mode (the data will automatically be converted to 32-bit float by an internal kernel, because the GPU does not support 64-bit float textures yet).
- Added 'blkcnt' and 'warpsize' (special kernel function parameters) for accessing the block count (or grid dimensions) and warp size from a kernel function.

29/11/2015

- Average compilation time improved by 10%-30%
- Separate compilation stages cause compilation errors to be reported faster.
- Extensive testing & regression fixes
- Redshift: 'which' command can be used to locate .q, .qlib and .dll files
- Windows: DLL_SEARCHPATH32, DLL_SEARCHPATH64 are now active and make the resolution of windows dlls more reliable. These folders are searched first, before the system path and other windows directories.
- Windows: all user files are now stored in the user application directory (and no longer in Program Files). Useful for when there are multiple users.
- CUDA runtime: memory allocation enhancements, allowing to deal better with CUDA driver memory fragmentation
- Updated nvcc_script.bat and cc_script.bat to be more robust against CUDA/MSVC versions and target CPU architecture support. Checks the existence of MSVC's cl.exe.
- Redshift load/save file dialogs now memorize the last folder and selected file extension
- Redshift output log: added context menu option to prevent the log from clearing automatically.
- Compiler: high-level inference for improved boundary checking. Will positively impact the performance of existing Quasar programs.

28/9/2015

- Runtime: added string manipulation functions tounicode, fromunicode, toascii, fromascii for converting between matrices and text strings
- Runtime: added minvalue(.) and maxvalue(.) functions for determining the min and max values for a given type
- Runtime: modified the behavior of the size(.) function for strings
- Runtime: added support for OpenGL antialiasing modes (can be configured from Redshift)
- system.q: is using the latest atomic operators and minvalue, maxvalue functions
- system.q: added "last" mode for the find function.
- colormap.q: the default color map is now "gray"
- Compiler: added .qlib build feature, via the -make_lib parameter
- Redshift: enabled the .qlib build feature for making shared libraries that can be used from other Quasar code
- Redshift: added support for the Microsoft Visual C++ 2015 back-end compiler
- Quasar C++ host: added possibility to load only the runtime (and not the compiler), resulting in faster loading of the runtime environment

- Quasar C++ host: implemented the LoadBinaryModule function for loading .qlib files
- Quasar C++ host: added OpenGL/CUDA interoperability samples
- Redshift: added preview pane in the import raw data dialog box.
- Added LibDICOM for reading DICOM files (medical images)
- Some documentation updates and fixes in Redshift

3/9/2015

- Added OpenGL optimizations in CUDA-OpenGL operability
- Improved the compiler type inference
- Fix for the realfft3 and irealfft3 functions
- Improved the 3D image viewer
- Image viewer: added support for 4D data volumes (3D + RGB/RGBA color)
- qplot class: added xlog, ylog, xtick, ytick, xticklabel, yticklabel properties
- added legend() function

15/4/2015

- Added {! interpreted for} attribute (next to {!parallel for} and {!serial for}), to enforce interpretation of the loop (e.g. for debugging purposes).
- Easy multi-GPU mode by using attribute

```
for k=0..N-1
    {!unroll times=2; multi_device=true}
end
```

- Debugger: enhancement to reduce module load times
- Redshift: improved profiling & visualization for OpenCL
- Compiler: removed some unnecessary warnings
- CUDA back-end: added optimizations for the pow(.) function
- CUDA back-end: experimental support for CUDA dynamic parallelism (sm_35 or higher). Requires Geforce 780, Geforce TITAN or Geforce 920+.
- CPU back-end: added simple run-time load balancing for nested kernel functions with OpenMP (improves CPU performance for nested kernels in many cases)
- CPU/CUDA back-end: shared(.), zeros(.), ones(.) and uninit(.) now also allow support the creation of higher dimensional (>3 dims) matrices
- CUDA run-time: added option for 16-bit floating point textures (improve global memory bandwidth) instead of 32-bit floating point, for applications such as video processing where performance is more important than accuracy.

- Runtime: added experimental support for the Intel Xeon Phi accelerator (OpenCL)
- added irealfft1 function (fast inverse real-valued 1D FFTs)
- quasar.h: renamed complex to cscalar to avoid confusion with std::complex
- quasar.h: renamed raise_error to quasar::raise_error to avoid confusion with boost
- quasar_dsl.h: added missing operator= for the ref class (it is no longer required to use boost::shared_ptr instead)
- quasar -version now also prints OpenCL devices

8/3/2015

- Redshift: added support for program arguments (passed to the main function)
- Redshift: added dialog box for setting program arguments
- Compiler: added type narrowing for type reconstruction
- Redshift: ability to enable/disable OpenGL from the Redshift Program Settings
- Redshift: ability to select the default image codec provider (GDI+ or GTK+)
- added support for the Intel Compiler v15
- more conservative use of system memory, after extensive memory profiling
- various enhancements for the CUDA memory manager

11/01/2015

- First Quasar version with MAC OS X support! Some issues still remain to be resolved.
- Added C++ bi-directional API for using Quasar from C++ (or using C++ from Quasar)
- Added new atomic operations |= (OR), &= (AND), ~= (XOR), ^^= (MAX), __= (MIN)
- Extended the parallel reduction loop transform to support the new atomic operators
- Added regions for user-defined code folding ({!region} - {!endregion})
- Added braces {} as a replacement for ’ in defining cell matrices (more convenient for people familiar with MATLAB). From now on, the syntax’ is still supported, but obsolete. In future versions, ‘’ will be used for ‘expressions’.
- Added {} for the construction of anonymous objects: f = {x:=4, y:=6} is equivalent to: f = object(); f.x = 4; f.y = 6
- Added support for dynamic classes
- Implemented run-time short-circuiting for && and ||
- Compiler now generates a performance warning for implicit conversions between integer and floating point numbers.
- Compiler: improved dependency analysis report for automatic loop parallelizer
- Added support for higher dimensional matrices (4D, 5D etc). Up till 16D is currently supported. Should be sufficient to cover string and M-theories:-)
- Added generic OpenCL code generator back-end (writes pure .cl files).
- Optimizer: added support for loop vectorization (via loop tiling transform)
- Redshift: added full recompile button
- Redshift: added compiler back-end configuration options
- Redshift: added horizontal line/function separator for improved code viewing

- Redshift: added packaging tool
- Added new parallel high performance function library (see parallel_blocks.q)
- Redshift: added license mgt system and connection with the webserver quasar.ugent.be. Current users should enter their name, institution etc and will automatically obtain a license from the server.

16/11/2014

- Redshift: loop optimizer panel for viewing loop transforms
- Redshift: several memory enhancements (after finding an issue where older versions preallocated too much memory)
- Redshift: added printing feature (to print .q files to a local printer)
- Redshift: upgraded the toolwindow docking system
- Redshift: new more modern design for tabs (Windows + Linux)
- Redshift: error underlining feature & tooltips are enhanced
- Add support for the placeholder for assignments [a, _] = func(1, 2)
- Revised the compiler back-end (more efficient CUDA and OpenMP code generation)
- New supported C++ back-end compilers: Clang and Visual C++ 2013!
- Support for platform-specific kernel and device functions
- Compiler front-end enhancements: several new loop transforms
- The compiler now uses less memory while compiling (which should improve the compilation speed)
- Internal run-time optimizations
- the size(.) function no longer returns a vector of length 3
- Improved back-end support for atomic operations in Windows and Linux
- Improved support for recursive generic types
- Improvements for the CIL back-end (which generates Quasar .EXE files)

22/10/2014

- Added a compiler warning for when the parallel_do size dimensions do not match the kernel function.
- Added qimshow.start_capture, qimshow.end_capture, qgldisplay.start_capture and qgldisplay.end_capture functions for video recording.
- Fixed a longstanding problem with OpenGL rendering in Windows, where the the controls turn white after switching between tabs in Redshift.

15/9/2014

- Redshift: directory browser & profiler views are now sortable by clicking onto the column headers

2/9/2014

- Implemented labeled loops with break (see wiki)
- Keys Ctrl-Tab, Ctrl-Shift-Tab for switching between editor documents

- Updated the documentation (small changes)
- Redshift preferences dialog box: fixed an issue with changing the editor font
- Fixed C/C++ syntax highlighting (removed duplicate keywords, added new keywords)
- Fixes #152 - GCC compilation error from last commit
- Fixes #176 - imshow() auto intensity range

11/8/2014

- Redshift: parsing & error highlighting while typing
- Redshift: improved auto-completion
- Redshift: added edit-and-continue feature (editing code when the program is paused)
- Redshift: added breakpoints shortcut menu
- Redshift: ability to compare different profiles
- Redshift: added several new user settings
- Redshift: improved document browser
- Redshift: added documentation preview window
- Redshift: added code listing window
- Quasar.UI.dll: added combo boxes & menus
- Quasar.UI.dll: extended qgldisplay and qvectorlayer3d classes
- Quasar.UI.dll: added access to the RGB color & depth buffers of the GPU
- Several fixes and enhancements

29/1/2014

- added parametric types
- the dynamic kernel memory feature has been significantly improved and should now be usable, some more testing will be needed
- added the cooperative and concurrent dynamic kernel memory models
- functions mean, sum, prod, min, max, dotprod, transpose, eye, copy, reshape, repmat, shuffledims, matrix multiplication, seq, linspace are now supported inside device/kernel functions with matrices of variable size (turn on the compiler setting for dynamic kernel memory)
- improved the automatic loop parallelizer to take advantage of dynamic kernel memory when enabled
- improved the type inference for the repmat and shuffledims functions
- implemented type reconstruction; this feature significantly improves the type inference; in many cases it is no longer required to specify types for function arguments
- support for nested parallelism (parallel_do/serial_do can be used from within kernel functions)
- improved the CUDA computation engine to handle concurrent kernel execution more efficiently (speedups are possible from 10-20%)
- support for shared memory allocation of non-scalar types (sharedT function)
- added support for high-level operations inside kernel functions (e.g. matrix subtraction…)
- added the deepcopy function
- added the nan, posinf, neginf variables

- added support for class inheritance
- added constructor pattern for creating objects
- added parametric reductions (reductions for which the types of some of the input variables are parametric)
- improved the opengl_renderer (qgldisplay) to allow camera rotation (yaw, pitch and roll), rendering of multiple objects and vector layers.
- extensive regression testing => several critical bugs were identified and solved
- added the surf3d function for rendering surface plots

15/12/2013

- updated the manual with some changes that were already pending for more than one year
- added support for main functions with optional arguments
- added protection against infinitely recursive function calls
- Redshift: code completion window also shows member functions
- Redshift: improved debugger step-into/step-over speed
- Redshift: code completion window now also works for the immediate window
- Redshift: added tooltips for member functions
- Redshift: improved the background compilation system
- Redshift: added a documentation browser system
- Redshift: first version of the cloud tuning system
- Redshift: added preferences dialog box
- Redshift: improved the efficiency of the variables window
- Redshift: improved the error list window
- Redshift: added comments for function parameters in the argument tooltip boxes

25/11/2013

- general optional arguments are now supported (such as functions, matrices, variables, ...). For example:

  ```
  function y = do_something(x = eye(4))
  function y = do_something(x = eye(4), y = [[1,2],[3,4]])
  ```

  is now accepted by the compiler

- Implemented constant propagation for kernel functions. Correspondingly, the following will generate an assertion failure function [] = __kernel__ kernel (b : scalar , pos : ivec3) assert (b==3) end parallel_do (size (im) ,2, kernel)

- Improved shared memory bounds estimation (improves the performance for certain kernel functions)

- Redshift: improved tooltip detection in the code editor

- Redshift command window: copy-paste of multiple lines is now treated correctly

- Redshift: added window list dialog

- Runtime system: added the HSV color map to imshow()
- Redshift profiler: now shows multiple launch configurations in the kernel details panel
- imfilter.q: added 'clamped' boundary extension mode
- cannyedge.q: first implementation of a canny edge detector

11/11/2013

- updated libavcodec (windows) to version 2.1
- added new modules to the library: bilateral_filter.q, colored_noise.q
- added several new demo programs: opengl_cube.q, nlmeans_denoising_video.q, subplot_example.q, video_viewer.q
- existing user interface library has been migrated to GTK and needs to be imported from now on (import "Quasar.UI.dll")
- added new operators .*=, ./= and .^= (point-wise inplace atomic operations)
- device functions are now run natively or interpreted depending on the context and their complexity
- loop parallelizer now recognizes #pragma force_serial/force_parallel in innter loops
- added reduction loop transform for aggregation. For example:

  ```
  total = 0
  for m=0..numel(v)-1
      total += v[m]
  end
  ```

  will now be expanded in a sophisticated kernel function that makes use of shared memory. See documentation (wiki) for all possible use cases, actually, the loop transform is quite general.
- loop parallelizer: more than 32 input variables are now supported
- big improvements to the generic programming system and specialization engine. Kernel function arguments actually no longer need to be typed, in many situations the compiler can infer the correct type from the context (or by specializing outer functions).
- improved reduction handling system. In previous versions, the reduction application order could give certain (rare) problems and some reduction conflicts were not recognized. This is now fixed.
- improved the assertion system (dealing with different code paths and basic blocks)
- added compiler option "logically optimize expressions" (useful for if expressions)
- reduction system now correctly treads the reduction "A : mat -> A[:,:] = A" and only applies it to the RHS of expressions.
- 32 bbp RGBA images (PNG) are now loaded with 4 color components, so that the alpha component is still retained.

- CUDA engine: updated to take advantage of the CUDA 5.5 functionality. Result: error handling in concurrent kernel execution mode is now working. Profiler results for kernel end times are now improved.
- Profiler: occupancy calculation now takes both the grid dimensions and the maximum number of blocks per streaming processor into account. Occupancy results are now equal to NVidia's occupancy calculator.
- Redshift: added the parallel debugger pane.
- Redshift / Spectroscope: debugging of kernel functions with shared memory and/or thread synchronization is now supported.
- Redshift: revised the "view" menu, added "hide/show all tool windows" and "function bar" menus.
- 50 small fixes.

21/10/2013

- Various improvements and speed ups of the compiler (up to 40%-50%) and runtime
- function overloading for functions with optional arguments is now supported
- classes can now have string members
- added the function fgets(.)
- added the function opt_block_size(.) function for determining the optimal block size
- added the function squeeze(.) for removing singleton dimensions
- implicit member function definitions with 'self' argument are now supported
- native function calls now support multiple output arguments
- rewrote the code generator for generating binaries: extra optimizations are possible that significantly speed up the generated code (see wiki on GitHub)
- assertion system now takes branch basic blocks into account (and should work correctly with loops)
- Redshift profiler: added the collection of block and grid dimensions
- CUDA engine - implemented a new algorithm for optimizing the block size (now works correctly with even the most exotic data dimensions)
- Redshift: added documentation for built-in functions
- Redshift: added 'help(function)' debugging command

24/09/2013

- Mainly a maintenance release with substantial improvements in compilation/performance/stability
- Implemented native C/C++ interface (see ExternalInterfaceReference.pdf): kernel and device functions can now be implemented directly in C/C++. This is very useful when integrating existing C++ code with Quasar.
- CPU engine: device functions are now natively called rather than interpreted (idem for GPU engines). This gives huge performance benefits for **device** functions.
- CPU engine: the serial_do/parallel_do function has completely been rewritten to use the new native call functionality (result: 10%-25% faster)
- Redshift: C++ syntax highlighting for C/C++/CU and related header files

- Redshift: C/C++ files can now be opened (they are also listed in the directory browser)
- Image viewer: added "save screenshot" menu
- Redshift: added zoom button in code editor (Ctrl+,Ctrl- keys) + line number information
- Redshift: auto code formatting
- Redshift: added 'edit' command to open text documents (example: edit dwt.q)
- Redshift: first version of the memory profiler
- Redshift: first version of the kernel function debugger: breakpoints can be placed inside kernel functions, which causes the debugger to interpret the kernel function.

16/06/2013

- Added logical and/or/not operators (&&, ||, !) for matrices, as well as the bitwise operations and, or, xor, shl, shr, not
- The "mean" function can now also be used from **kernel**/**device** functions.
- Compiler setting "automatic inlining of lambda expressions" now also controls the automatic function specialization
- Added the conditional matrix assignment reduction to system.q:

```
reduction (x : matrix_type, a : matrix_type, b, c) -> (x[a > b] = c) = (x += (c - x) .* (a > b))
```

- Added some new filters to imfilter.q
- Added TIFFIO library for advanced support of .TIF files (source code included in Interop_Samples)
- Added progress dialog box when loading/saving big .qd datasets
- Added shared memory optimization by specifying an upper bound of the shared memory through assertions
- Redshift: changing directory also affects the OS current directory
- imread and fopen functions now perform a small search when the file is not found
- It is now possible to save profiling information to .qprof directly from commandline: Quasar.exe -profile:my_prog.qprof my_prog.q
- Improved profiler: now also collects CPU memory allocation information
- Redshift: renewed the profiler with novel tabs (dashboard, kernel details, memory profiling details)
- Redshift: program settings - added configuration tab for the compiler backend (under construction, not operational yet).
- Extensive testing of the CPU engine

20/05/2013

- Improved the assertion system (by using a new logic system)

- Added ‘unassert’ function
- Compiler checks for function existence
- Implemented several meta functions ($subs, $str, $inline, $specialize)
- Compiler: added type inference for cell array constructions (if all elements have the same type)
- Compiler: added option for avoiding function pointers
- Compiler: added option for dynamic kernel memory
- Added support for dynamic kernel memory (that can be used from kernel/device functions).
- Compiler: added reduction where clauses (conditional reductions)
- Compiler: implemented type classes, such as [int|scalar|cscalar]
- Compiler: implemented generic functions
- imfilter.q module is updated.
- Redshift: various small fixes (w.r.t breakpoints, function debugging)
- Redshift: added session manager

02/05/2013

- Added support for additional image formats (PPM, PGM, …)
- Added CUDA Memory Enhancements I
- assert() function can now be used from within kernel/device functions
- **device** functions can now have multiple output arguments
- Redshift image viewer has been moved to the Quasar runtime (which means that the extra editing functionality is now available when running via Quasar.exe)
- vecX, ivecX and cvecX can now be used inside classes, with minimal overhead
- imshow() is now able to display 3D data (slice-by-slice browser)
- #pragma force_serial for forcing code to be serially executed (for example code that cannot be parallelized). Works also automatically.
- Windows 32-bit version: switched to largeaddressaware-mode, which allows up to 4 GB of RAM memory to be used (more than 4GB: use the 64-bit version). Note that Quasar Redshift currently only works in 32-bit mode in Windows.
- fread function is now about 10x faster
- Redshift: added the output log pane (for C++/NVCC error messages)
- Redshift: left pane can be hidden by double clicking onto an editor tab
- Redshift immediate window: “print” is no longer required, values are printed automatically

10/04/2013

- Mainly maintenance and bugfixes
- Added libraw 0.15 - allows to read .raw files from digital still cameras. (see Samples/raw_dsc.q)
- Linux installation script (by Dirk)

01/04/2013

- Redshift: dealing with missing recent files

- Redshift imageviewer: added fullscreen button, video quality menu
- Redshift error list: added context menu (with copy to clipboard function)
- Redshift: automatic checking for external file modifications (and reloading documents)
- Redshift: bug reporting is now possible via the menu Help/File a bug, which redirects to UGent hithub!
- implemented CUDA hardware texturing (modifiers hwtex_nearest, hwtex_linear): speed-up for 2D linear interpolation: 30-40% speed-up for 3D linear interpolation: 90-105%
- fixed a bug in the kernel generator (constructor with invalid field name)
- CuMatrix.cs: implemented 1D, 2D and 3D arrays (necessary for texture access)
- implemented 3D control in imshow windows (especially in Redshift)
- Revised the configuration system, making it more transparent. The configuration is stored in the following files:
  - Quasar.config.xml - Quasar compiler
  - Quasar.Runtime.config.xml - Quasar runtime system
  - Quasar.Redshift.config.xml - Quasar Redshift
- Windows: installation is now simplified - it is NO LONGER necessary to copy the dlls from OS_Runtimes\32bit_Windows or OS_Runtimes\64bit_Windows to Quasar root folder. The Quasar runtime automatically detects whether it is run in 32-bit mode or 64-bit mode and selects the appriopriate binary directory. (UPDATE 10/06/2013: in some cases, the GTKSharp dlls still need to be copied)
- linux: executable permissions for the shell scripts are now set automatically
- linux: video recording is now working. IMPORTANT: need to update the LD_LIBRARY_PATH LD_LIBRARY_PATH=$LD_LIBRARY_PATH$:{QUASAR_PATH}/OS_Runtimes/64bit_Ubuntu
- linux: the latest LINUX version of TNGVideoPlayer is now included (see NewestVersion/OS_Runtimes/64bit_Ubuntu)
- Samples/video.q has been updated and now works in linux.
- Updated the installation guide.
- Updated the documentation (a new section on Advanced GPU concepts)

27/03/2013

- 3D FFT/IFFT optimizations (irealfft3(x) = real(ifft3(x)), fft of real data)
- Video recording (MPEG4 format) for OpenGL-based visualization (however not tested in Linux yet)
- Video recording (MPEG4 format) for Redshift image viewer, useful for quickly generating demo videos!
- CUDA engine: improved the memory transfer handling for slice selectors.
- Redshift profiler now indicates per line which percentage of calls make use of the GPU
- Type modifiers 'mirror, 'circular, 'clamped are now also implemented outside kernel functions
- Implemented automatic background compilation (still causes some minor problems in some cases)
- Change of the compiler/runtime settings now causes a debugger reset/recompilation (so that the changes are effective)
- Plots generated using the plot() function can now be saved in PNG (bitmap)/EMF (vector graphics) format
- Standalone compiled EXE's now work under windows, independent of in which directory they are located (Quasar needs to be present however).

28/02/2013

- mainly maintenance and bugfixes.

28/01/2013

- Various performance enhancements in the CPU computation engine. Also affects the CUDA computation engine because both work together. Some CPU programs that heavily rely on matrix multiplications, additions, slice get/set are now 70%-100% faster
- IMPORTANT: the result of the division of two integers is now scalar (before this only holded outside kernel/device functions). This is to solve a well-known source of bugs in C/C++ with respect to division. In case you need the integer division, you can use "int(a/b)". The conversion int(.) will be automatically optimized into the integer division.
- generic types vec[.], mat[.], cube[.] can now directly be used to construct new objects, for example `vec[int](10)`, `cube[int8](64,64,3)`, ...
- the automatic for-loop parallelizer is now activated by default.
- added a new built-in function clamp: clamps the input coordinate between [0,p], i.e. `clamp(x, p) = max(0, min(p, x))`.
- added the "clamped" modifier for vectors, matrices and cubes (vec'clamped, mat'clamped, cube'clamped).
- Redshift: improved the image viewer. It is now possible to "select" individual pixels and see the intensity values
- Redshift: vertical and horizontal split views, document tabs can be dragged
- Redshift: menu to start the Quasar program externally (instead of internally)
- Redshift: definition window: added a Type column with the data type of the variables.
- Redshift: added drag&drop to evaluate expressions

6/01/2013

- Several runtime system fixes/improvements: memory copies from CPU->GPU of >100000 objects are now accumulated into single large batch copies.
- Added C#/C++ interoperability samples folder (see folder Interop_Samples)
  1) A Quasar library to connect to DirectShow (Windows) for camera access
  2) A wrapper for the 2D watershed segmentation algorithm by Johan De Bock.
- A big milestone: included Quasar Redshift, the first IDE for Quasar. Features:
  - Multiple document editing with syntax highlighting and completion lists.
  - Built-in Quasar Spectroscope for debugging and interactive programming
  - Call stack, variable definition, variable watch, breakpoints windows, ...
  - Incremental compilation: significantly decreases overall compilation time
  - Source code parsing, function browsing, comment parsing
  - Ability to break and continue running programs.
  - Integrated OpenGL functionality for fast visualization
  - Improved image viewer
  - Improved profiler with timeline profiling view and special profiler code editor margin.
  - A lot of introspection which helps faster development of Quasar programs.

4/12/2012

- This is mainly a maintainance release. I performed extensive regression testing and found a number of bugs in the compiler (that are now solved).

- Named function arguments are now implemented at compiler level. This facilitates calling functions with a lot of arguments. For example:

  ```
  function [] = do_something(image,color_space="rgb",num_iterations)
  ```

  can be called by explicitly passing the parameter names:

  ```
  do_something(image:=my_image,num_iterations:=10)
  ```

22/11/2012

- The automatic expression optimizer (kernel generator) is now improved (we can now deal with expressions such as A[:,a..b,c]). This means, if you use expressions such as (in rgb2gray):

  ```
  y = 0.2126 * x[:,:,0] + 0.7152 * x[:,:,1] + 0.0722 * x[:,:,2]
  ```

  A kernel function will be generated that performs the same task. This saves a lot of runtime overhead, and only leads to 1 kernel function call (instead of 5 in the above example). By default, the automatic expression optimizer is now enabled. However, it is still possible that some incorrect compilation errors may be given for some programs. Therefore, the expression optimizer can still be disabled by setting COMPILER_AUTO_KERNEL_GENERATION = False (see Quasar.exe.config).

- Missed opportunities for optimizations, are now hidden by default. Actually, the optimization hint system is further improved, but often leads to too many messages printed to the console. It is possible to show the “missed opportunities” gain, by changing the global compilation setting COMPILER_SHOW_MISSED_OPT_OPPORTUNITIES (see Quasar.exe.config).

- Some modifications are made for MAC OS X (however still needs to be tested)

- Improved the kernel error handling. In case of an error (such as integer overflow, out of bounds, error function), the kernel function is immediately terminated.

- Added type checking for the ‘load’ function. For example:

  ```
  [im_out : cube] = load("image.qd")
  ```

  if the type of im_out is not ‘cube’, a runtime error will be generated. Moreover, this construct gives more information to the compiler about the type of ‘im_out’, so that it can generate more optimal code (or enable automatic for loop parallelization for this variable). The same technique can be applied to other functions that have multiple output arguments.

- Multiple GPU systems: it is possible now to select the default GPU device. If your device has two GPUs you can instruct Quasar to use the second GPU, using the -gpu:1 command line option. Generally, the option is “-gpu:[deviceid]”, where the GPU device IDs are listed when running “Quasar -version”.

- Modified the LZMA compression parameters (save function + distributed processing). Compression is now 2x-3x faster, while still having a good compression ratio.

12/11/2012

- Improved the memory management - avoiding memory fragmentation & more efficient copying of data back to the CPU host memory. Note, in case the maximum kernel working set of memory (the maximum amount of memory used by a kernel function at a given time) is larger than ~50% of the GPU memory, some performance degradations due to memory transfers may apply. In future versions, this may further be enhanced, or perhaps a memory profiler will be added.
- Added output arguments for kernel functions, allowing easier outputting of scalars.
- Operations with (small) complex matrices in the CUDA engine are now automatically scheduled to the CPU if this would improve performance. For real matrices this was already done in earlier versions.
- Added serial_do keyword, to perform operations on the CPU (useful for algorithms that are difficult to parallelize).
- CUDA engine - switched to the cuLaunchKernel API functions (instead of the older cuLaunch function). The main benefit is that 3D data grids are now supported. Before there was a problem dealing with matrices of sizes such as 10x400x400
- CPU engine - "emulation" of kernels with "syncthreads" command enabled. Some kernels were not working correctly, because OpenMP does not provide an alternative to __syncthreads of CUDA (#pragma omp barrier cannot be used inside a loop). The previous solution was to run these kernels with only one thread (avoiding multi-core functionality). Now these kernels can also be used in multi-threaded mode.
- Added more rigorous compiler checks - improved the compiler checking systems: errors are only generated after all possible reductions have been applied.
- Improved the parser: by merging parser regular expressions, 10-20% speedup is obtained.
- Added Visual C++ 10 (VS 2010) to the list of supported compilers (msvc_script.bat)
- when Quasar runs in CPU mode, the NVCC compiler is not invoked. This allows compilation on computers on which NVCC is not installed (only CPU engine then).
- Implemented the feature to synchronize imshow windows to each other (e.g. zoom and scroll position). This makes it easier to compare different images.
- Added check for compute capability 1.3 (which is now the minimum required compute capability to run Quasar). For 1.3, atomic operations using floating point numbers are implemented using the atomicCAS function.
- Added some I/O functions, such as fprintf, fscanf, sscanf, dir
- Added experimental -turbo command line option. This option increases the voltage of the GPUs of the system by 33% :-), allowing faster execution of kernels (for primal dual restoration algorithm this is even up to +110% faster). This will later be the default option - however please be careful and monitor the GPU temperature when in use:)
- Added "natural docs" based documentation.

19/10/2012

- Kernel/device functions: the types of all input/output arguments should now be specified. Note that this changed from older versions of Quasar, in which the default type was 'scalar'.
- major change of the type system:

1) added parametric types vec[XX], mat[XX], cube[XX]
2) added classes 'type T : class'
3) added type definitions 'type fn : [(??,??)->??]'
4) added pointers and pointer types: 'type T_ptr : ˆT'
5) added integer types int8, int16, int32, int64, uint8, uint16, uint32, uint64
6) types for **device** and **kernel** functions: [**device** (scalar) -> (scalar)], [**kernel** (mat, ivec2) -> ()]

- Implemented interop with OpenGL vertex buffers (e.g. for surface plots, 3D rendering)
- Improved CUDA memory manager: matrices are now automatically copied back to system memory, in case of a potential out-of-memory. This allows all Quasar programs to work with larger amounts of memory than before.
- (internal) unification of **kernel**, **device** and regular functions: **device** functions can now be called from host code.
- device functions can now be passed as arguments of kernel functions (or as fields of classes)
- auto for-loop parallelizer can now generate code that uses **device** functions
- parser: recognizes binary numbers (11101b), octal numbers (1532o) and hexadecimal numbers (0EFh)
- added marshaling of classes, pointers and matrices to device memory and back.
- protection for memory leaks due to circular references (however: not yet implemented for cell matrices)
- cell matrices cannot be passed to kernel/device functions, but the new parametric matrices (e.g. cube[mat] can). The compiler needs full type information to allow the variables to be passed.
- improved compiler type checking & improved compiler error messages.
- type aliases: ivec, imat and icube are aliases for respectively vec[int], mat[int], cube[int]
- implemented device/kernel function closures
- added integer saturation/overflow checking
- types uint32, int64, uint64 are now mapped onto scalar inside kernel functions
- added first version of the inttypes.q library
- old cellmatrices: functionality is now replaced by the new vec[??],mat[??] and cube[??] types
- load/save works with most of the new datatypes (except function pointers)
- imshow can render images of types uint8, uint16, uint32 directly (either via GDI+ or OpenGL)

18/09/2012

- support for double precision floating point format (on a global level, see doc. Section 2.2.1.)
- Quasar spectroscope - the new command line Quasar computation/debugger tool.
- improved terminal session (e.g. ssh, rdesktop) detection. OpenGL is automatically disabled in this case.
- added 'checked modifier, see Reference Manual section 2.4.1.
- updated the documentation

12/09/2012

- added function closures
- added OpenGL support
- updated the Quasar documentation
- unified functions and lambda expressions (functions can be passed to functions with lambda expression arguments and vice versa)

- added compiler reduction safety level settings (see documentation)
- added function overloading
- added partial support for symbolic operations (symbolic keyword)
- added generic function types [??->??]
- plots: added xlabel, ylabel, xlim, ylim functions
- improved the parser: the syntax f(x)(y) (partial evaluation) is now working
- integration TNGVideoPlayer in Quasar
- vidplay function working in Windows 32-bit
- OpenGL rendering: imshow function