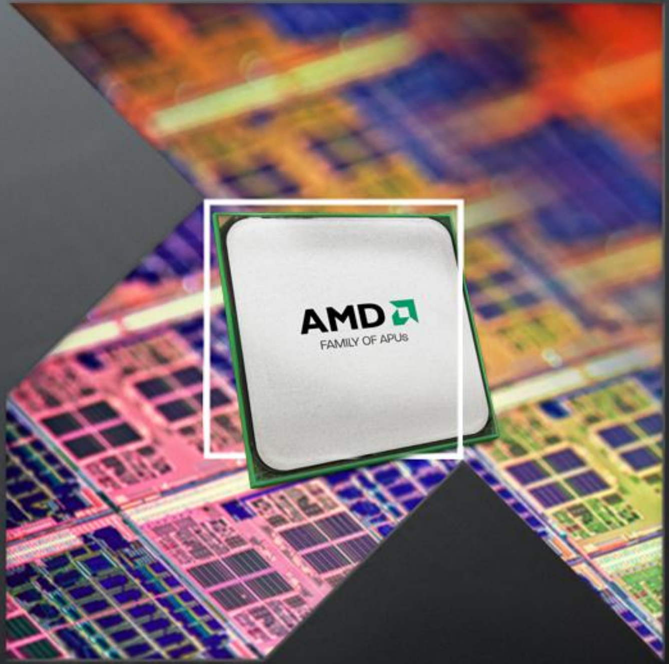


Supporting Computer Vision through High Performance GPU Programming

Dong Ping Zhang
12 Jan, 2013
AMD Research

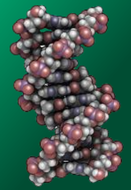
Most parallel code runs on CPUs designed for scalar workloads

WASTES POWER

APPLICATION AREAS WITH ABUNDANT PARALLEL WORKLOADS

Natural UI & Gestures

Touch, gesture, and voice



Biometric Recognition

Secure, fast, accurate: face, voice, fingerprints

Augmented Reality

Superimpose graphics, audio, and other digital information as a virtual overlay

Content Everywhere

Content from any source to any display seamlessly

Beyond HD Experiences

Streaming media, new codecs, 3D, transcode, audio

AV Content Management

Searching, indexing and tagging of video & audio. multimedia data mining

ARCHAEOLOGY?

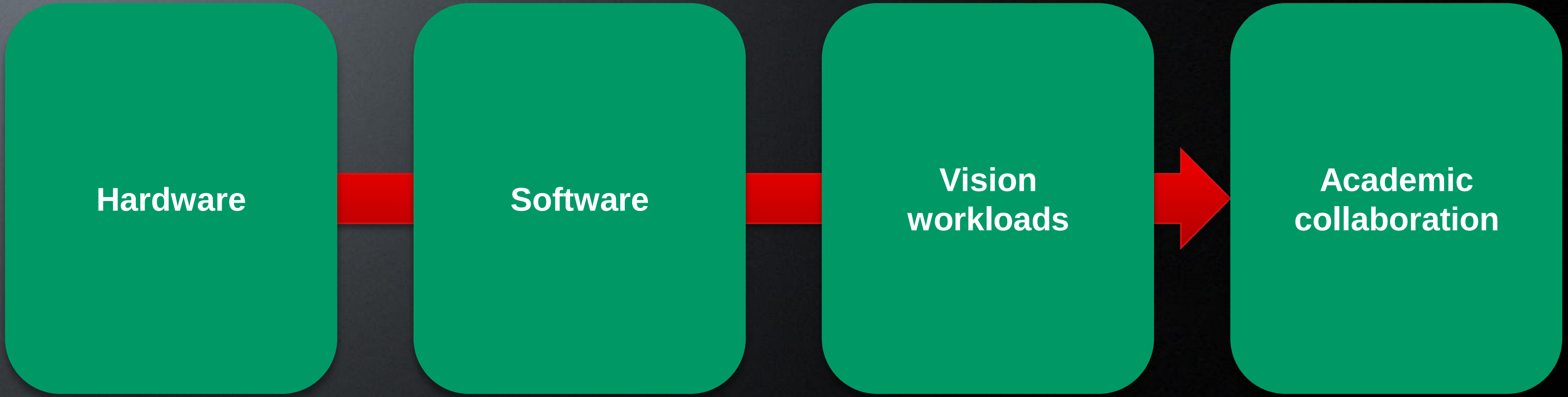
Tintagel Castle and Fountains Abbey

Source: English Heritage

Knossos

Source: wikipedia

WHERE AM I GOING WITH THIS?



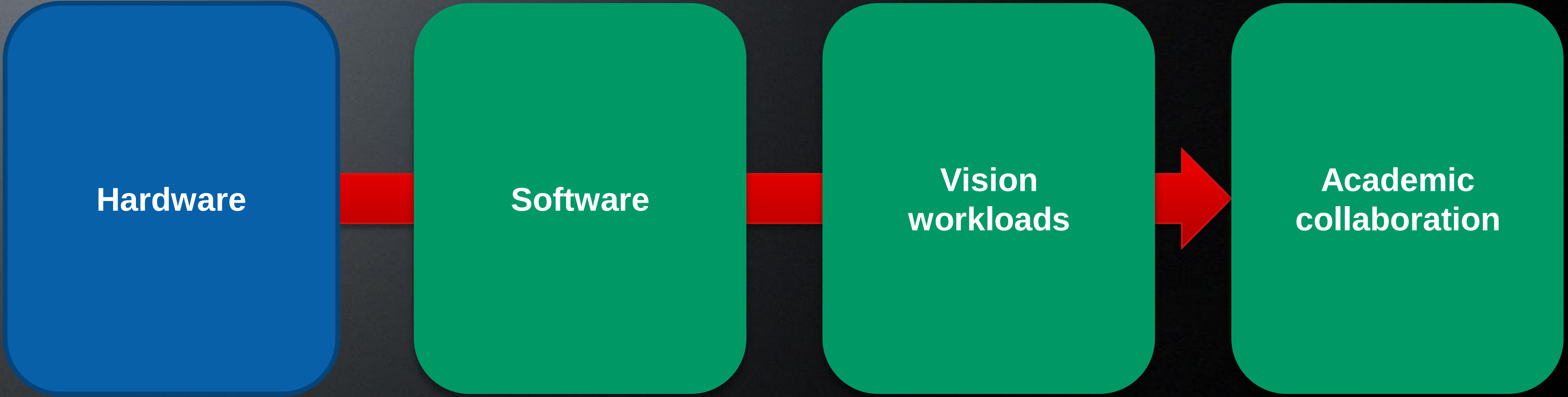
Hardware

Software

Vision workloads

Academic collaboration

WHERE AM I GOING WITH THIS?

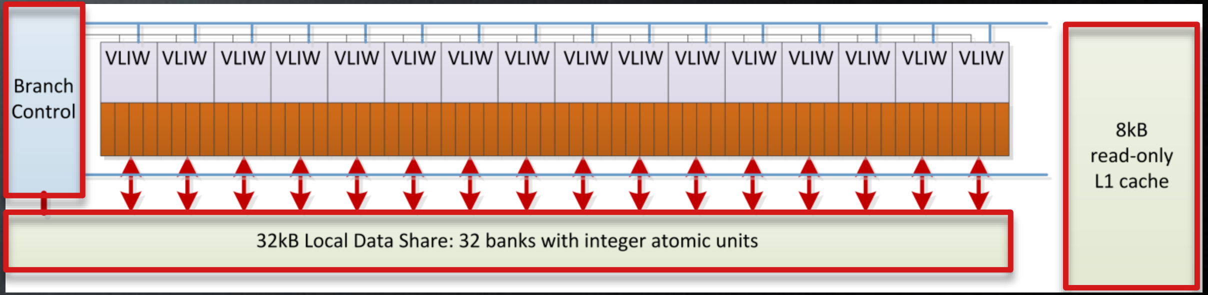


AMD GPU and APU Architecture

The HD7970 and Graphics Core Next

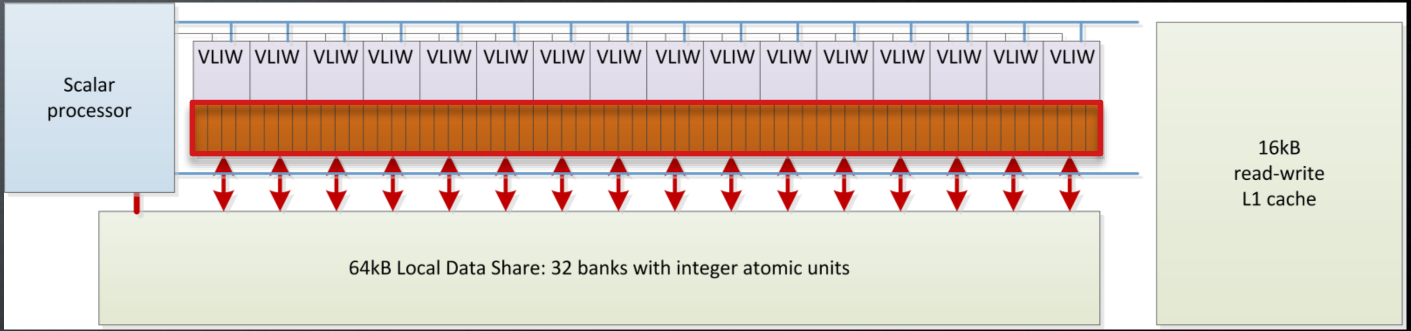
THE SIMD CORE

- The SIMD unit on the old Radeon architectures had a branch control but full scalar execution was performed globally



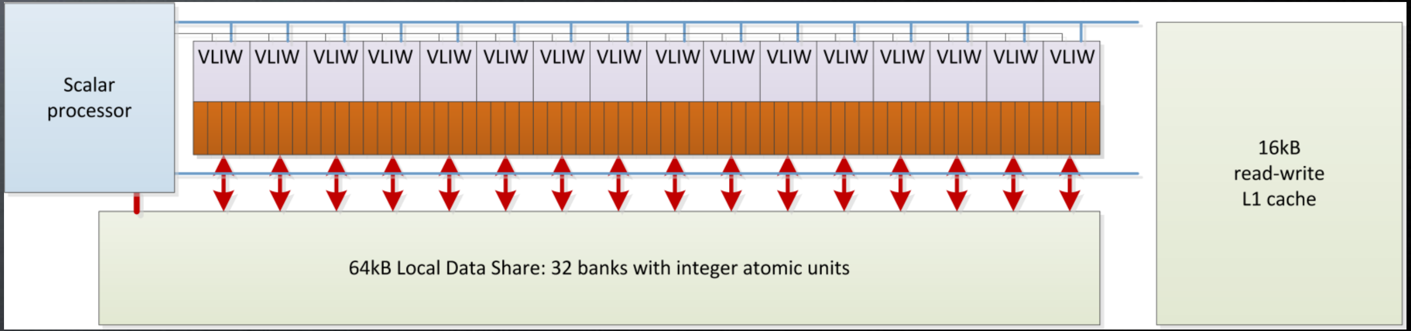
THE SIMD CORE

- On the HD7970 we have a full scalar processor and the L1 cache and LDS have been doubled in size
- Then let us consider the VLIW ALUs



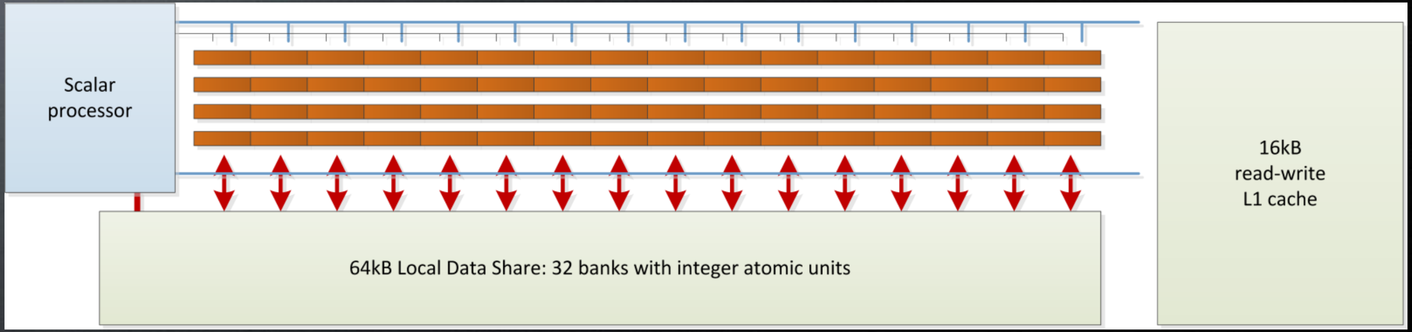
THE SIMD CORE

- Remember we could view the architecture two ways:
  - An array of VLIW units
  - A VLIW cluster of vector units



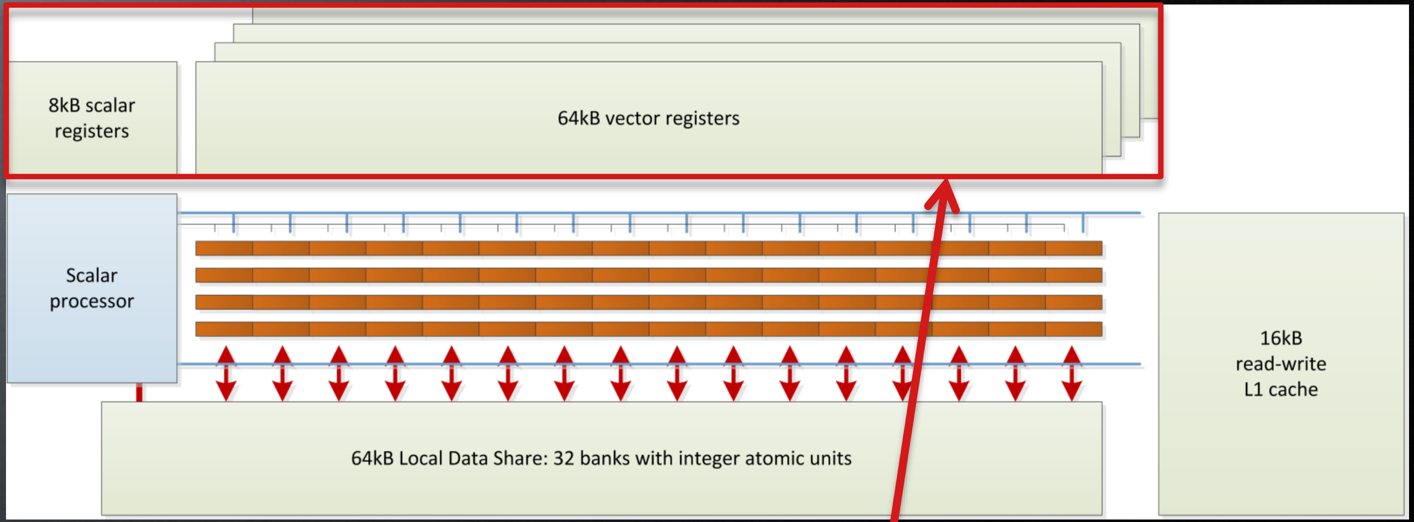
THE SIMD CORE

- Now that we have a scalar processor we can dynamically schedule instructions rather than relying on the compiler
- No VLIW!



- The heart of Graphics Core Next:
  - A scalar processor plus four 16-wide vector units
  - Each lane of the vector, and hence each IL work item, is now scalar

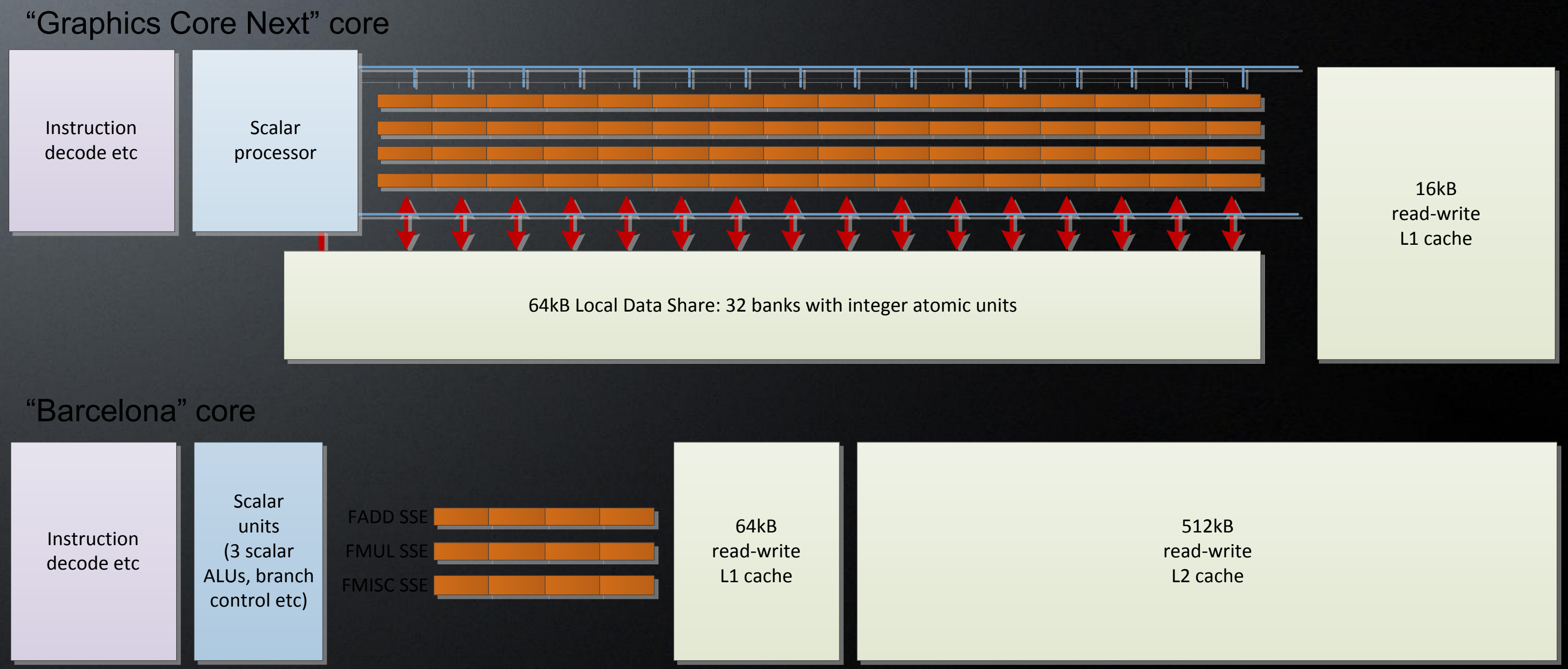
THE SIMD CORE



- The scalar core manages a large number of threads
  - Each thread requires its set of vector registers
  - Significant register state for both scalar and vector storage
  - 10 waves per SIMD, 40 waves per CU (core), 2560 work items per CU, 81920 work items on the HD7970

FAMILIAR?

- If we add the frontend of the core…



“Graphics Core Next” core
Instruction decode etc
Scalar processor
64kB Local Data Share: 32 banks with integer atomic units
16kB read-write L1 cache
“Barcelona” core
Instruction decode etc
Scalar units (3 scalar ALUs, branch control etc)
FADD SSE
FMUL SSE
FMISC SSE
64kB read-write L1 cache
512kB read-write L2 cache

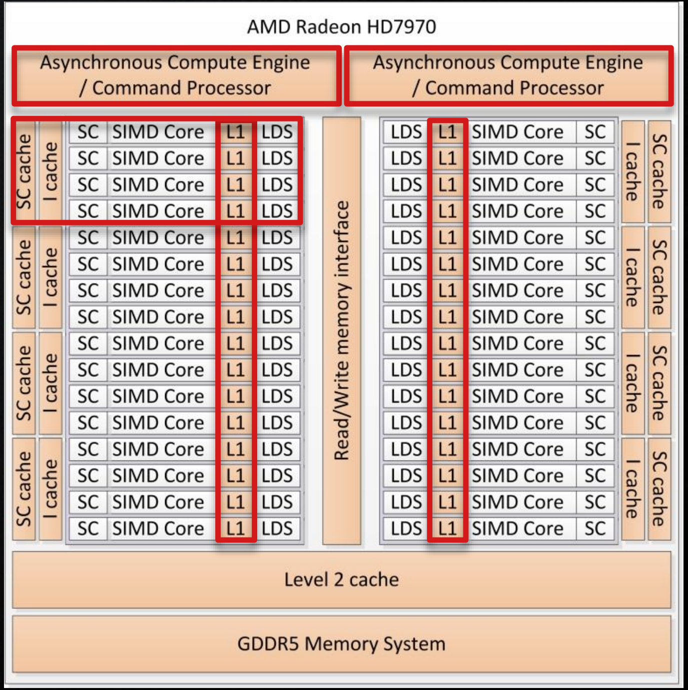
AMD RADEON HD7970 - GLOBALLY

- Two command processors
  - Capable of processing two command queues concurrently

- Full read/write L1 data caches

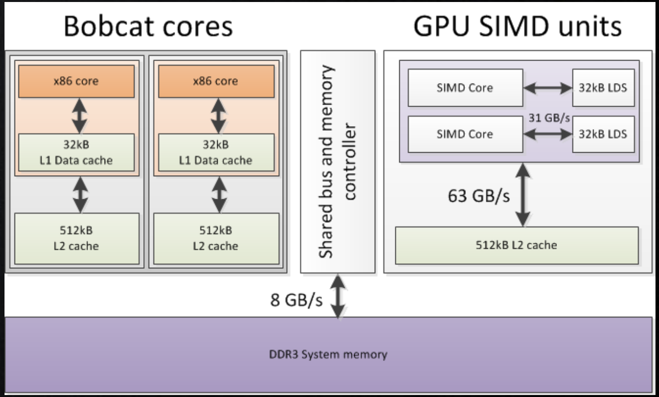
- SIMD cores grouped in fours
  - Scalar data and instruction cache per cluster
  - L1, LDS and scalar processor per core

- Up to 32 cores / compute units


AMD Radeon HD7970
Asynchronous Compute Engine / Command Processor
Asynchronous Compute Engine / Command Processor
SC cache
I cache
SC
SIMD Core
L1
LDS
Read/Write memory interface
Level 2 cache
GDDR5 Memory System

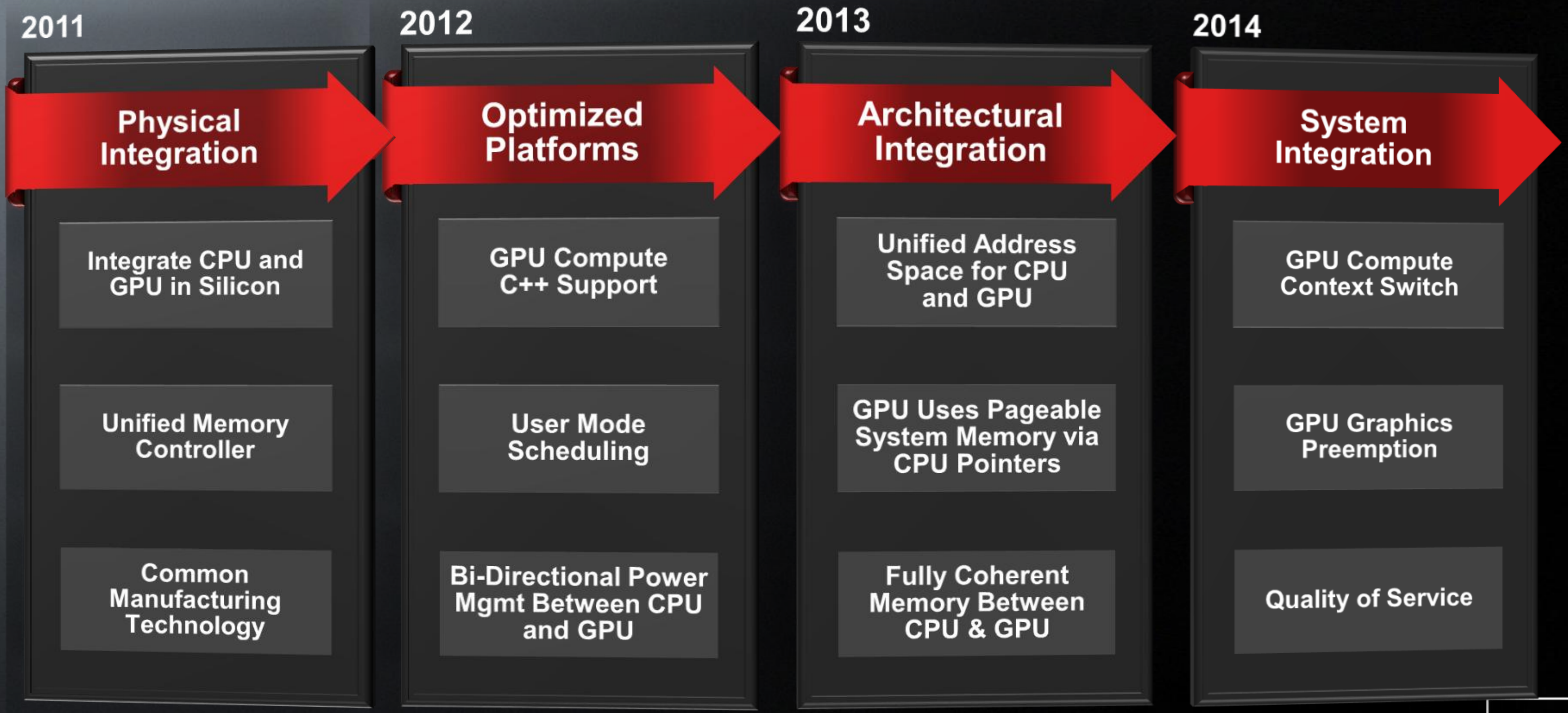
APU: ACCELERATED PROCESSING UNIT

- The APU has arrived and it is a great advance over previous platforms
- Combines scalar processing on CPU with parallel processing on the GPU and high bandwidth access to memory
- How do we make it even better going forward?
  - – Easier to program
  - – Easier to optimize
  - – Easier to load balance
  - – Higher performance
  - – Lower power



APU E350 (2011)

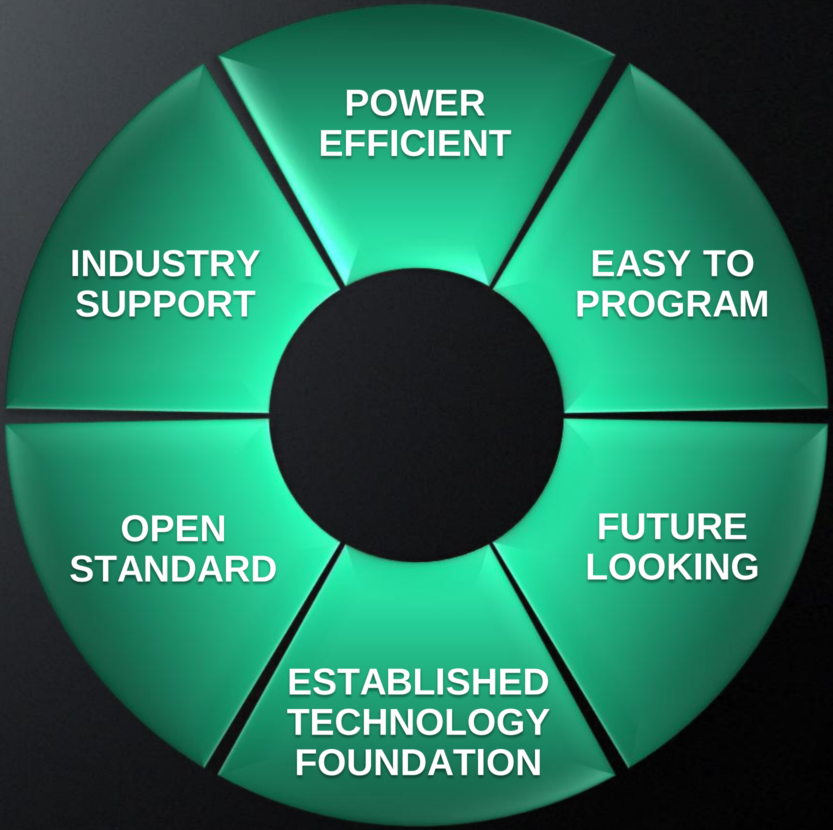
HETEROGENEOUS SYSTEM ARCHITECTURE ROADMAP



2011	2012	2013	2014
Physical Integration	Optimized Platforms	Architectural Integration	System Integration
Integrate CPU and GPU in Silicon	GPU Compute C++ Support	Unified Address Space for CPU and GPU	GPU Compute Context Switch
Unified Memory Controller	User Mode Scheduling	GPU Uses Pageable System Memory via CPU Pointers	GPU Graphics Preemption
Common Manufacturing Technology	Bi-Directional Power Mgmt Between CPU and GPU	Fully Coherent Memory Between CPU & GPU	Quality of Service

HETEROGENEOUS SYSTEM ARCHITECTURE

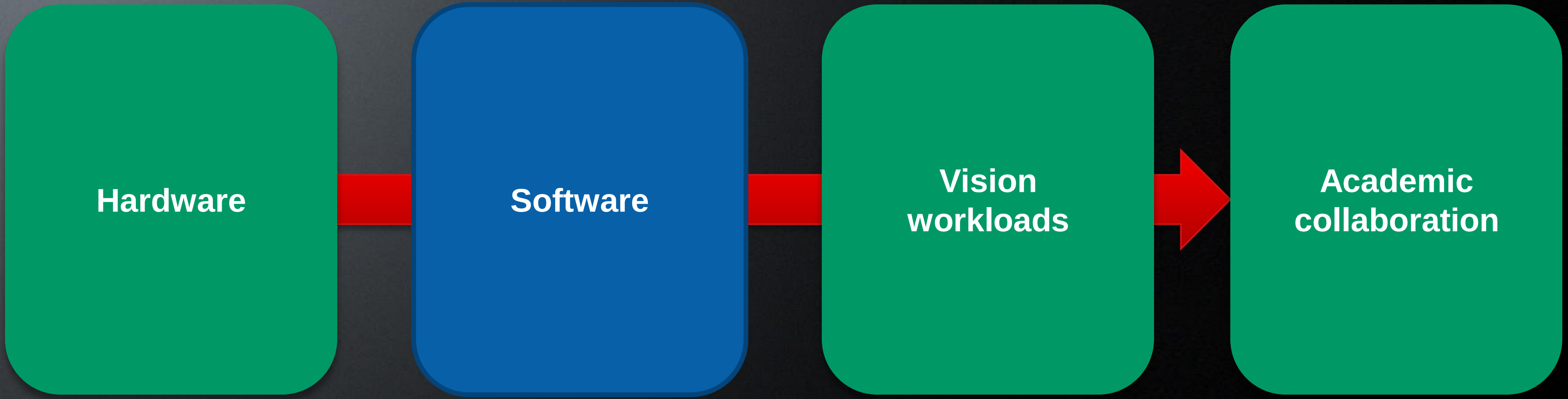
Brings All the Processors in a System into Unified Coherent Memory



HETEROGENEOUS SYSTEM ARCHITECTURE – AN OPEN PLATFORM

- Open Architecture, published specifications
  - HSAIL virtual ISA
  - HSA memory model
  - HSA system architecture
- ISA agnostic for both CPU and GPU
- HSA Foundation formed in June 2012
- Inviting academic partners to join us, in all areas
  - Hardware design
  - Operating Systems
  - Tools and Middleware
  - Applications

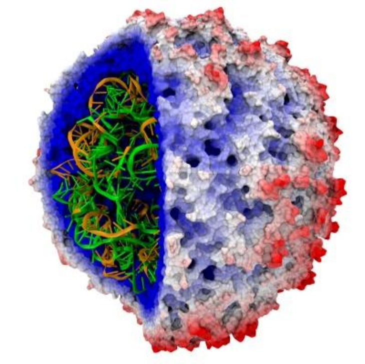
WHERE AM I GOING WITH THIS?



AMD Software Infrastructure

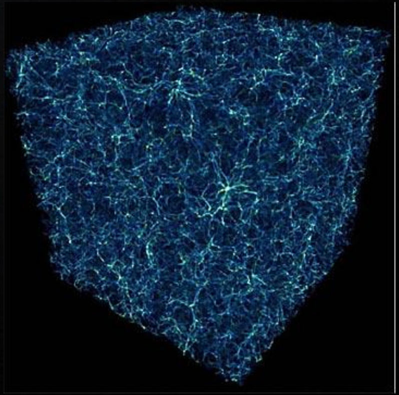
OPENCL™ TODAY

- OpenCL
  - Open development platform for multi-vendor heterogeneous architectures
  - Broad industry support: Created by architects from AMD, Apple, IBM, Intel, Nvidia, Sony, etc.
  - It provides C API and an extended subset of C99 kernel language.
  - Excellent performance, but programming can be longwinded and difficult



Molecular Dynamics
BUDE, NMAD, Folding@Home

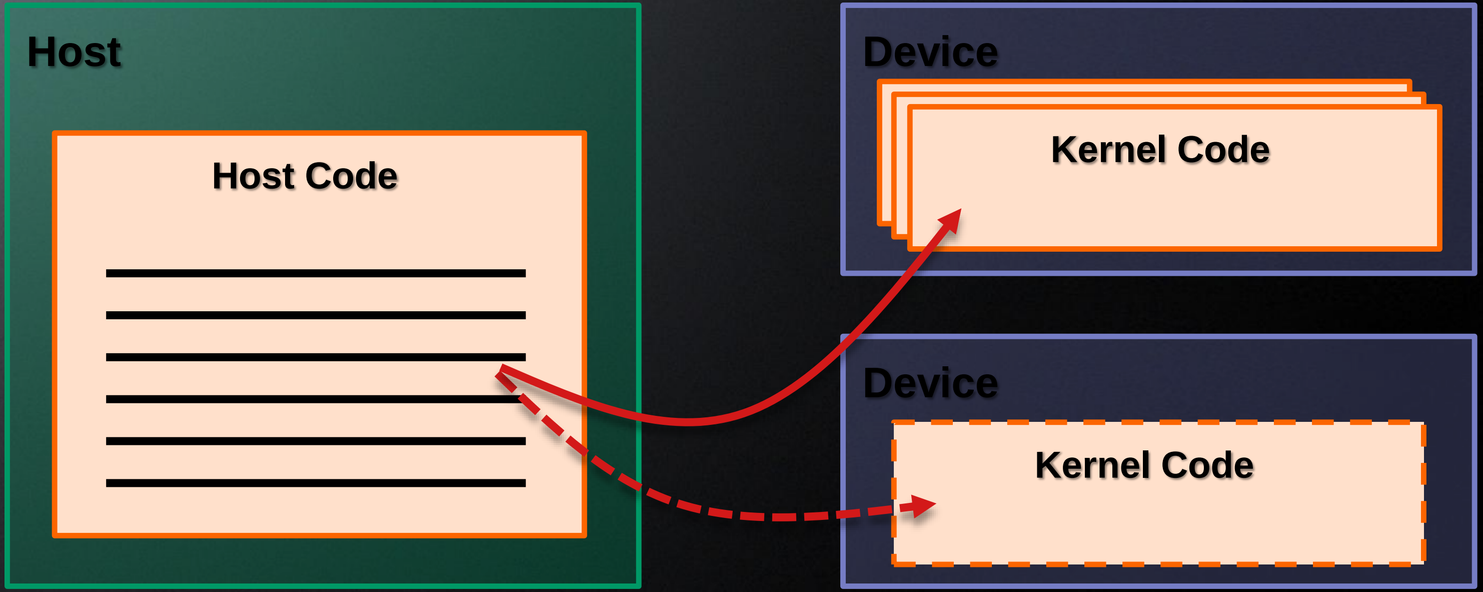
Ray Tracing
LuxRenderer



Search for missing matter in the Universe

HOST AND DEVICE MODEL

- OpenCL™ uses a host and device model
  - Host code dispatches kernel code to the devices



EXAMPLE – VECTOR ADDITION (HOST PROGRAM)

```
// create the OpenCL context on a GPU device
cl_context = clCreateContextFromType(0, CL_DEVICE_TYPE_GPU, NULL, NULL, NULL);

// get the list of GPU devices associated with context
clGetContextInfo(context, CL_CONTEXT_DEVICES, 0, NULL, &cb);
devices = malloc(cb);
clGetContextInfo(context, CL_CONTEXT_DEVICES, cb, devices, NULL);

// create a command-queue
cmd_queue = clCreateCommandQueue(context, devices[0], 0, NULL);
```

```
// allocate the buffer memory objects
memobjs[0] = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sizeof(cl_float)*n, srcA, NULL);
memobjs[1] = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_
memobjs[2] = clCreateBuffer(context, CL_MEM_WRITE_ONLY, sizeof(
```

```
// create the program
program = clCreateProgramWithSource(context, 1, &program_source
```

```
// build the program
err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
```

```
// create the kernel
kernel = clCreateKernel(program, “vec_add”, NULL);

// set the args values
err  = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *) &memobjs[0]);
err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), (void *) &memobjs[1]);
err |= clSetKernelArg(kernel, 2, sizeof(cl_mem), (void *) &memobjs[2]);

// set work-item dimensions
global_work_size[0] = n;
```

```
// execute kernel
err = clEnqueueNDRangeKernel(cmd_queue, kernel, 1, NULL, global_work_size, NULL, 0, NULL, NULL);
```

```
// read output array
err = clEnqueueReadBuffer(cmd_queue, memobjs[2], CL_TRUE, 0, n*sizeof(cl_float), dst, 0, NULL, NULL);
```

Define platform and queues

s

Same boiler-plate code across virtually every OpenCL host code

Create and setup kernel

Execute the kernel

Read results on the host

OPENCL™ C++ HOST API CODE FOR VECTOR ADD

```
std::function<Event (const EnqueueArgs&, Buffer, Buffer, Buffer)> vadd =
            make_kernel<Buffer, Buffer, Buffer>(Program(program_source), “vadd”);

memobj[0] = Buffer (CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,  izeof(float) * n, srcA);
memobj[1] = Buffer (CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sizeof(    t) * n, srcB);
memobj[2] = Buffer (CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sizeof(float)

vadd(EnqueueArgs(NDRange(n)), memobj[0], memobj[1], memobj[2]));

enqueu   dBuffer(memobj[2], CL_TRUE, sizeof(float) * n, dest);
```

Program automatically created and compiled

Defaults, no need to reference context, command queue

No clRelease*XXX* cleanup code required

OPENCL C++ INTERFACE AND KERNEL LANGUAGE EXTENSION

- Khronos has defined a common C++ header file containing a high level interface to OpenCL
  - Common defaults for the platform and command-queue
  - Simplify basic API by parameterizing through template types
  - Maintain object lifetimes through constructors and destructors
  - Support function-like kernel dispatch

- OpenCL C++ kernel language extension
  - Support static C++ language features
  - Support templating and overloading
  - Innovative integration with OpenCL C address spaces

- Make it easier to create applications for heterogeneous platforms.

CODE SIZE REDUCTION

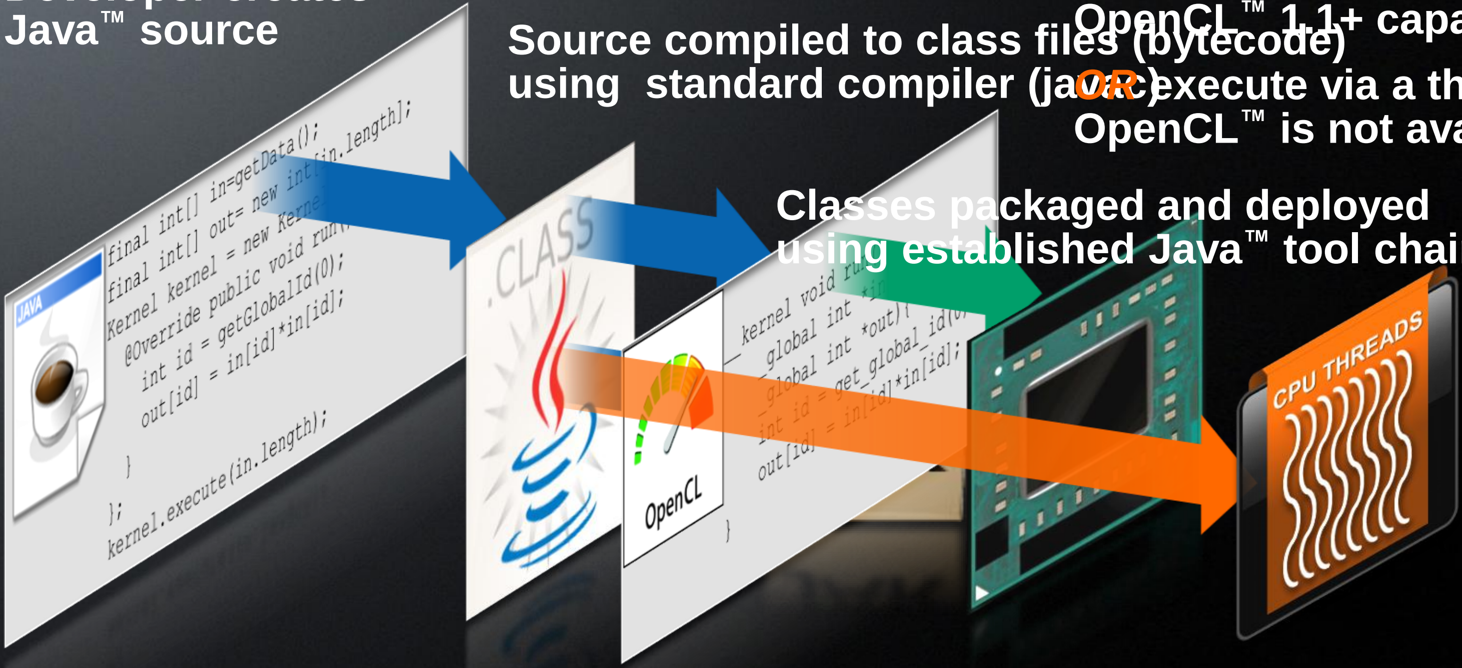
- Sometimes substantial reduction in code size:

Application	C lines	C++ lines	Reduction
Vector addition	268	140	47.7%
Pi computation	306	166	45.8%
Ocean simulation	1386	533	61.5%
Particle simulation	733	601	18.0%
Radix sort	627	593	5.4%

- Depending on the complexity, you may not see this in real use cases. But for beginners, the simplification of the program makes a difference.

JAVA ENABLEMENT BY APARAPI

Aparapi = Runtime capable of converting Java™ bytecode to OpenCL™



Developer creates Java™ source

Source compiled to class files (bytecode) using standard compiler (javac)

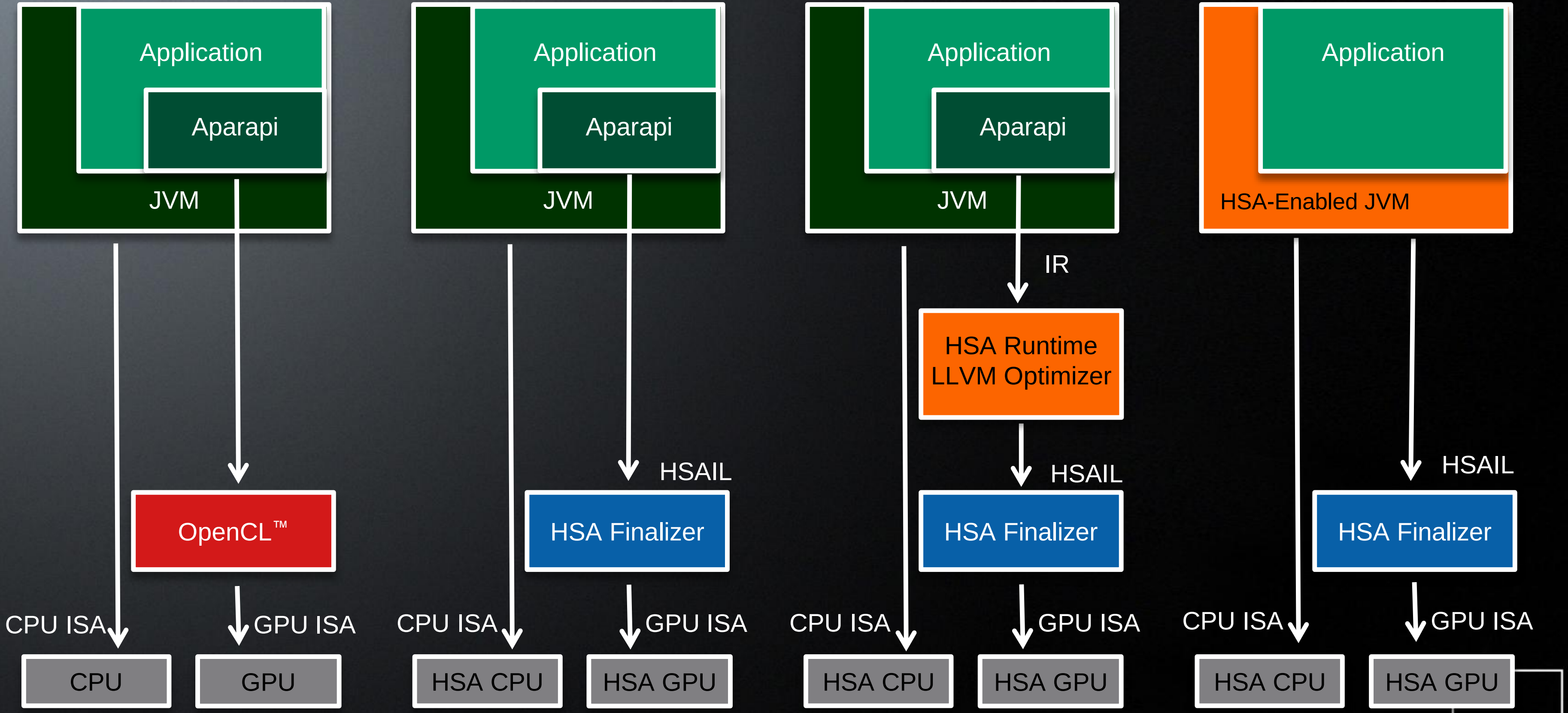
Classes packaged and deployed using established Java™ tool chain

For execution on any OpenCL™ 1.1+ capable device **OR** execute via a thread pool if OpenCL™ is not available

```
final int[] in=getData();
final int[] out= new int[in.length];
Kernel kernel = new Kerne
  @Override public void run(
    int id = getGlobalId(0);
    out[id] = in[id]*in[id];
  }
};
kernel.execute(in.length);
```

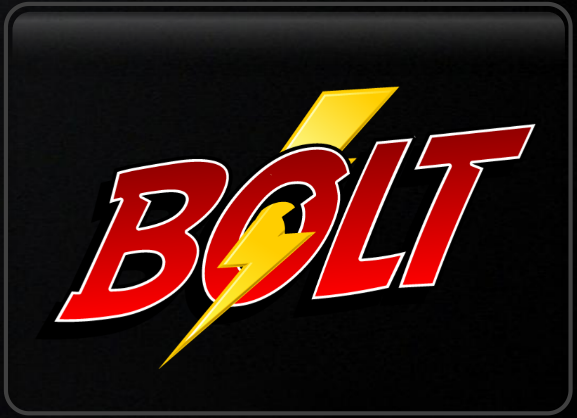
```
__kernel void ru
  __global int *i
  __global int *out){
  int id = get_global_id(0
  out[id] = in[id]*in[id];
}
```

JAVA AND APARAPI HSA ENABLEMENT ROADMAP

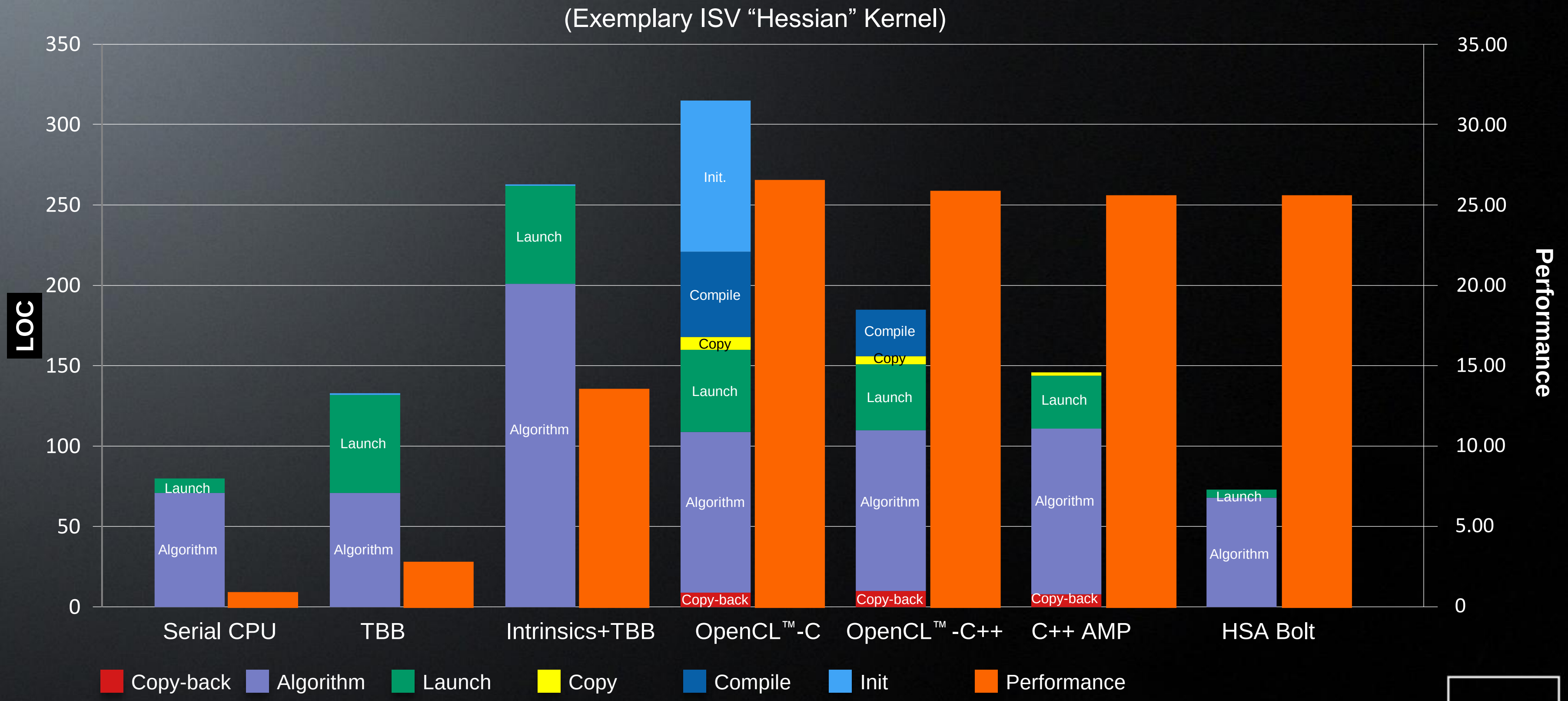


INTRODUCING HSA BOLT – PARALLEL PRIMITIVES LIBRARY FOR HSA

- Easily leverage the inherent power efficiency of GPU computing
  - Common routines such as scan, sort, reduce, transform
  - More advanced routines like heterogeneous pipelines
  - Bolt library works with OpenCL or C++ AMP
- Enjoy the unique advantages of the HSA platform
  - Move the computation, not the data
  - Use appropriate computation resource (CPU or GPU or others)
- Finally a single source code base for the CPU and GPU!
  - Scientists can focus on innovations!
- Bolt preview available in AMD APP SDK 2.8.



LINES-OF-CODE AND PERFORMANCE FOR DIFFERENT PROGRAMMING MODELS



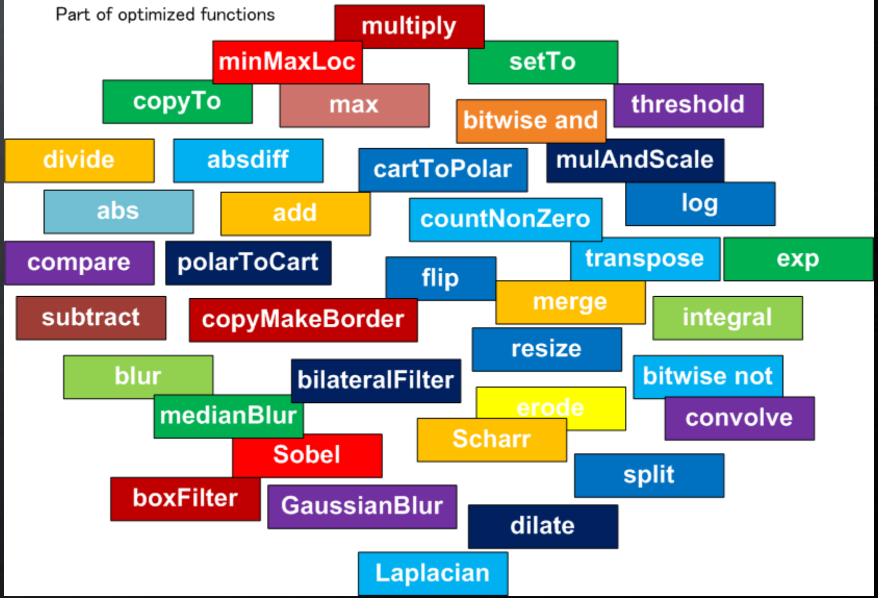
AMD A10-5800K APU with Radeon™ HD Graphics – CPU: 4 cores, 3800MHz (4200MHz Turbo); GPU: AMD Radeon HD 7660D, 6 compute units, 800MHz; 4GB RAM.
Software – Windows 7 Professional SP1 (64-bit OS); AMD OpenCL™ 1.2 AMD-APP (937.2); Microsoft Visual Studio 11 Beta

OPENCL MODULE FOR OPENCV LIBRARY

- Based on OpenCV 2.4, first OCL module released in Nov 2012.
- Implement and maintain an OpenCL module that is optimised for AMD GPU and APU platforms, including utility functions, low-level vision primitives, and high-level algorithms.
- Support any OpenCL 1.1 compatible device, tested on AMD's, Intel's and NVIDIA's GPU.
- Designed as a host-level API and device-level kernels.
- Requirement: OpenCL SDK, AMD FFT and BLAS library
- No prior knowledge of OpenCL required.

OPENCL MODULE FOR OPENCV LIBRARY

- Over 80 kernels are provided with OCL module in 2.4.9 release.
- OCL module can run on NVIDIA, Intel and AMD GPUs without any modification.



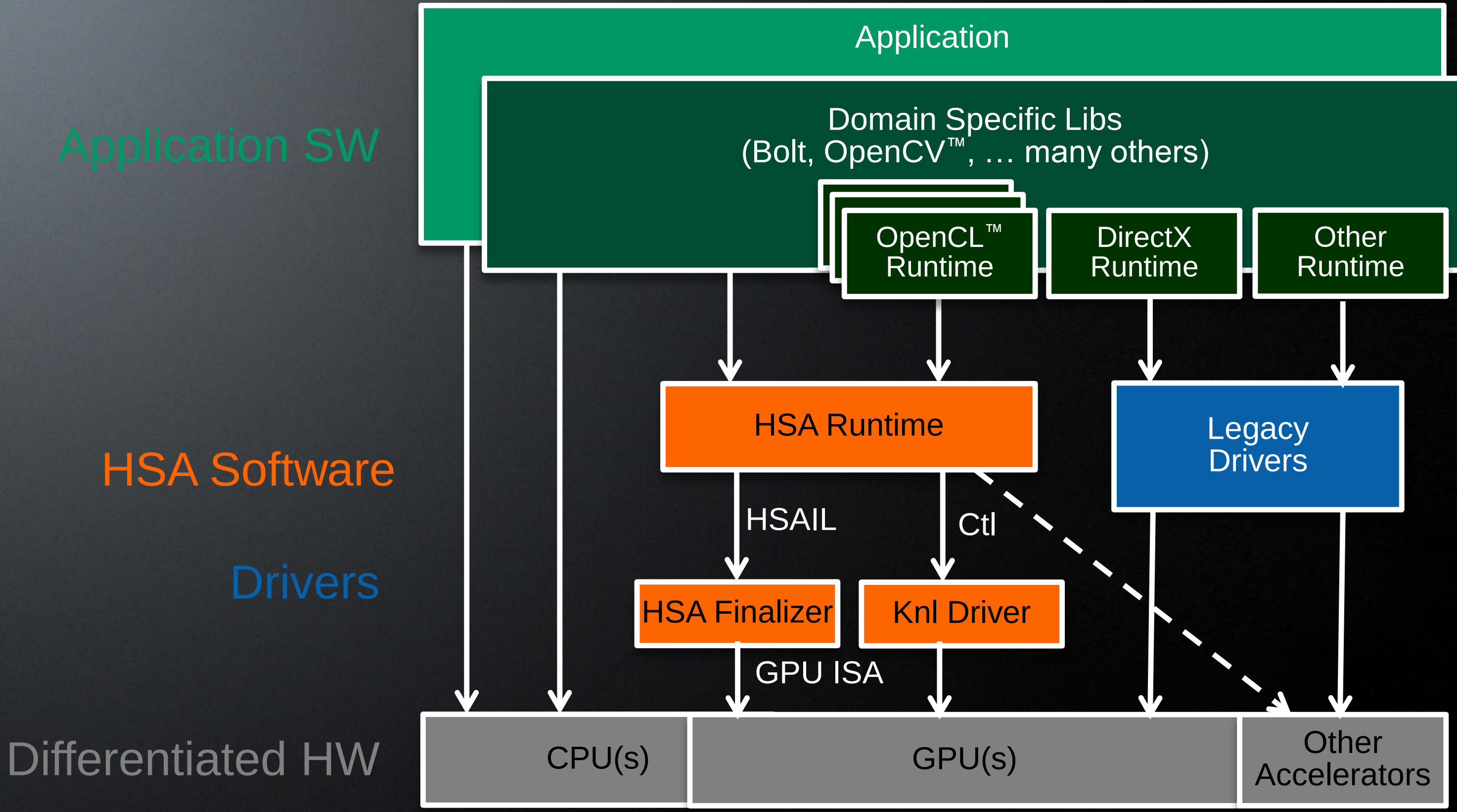
Part of optimized functions

multiply
minMaxLoc
setTo
copyTo
max
threshold
bitwise and
divide
absdiff
mulAndScale
cartToPolar
log
abs
add
countNonZero
compare
polarToCart
transpose
exp
flip
merge
subtract
copyMakeBorder
integral
resize
blur
bitwise not
bilateralFilter
erode
medianBlur
convolve
Scharr
Sobel
split
boxFilter
GaussianBlur
dilate
Laplacian

OTHER LIBRARIES AND TOOLS

- APPML, contains FFT and BLAS functions, primarily targeting AMD GPUS and APUs.
  - User specifies problem parameters through library API
  - Kernel generator creates tailored OpenCL kernels
  - Kernels are dispatched for execution
- APP Kernel Analyzer
  - static analysis tool to compile, analyse and disassemble OCL kernel for GPUs
- APP profiler
  - performance analysis tool that gathers data from OCL runtime and GPUs during the execution.
- Code Analyst
- gDEBugger, an OpenCL and OpenGL debugger and memory analyser
- GPUPerfAPI
  - a library can be integrated directly into your application for accessing GPU performance counters.
- CodeXL, offers GPU debugging, CPU and GPU profiling, static OpenCL kernel analysis capabilities.

HSA SOLUTION STACK

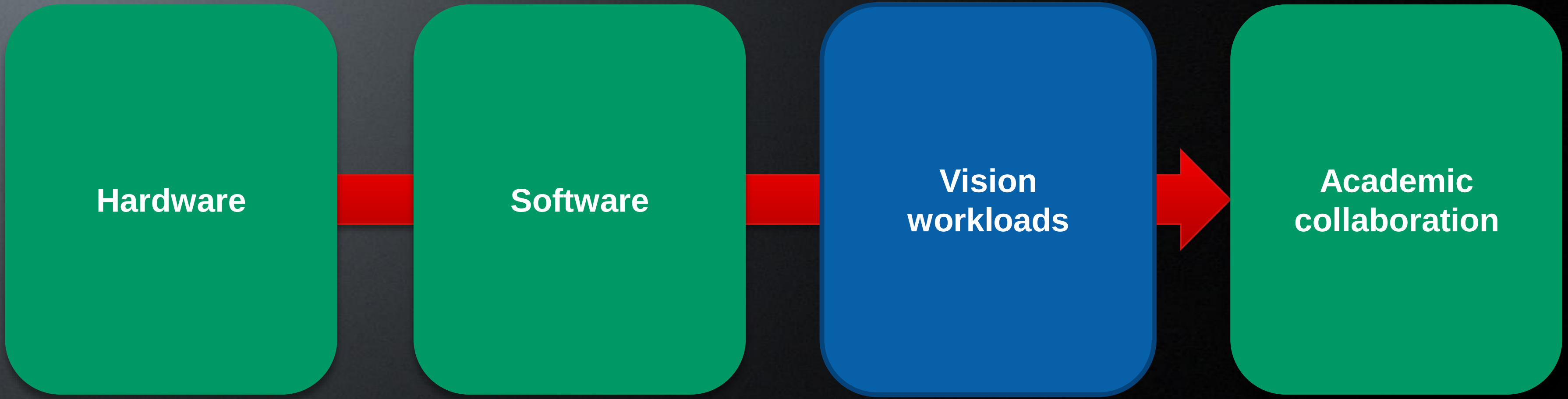


AMD'S OPEN SOURCE COMMITMENT TO HSA

- We will open source our linux execution and compilation stack
  - Jump start the ecosystem
  - Allow a single shared implementation where appropriate
  - Enable university research in all areas

Component Name	AMD Specific	Rationale
HSA Bolt Library	No	Enable understanding and debug
OpenCL HSAIL Code Generator	No	Enable research
LLVM Contributions	No	Industry and academic collaboration
HSA Assembler	No	Enable understanding and debug
HSA Runtime	No	Standardize on a single runtime
HSA Finalizer	Yes	Enable research and debug
HSA Kernel Driver	Yes	For inclusion in linux distros

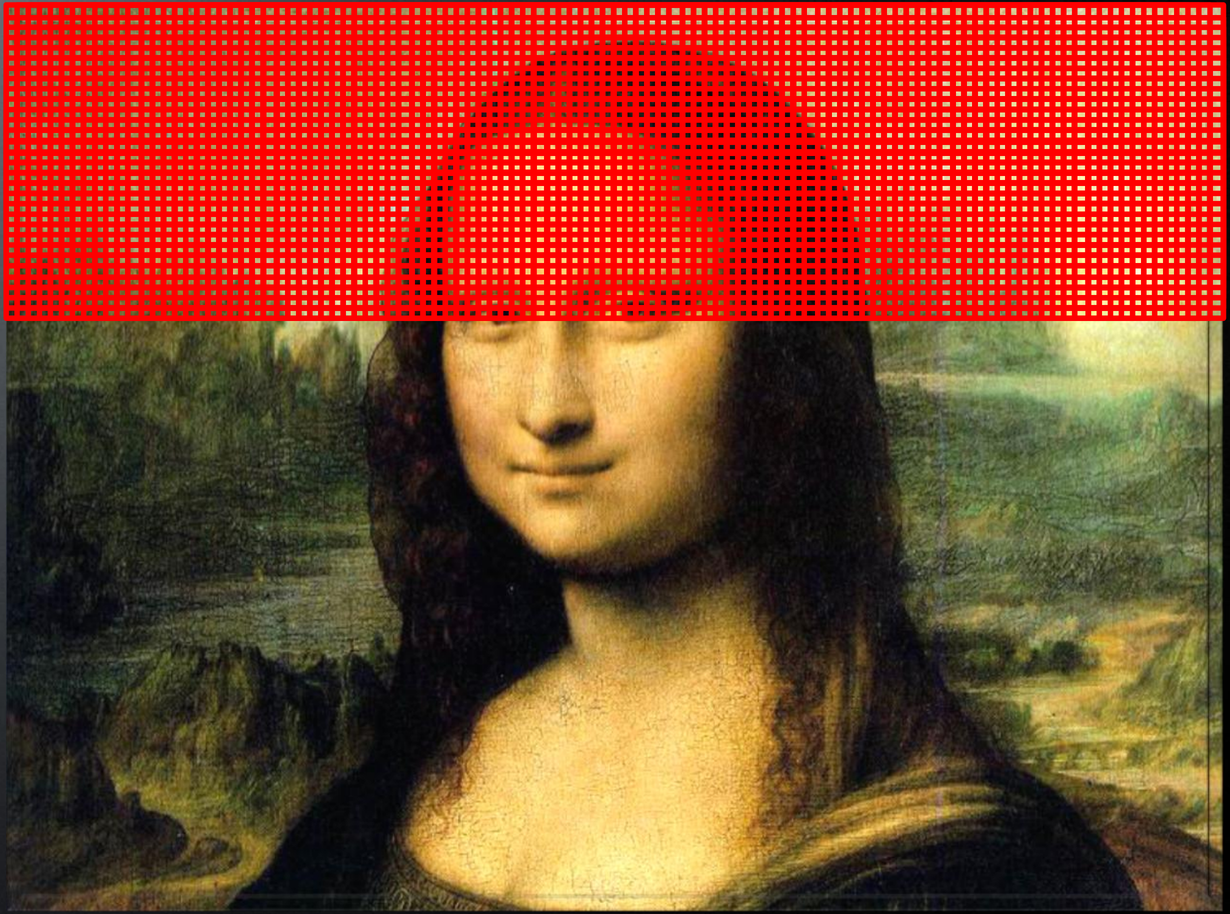
WHERE AM I GOING WITH THIS?



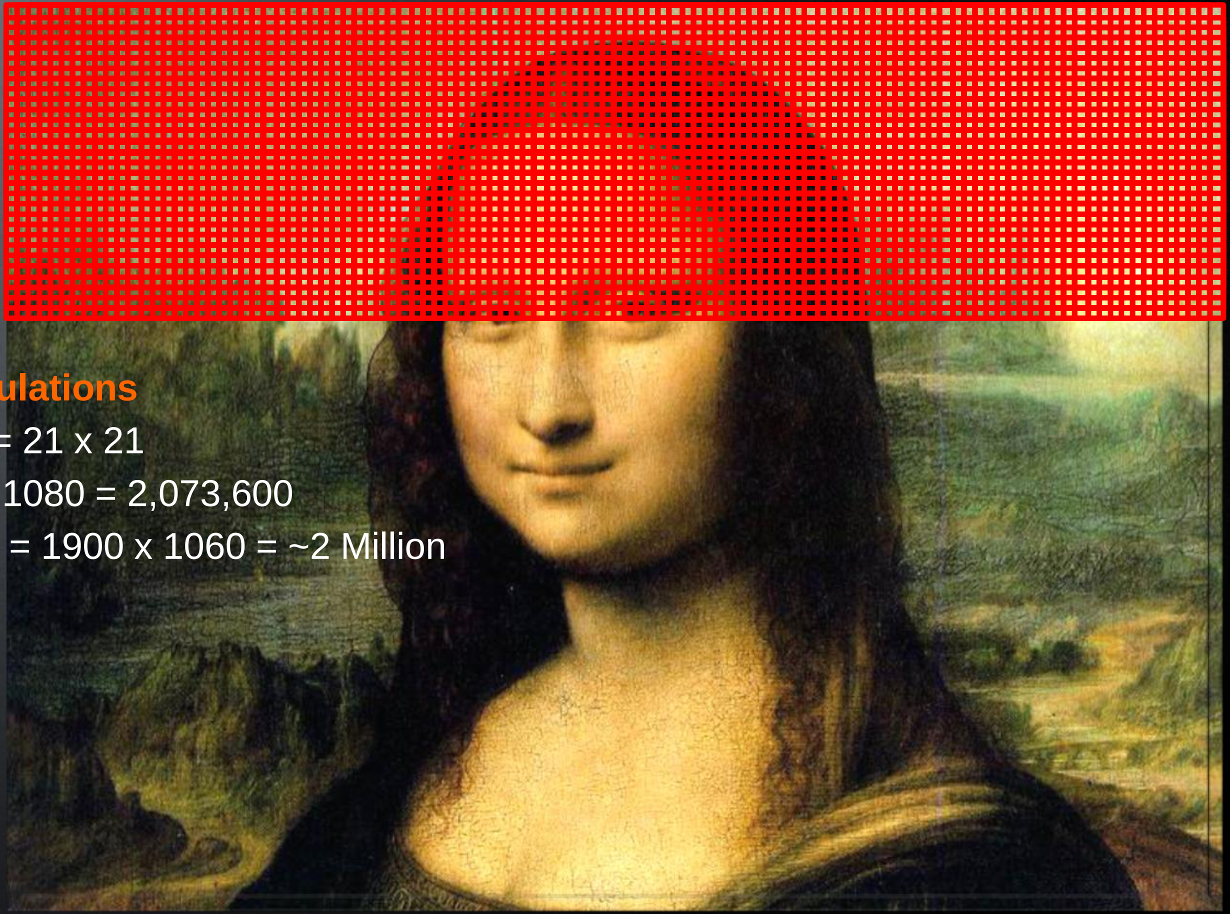
Example ISV workload: Haar Face Detection

Cornerstone Technology for Computer Vision

LOOKING FOR FACES IN ALL THE RIGHT PLACES



LOOKING FOR FACES IN ALL THE RIGHT PLACES



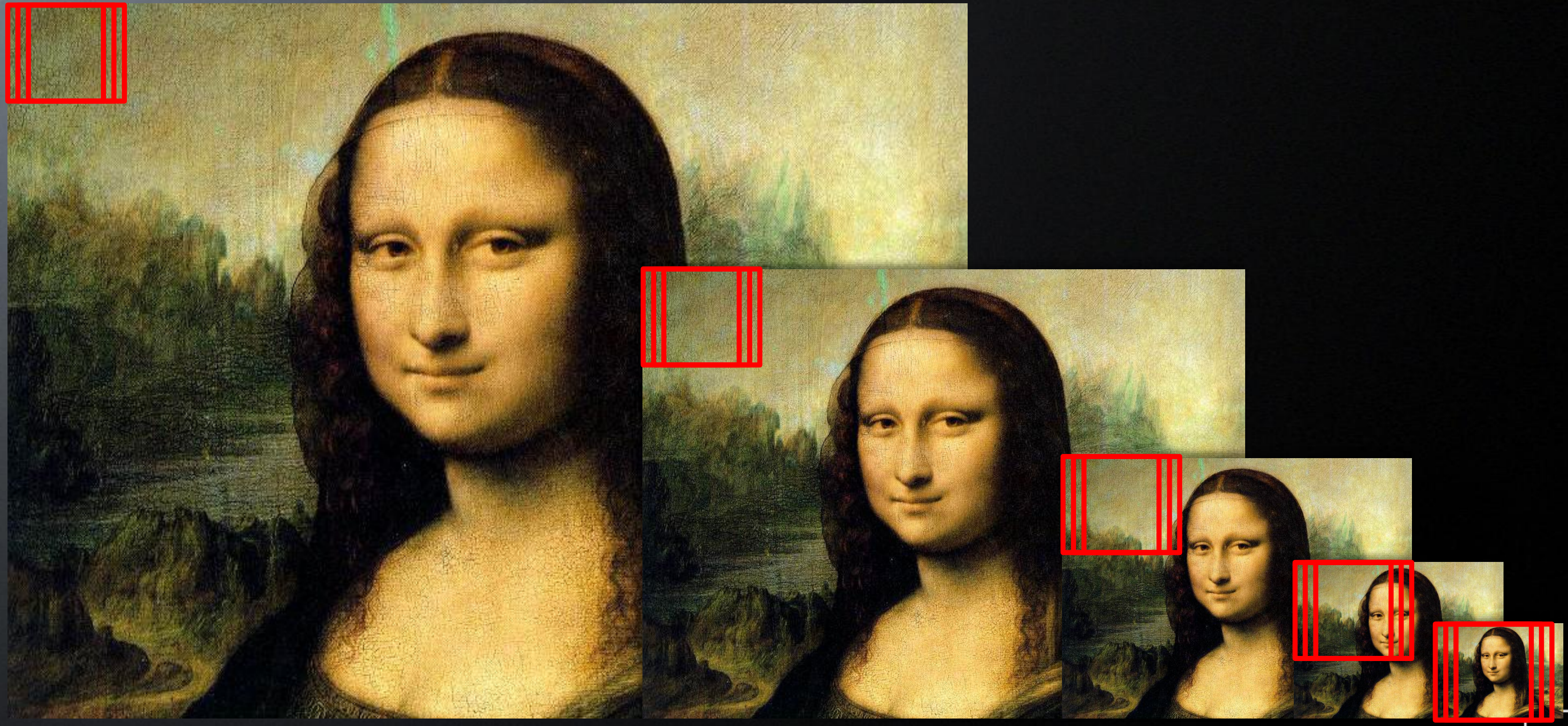
Quick HD Calculations

Search square = 21 x 21

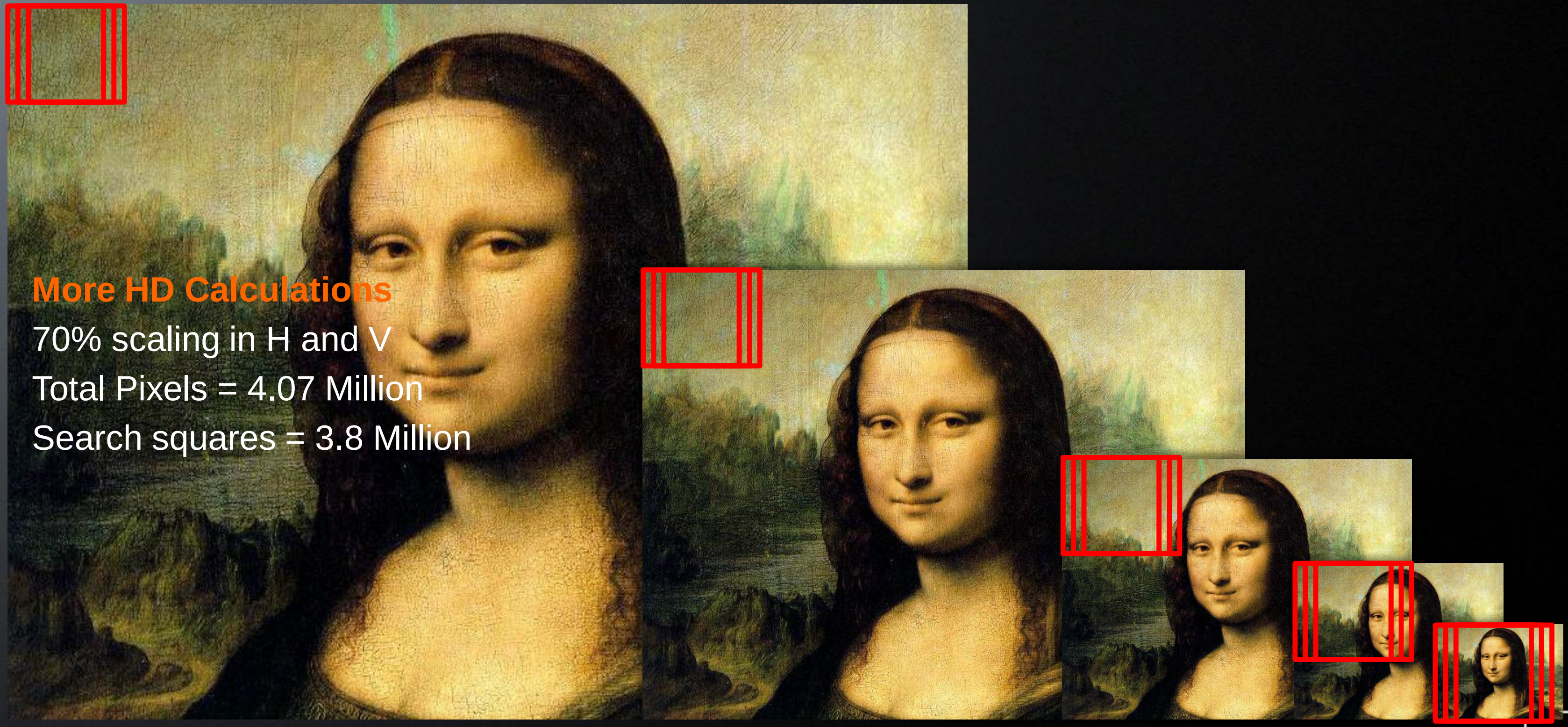
Pixels = 1920 x 1080 = 2,073,600

Search squares = 1900 x 1060 = ~2 Million

LOOKING FOR DIFFERENT SIZE FACES – BY SCALING THE VIDEO FRAME



LOOKING FOR DIFFERENT SIZE FACES – BY SCALING THE VIDEO FRAME



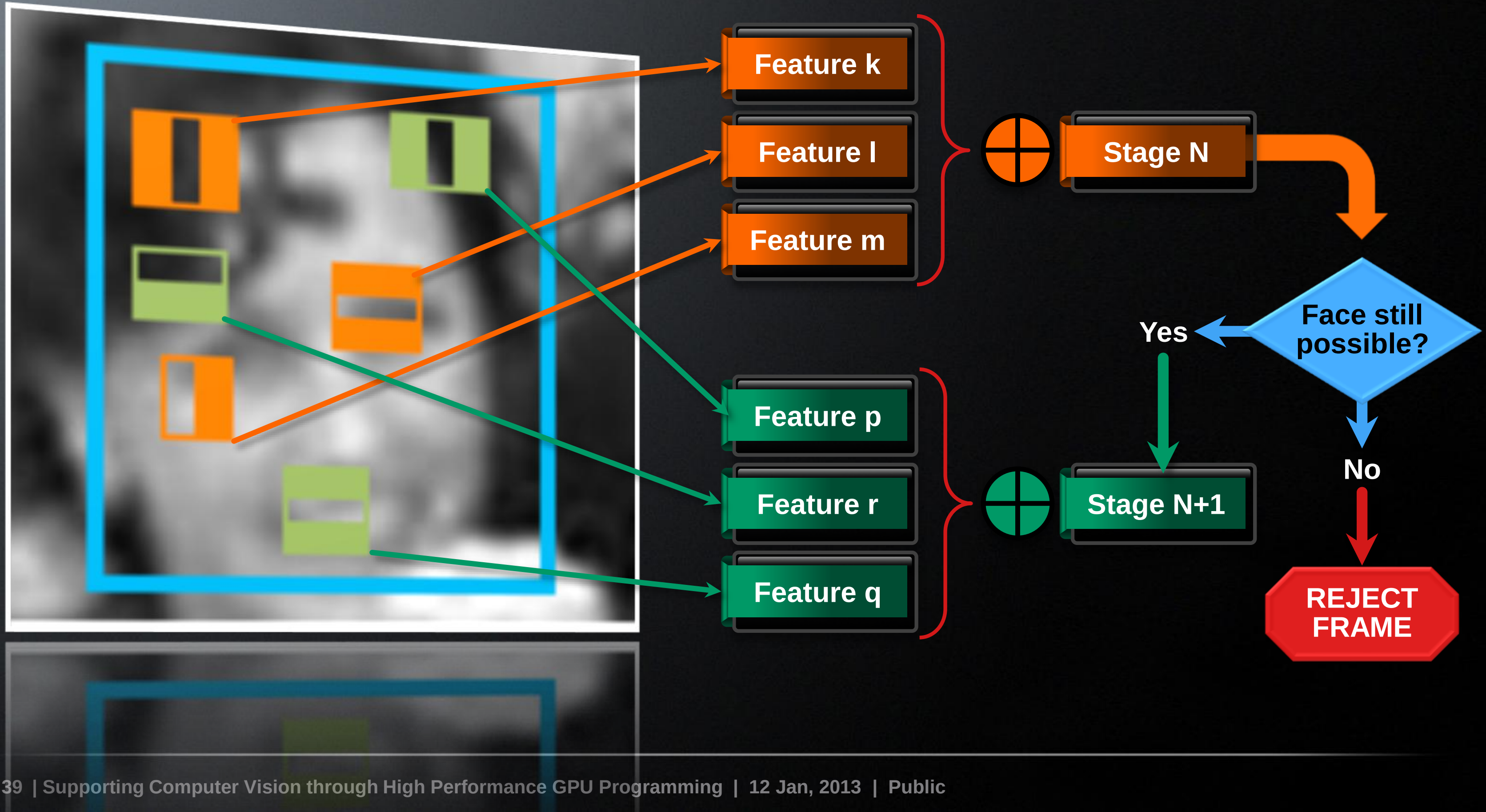
More HD Calculations

70% scaling in H and V

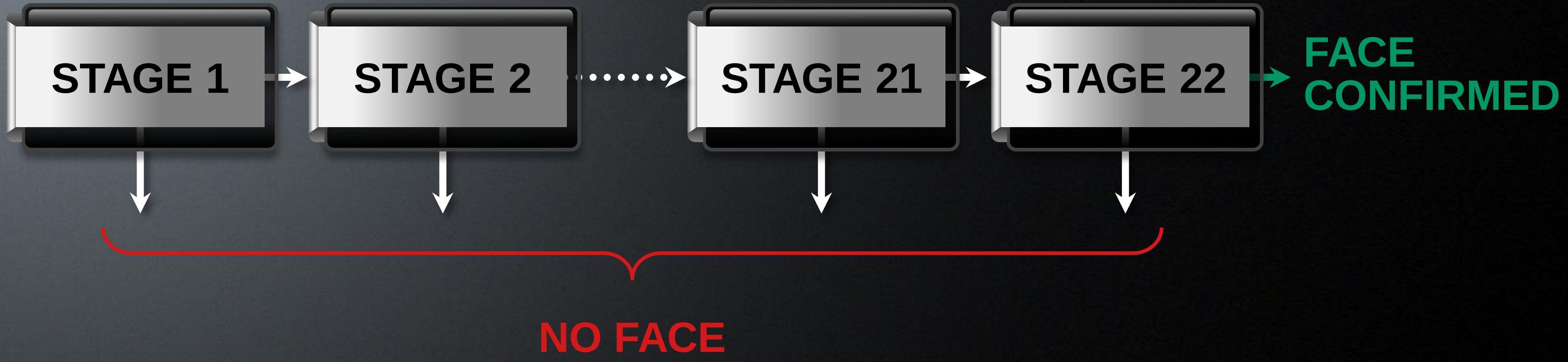
Total Pixels = 4.07 Million

Search squares = 3.8 Million

HAAR CASCADE STAGES



22 CASCADE STAGES, EARLY OUT BETWEEN EACH



Final HD Calculations

Search squares = 3.8 million

Average features per square = 124

Calculations per feature = 100

Calculations per frame = 47 GCalcs

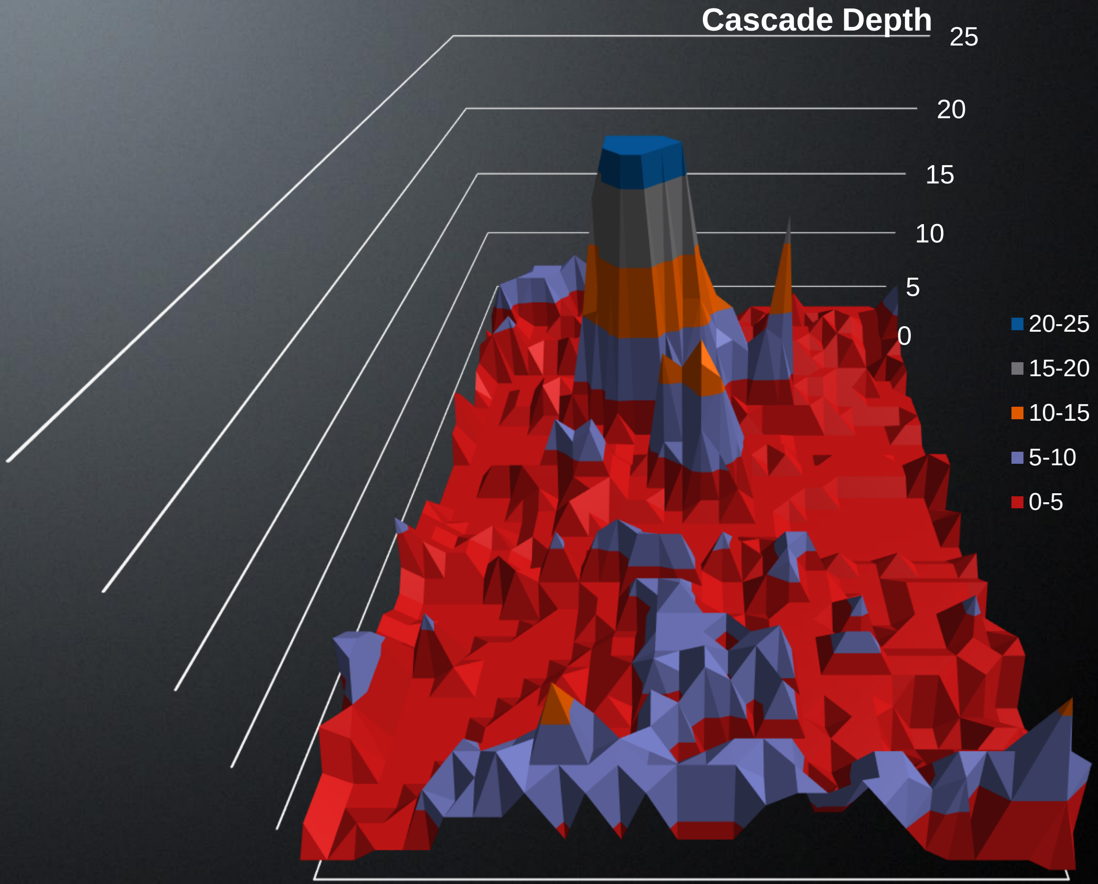
Calculation Rate

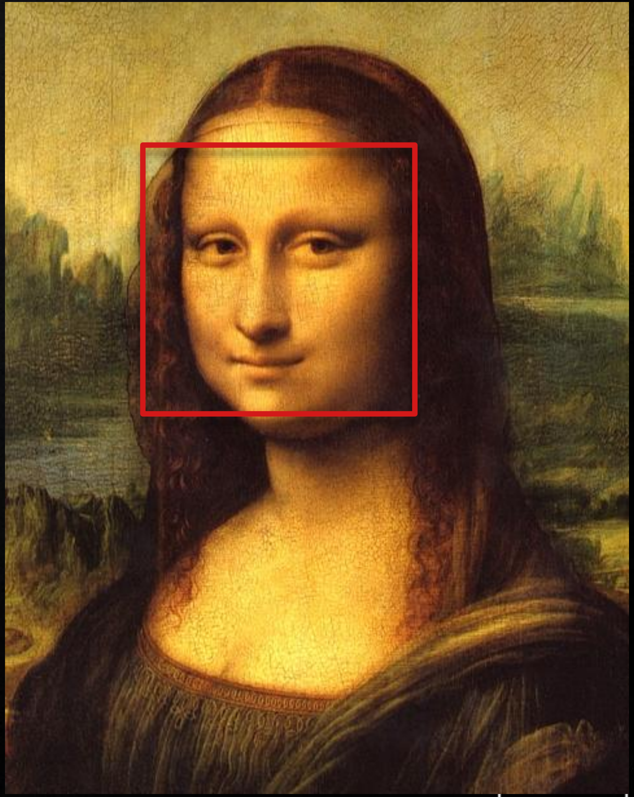
30 frames/sec = 1.4TCalcs/second

60 frames/sec = 2.8TCalcs/second

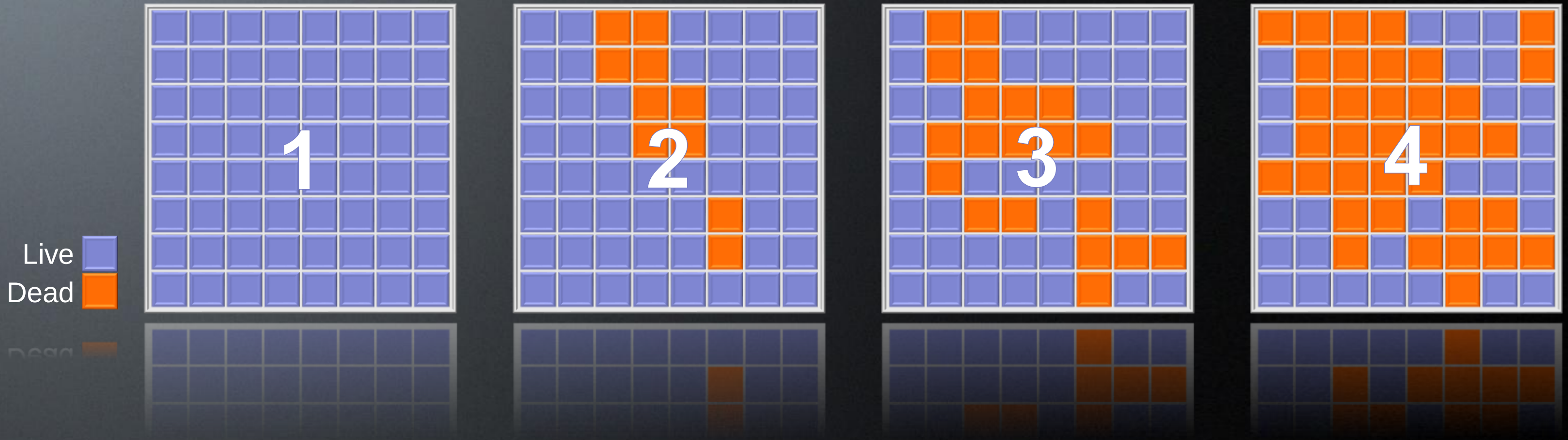
…and this only gets front-facing faces

CASCADE DEPTH ANALYSIS



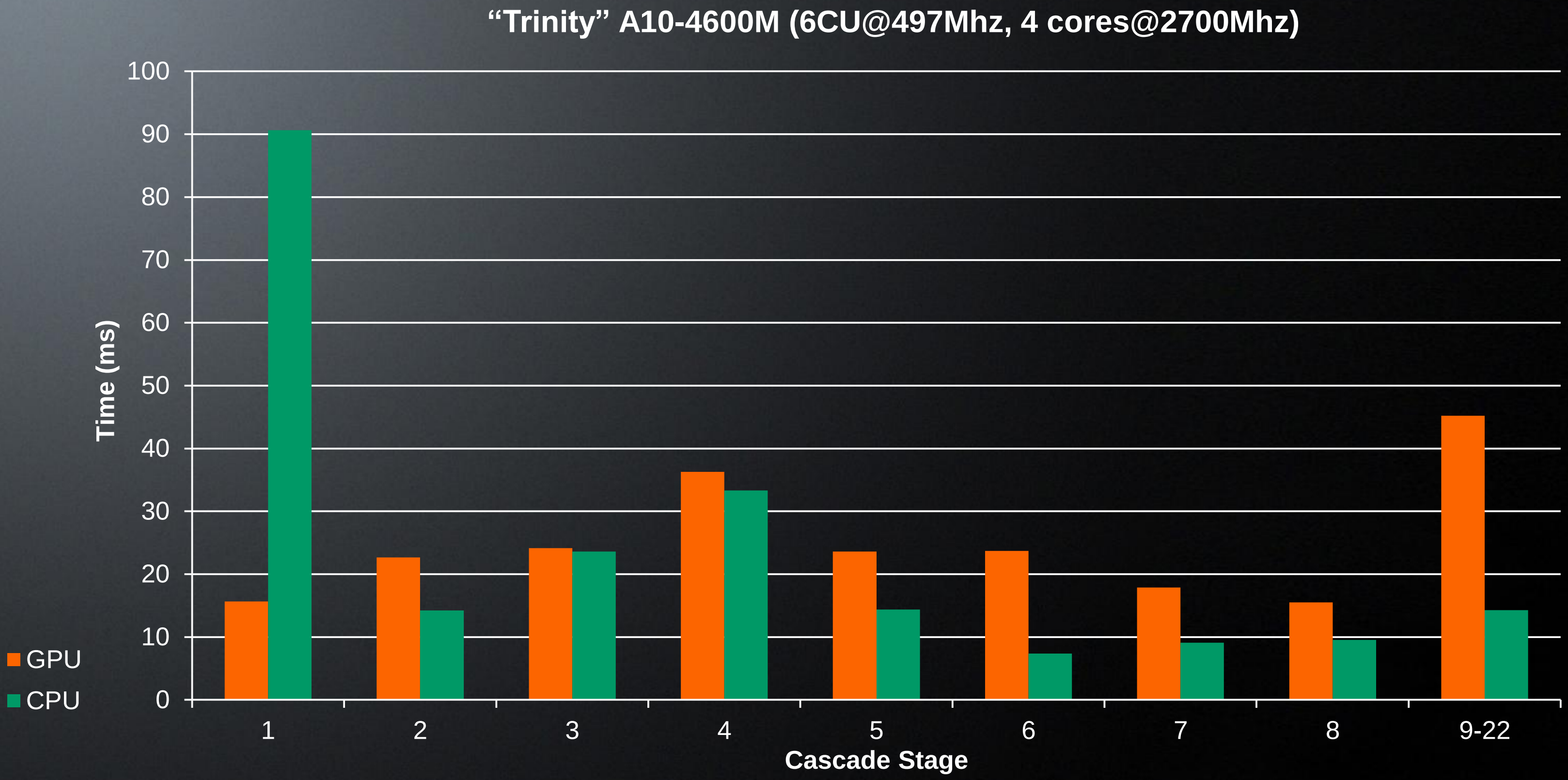


UNBALANCING DUE TO EXITS IN EARLIER CASCADE STAGES



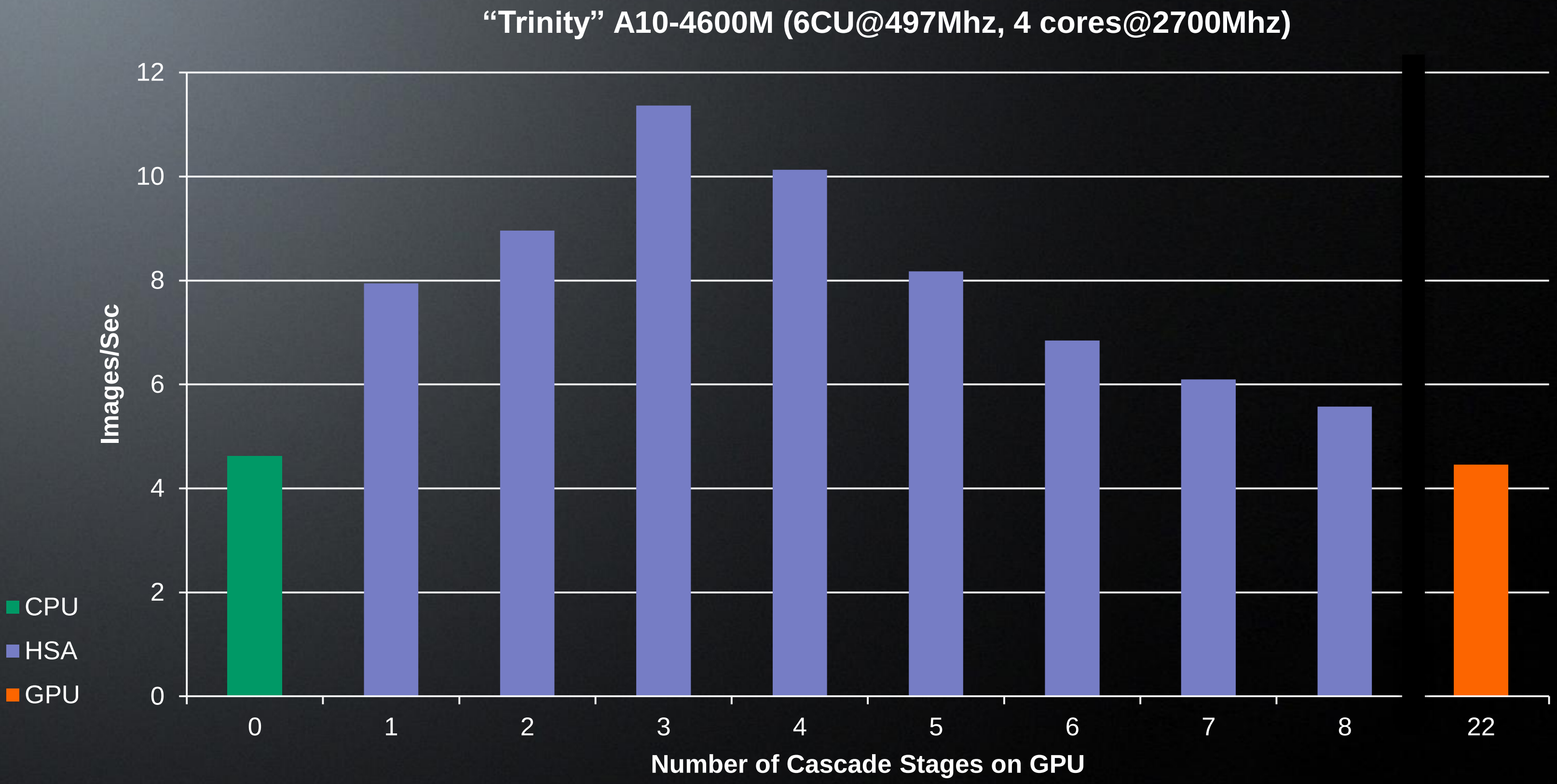
- When running on the GPU, we run each search rectangle on a separate work item
- Early out algorithms, like HAAR, exhibit divergence between work items
  - Some work items exit early
  - Their neighbors continue
  - SIMD packing suffers as a result

PROCESSING TIME/STAGE



AMD A10 4600M APU with Radeon™ HD Graphics; CPU: 4 cores @ 2.3 MHz (turbo 3.2 GHz); GPU: AMD Radeon HD 7660G, 6 compute units, 685MHz; 4GB RAM; Windows 7 (64-bit); OpenCL™ 1.1 (873.1)

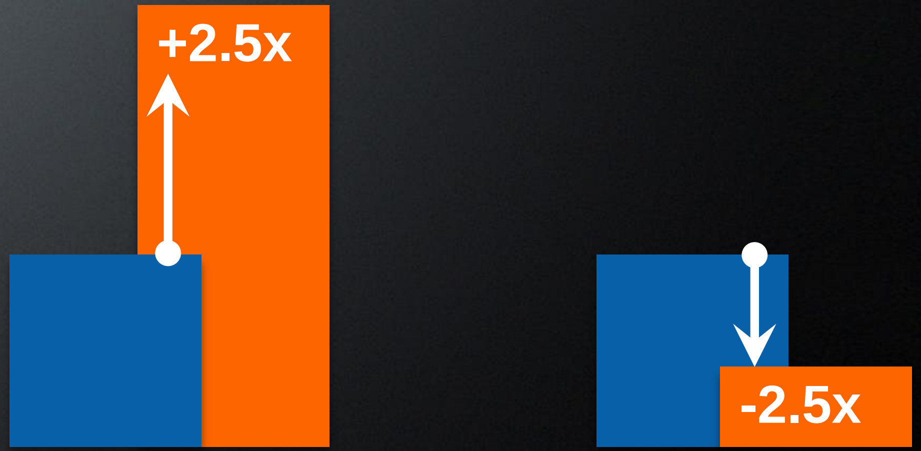
PERFORMANCE CPU-VS-GPU



AMD A10 4600M APU with Radeon™ HD Graphics; CPU: 4 cores @ 2.3 MHz (turbo 3.2 GHz); GPU: AMD Radeon HD 7660G,
6 compute units, 685MHz; 4GB RAM; Windows 7 (64-bit); OpenCL™ 1.1 (873.1)

HAAR SOLUTION – RUN DIFFERENT CASCADES ON GPU AND CPU

By seamlessly sharing data between CPU and GPU,
HSA allows the right processor to handle its appropriate workload



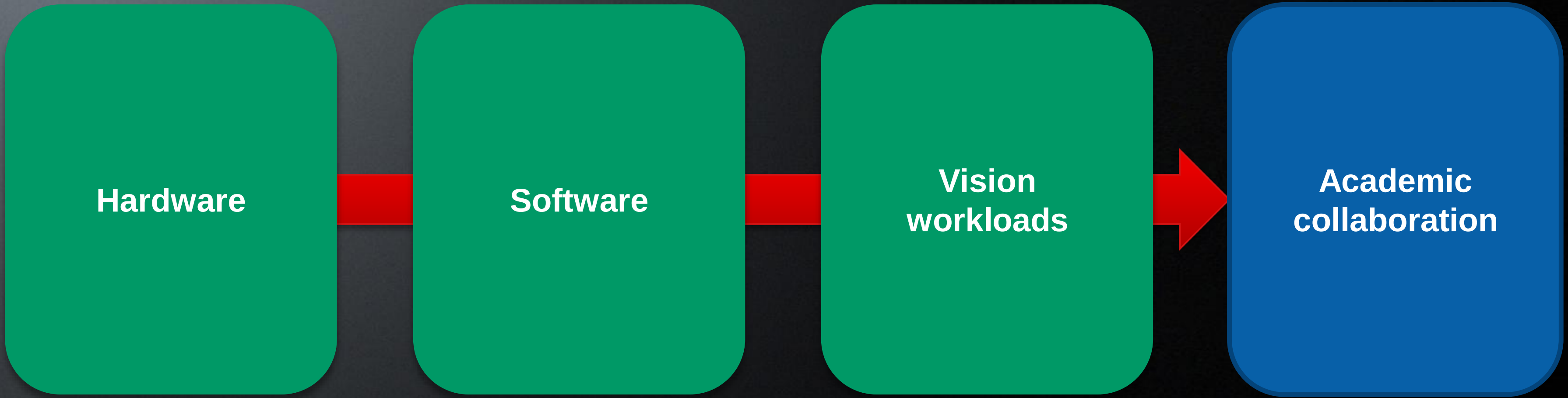
INCREASED PERFORMANCE

DECREASED ENERGY PER FRAME

OTHER LEADING ISVs

- Adobe
- Altair
- ANSYS
- ArcSoft
- Autodesk
- AVG
- BioWare A DIVISION OF EA
- codemasters
- COREL
- CyberLink
- DASSAULT SYSTEMES
- DIVX
- eidos MONTREAL
- fingertapps
- MAGIX
- MAIN CONCEPT now part of rovi
- Microsoft
- MotionDSP
- Muvizu
- nuvixa BE PRESENT
- ROXIO
- SEGA
- SONY make.believe
- Viewdle

WHERE AM I GOING WITH THIS?



University Collaboration Projects

SLAM: Simultaneous Localisation and Mapping

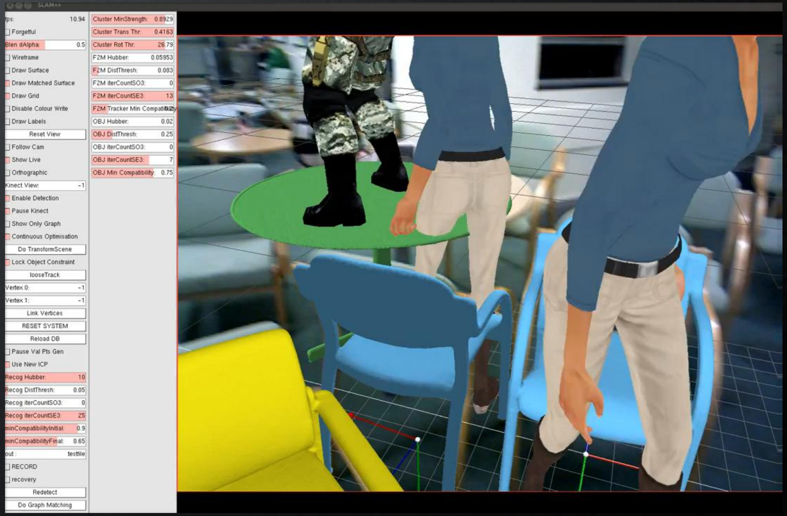
Collaborators:
A. Davison, P. Kelly, R. S. Moreno et al.
Imperial College London

SLAM

- SLAM: Simultaneous Localisation and Mapping
  - Given an new scene, camera tries to generate a map of the environment and track its own position from it
- Previous SLAM systems used low-level map elements like points or fiducial markers.
- We now use higher-level Objects
- Objects provides more meaningful scene understanding
  - e.g. With objects, a table detection can 'infer' the floor it lies on.
  - Natural occlusion handling
  - Physical predictions: Constrain objects to lie on the floor plane → improves detection quality
- Reduced complexity on map optimisation (#Objects <<< # Points)
- More realistic Augmented Reality applications

SLAM | OBJECT LEVEL

- We construct a database of scanned objects
- At runtime objects are detected on the GPU
  - Generating a map of objects locations/orientations
  - The map is immediately used to track the pose of the camera in real-time
- A batch optimisation routine executes on CPU to eliminate drift in camera motion



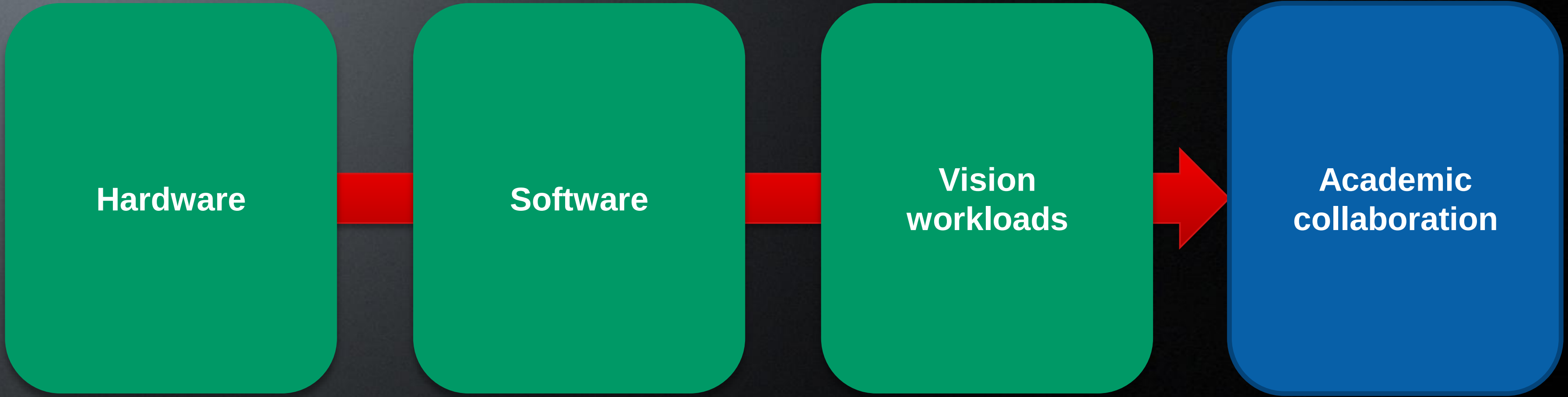
fps: 10.94
□ Forgetful
Blen dAlpha: 0.5
□ Wireframe
□ Draw Surface
☑ Draw Matched Surface
☑ Draw Grid
□ Disable Colour Write
□ Draw Labels
Reset View
□ Follow Cam
☑ Show Live
□ Orthographic
Kinect View: -1
☑ Enable Detection
☑ Pause Kinect
□ Show Only Graph
☑ Continuous Optimisation
Do TransformScene
☑ Lock Object Constraint
looseTrack
Vertex 0: -1
Vertex 1: -1
Link Vertices
RESET SYSTEM
Reload DB
□ Pause Val Pts Gen
☑ Use New ICP
Recog Hubber: 10
Recog DistThresh: 0.05
Recog iterCountSO3: 0
Recog iterCountSE3: 25
minCompatibilityInitial: 0.9
minCompatibilityFinal: 0.65
out : testfile
□ RECORD
□ recovery
Redetect
Do Graph Matching

Cluster MinStrength: 0.8929
Cluster Trans Thr: 0.4163
Cluster Rot Thr: 26.79
F2M Hubber: 0.05953
F2M DistThresh: 0.083
F2M iterCountSO3: 0
F2M iterCountSE3: 13
F2M Tracker Min Compatibility: 0.2
OBJ Hubber: 0.02
OBJ DistThresh: 0.25
OBJ iterCountSO3: 0
OBJ iterCountSE3: 7
OBJ Min Compatibility: 0.75

Real-time Depth Maps from Video

- A pose of a monocular camera is first tracked with PTAM on the CPU
  - [Klein et al. ISMAR 08]
- Given a set of frames with known camera poses we build a depth map on the GPU
  - Camera must move to provide disparity information to figure depth
  - Generated depth maps used as a first step towards a full 3D object reconstruction pipeline
- Uses photo-consistency across frames
  - Assume lighting remains constant with small camera motion
- Look for a similar looking pixel of Frame A in Frames B, C, D, E ...
- True depth of pixel should generated similar intensities over frames
- Regularise solution to return smooth depths over large pixel areas
  - NewCombe et al. ICCV 2011

WHERE AM I GOING WITH THIS?

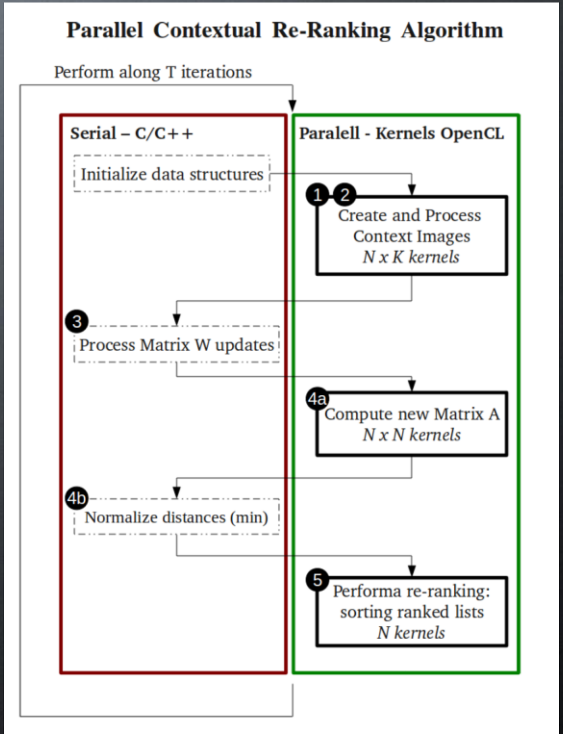


University Collaboration Projects

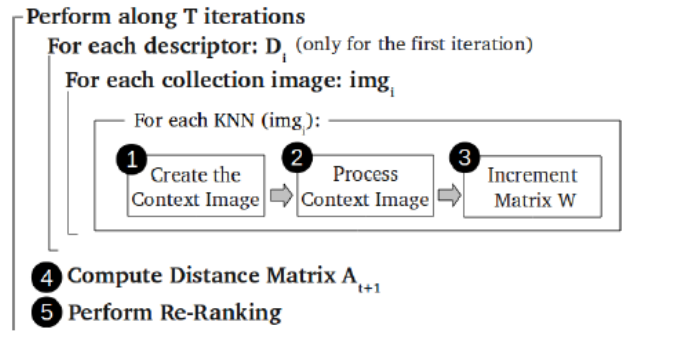
Content-based image retrieval on APU clouds

Collaborators:
D. Pedronette, E. Borin, R. Torres UNICAMP, Brazil
M. Breternitz, AMD Research

CBIR | Image Re-ranking and Rank Aggregation



Parallel Contextual Re-Ranking Algorithm
Perform along T iterations
Serial – C/C++
Paralell - Kernels OpenCL
Initialize data structures
1
2
Create and Process Context Images
N x K kernels
3
Process Matrix W updates
4a
Compute new Matrix A
N x N kernels
4b
Normalize distances (min)
5
Performa re-ranking: sorting ranked lists
N kernels



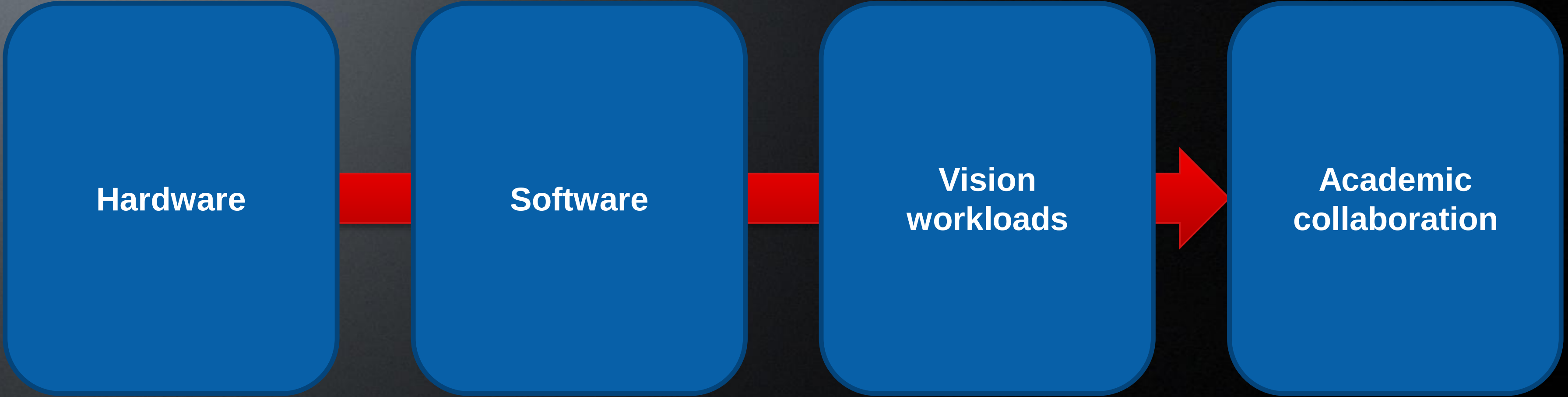
Perform along T iterations
For each descriptor: D_i (only for the first iteration)
For each collection image: img_i
For each KNN (img_i):
1
Create the Context Image
2
Process Context Image
3
Increment Matrix W
4
Compute Distance Matrix A_{t+1}
5
Perform Re-Ranking

Contextual rank aggregation algorithm

"Efficient Image Re-ranking and Rank Aggregation Computation on GPUs", 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, July, 2012

AMD Opteron 6168 1.9GHz, 12 Cores; ATI FirePro V7800; Ubuntu 10.04; AMD-APP-SDK-v2.4

SUMMARY



Hardware

Software

Vision workloads

Academic collaboration

Q&A

Founders

AMD ARM Imagination MEDIATEK QUALCOMM®

SAMSUNG TEXAS INSTRUMENTS

Promoters

LG Electronics

Supporters

Arteris MULTICORE WARE

Contributors

apical <symbio> SONICS DMP CEVA

Associates

VIVANTE

Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, the HSA logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. OpenCL™ is a trademark of Apple Corp. which is licensed to the Khronos Organization. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.