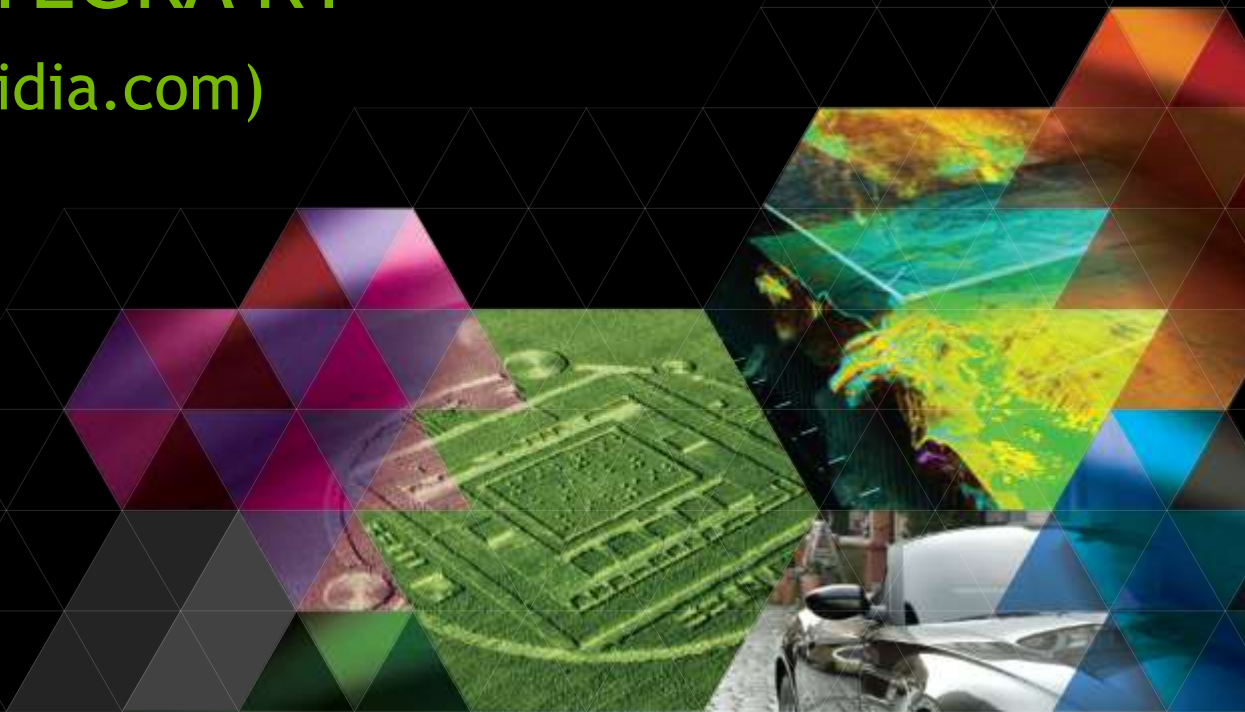
GPU TECHNOLOGY CONFERENCE

NVIDIA

COMPUTE WITH TEGRA K1

Amit Rao (amitr@nvidia.com)



AGENDA

- Introducing Tegra K1
- Tegra K1 Compute Software Capabilities
- Closer look at CUDA 6.0 on Tegra
- Compute software development on Tegra K1
- Compute in action on Tegra K1

TEGRA K1 FOR COMPUTE

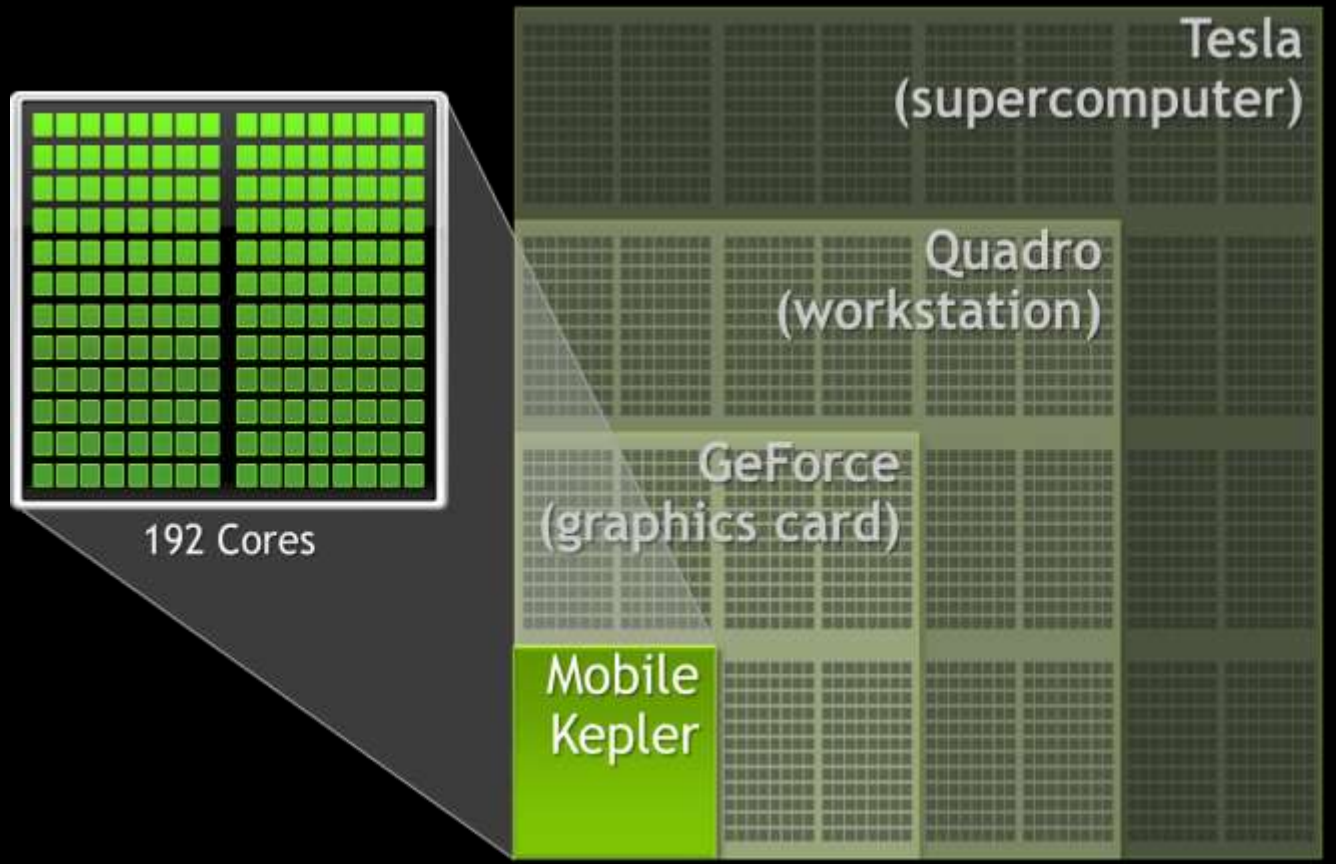
Kepler Architecture
192 CUDA Cores

Cortex-A15
4-Plus-1

Shared Physical Memory

2D Engine / ISP

KEPLER FOR COMPUTE



Kepler Architecture
192 CUDA Cores

ISA Compatible to GeForce, Quadro, Tesla

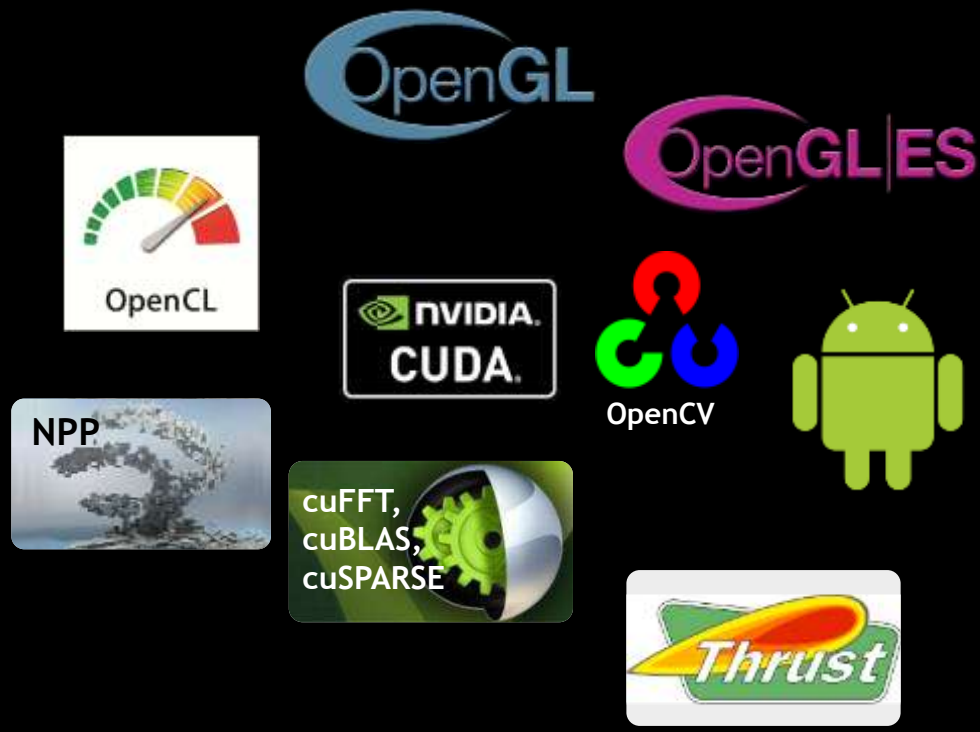
SM 3.2

64kb L1 Cache and Shared Memory

128kb L2 Cache

128 kb Register File

SOFTWARE FOR COMPUTE



Tegra K1 can accelerate

Renderscript

OpenGL /OpenGL ES with compute shaders

OpenCL full profile

CUDA
and a whole list of libraries enabling compute on the GPU

RENDERSCRIPT

C99 based kernel language

Easy programmability with host and device portability. Portable across wide range of devices, fastest on Tegra K1

GPU (+ CPU) and more

Renderscript API 19 Support

TEGRA K1 FOR RENDERSCRIPT

Kepler Architecture
192 CUDA Cores

Cortex-A15
4-Plus-1

Shared Physical Memory

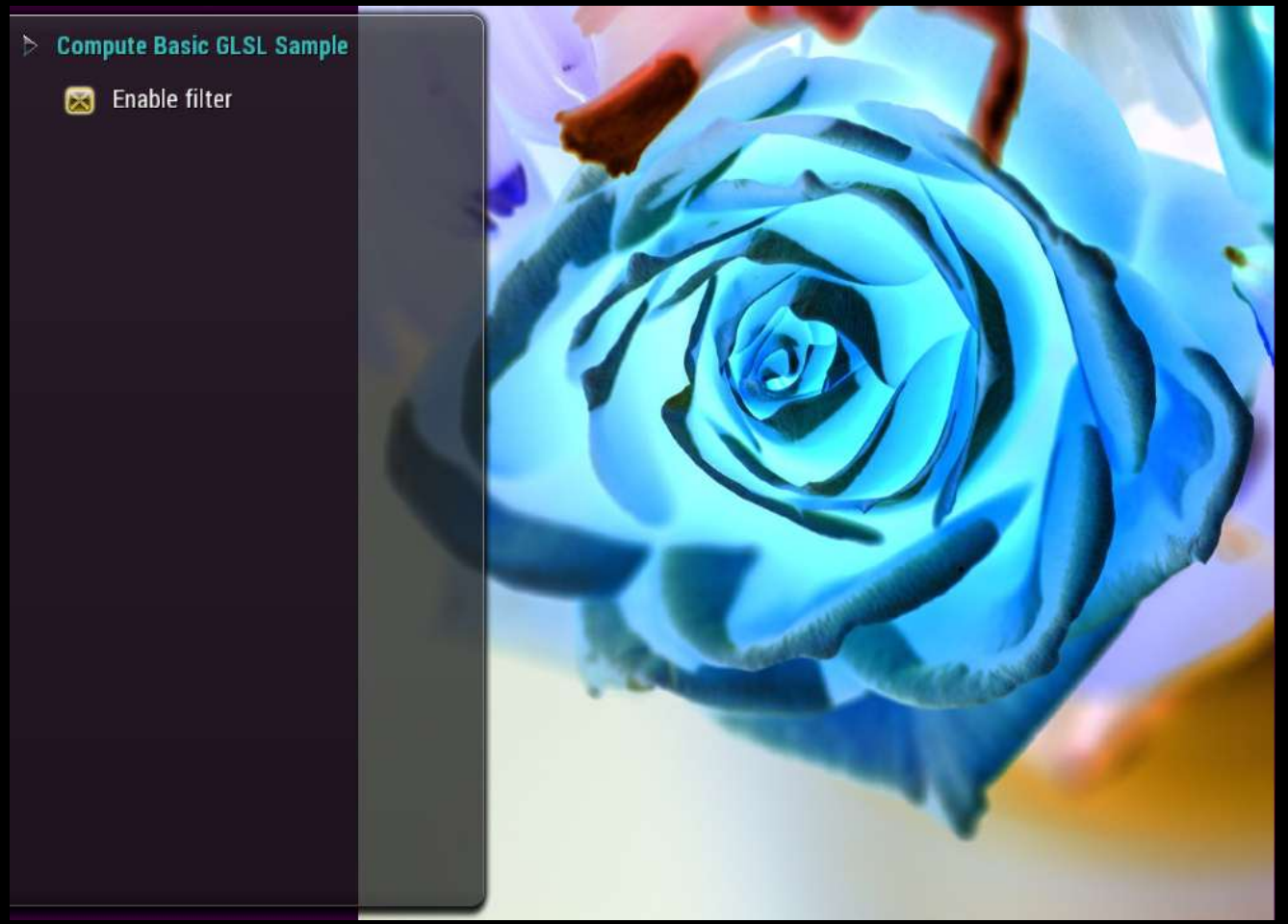
2D Engine / ISP

RENDERSCRIPT ON THE SOC

- Acceleration of Renderscript Scripts over GPU
  - ScriptC, not just ScriptIntrinsics
  - Huge gains in performance and performance/watt
- Runtime capable of scheduling work across units
  - CPU
  - GPU
  - 2D Engine/ISP
- Related Session:

S4885 - Efficient Parallel Computation on Android
Jason Sams, Tim Murray
Wednesday, 10:30 A.M.

OPENGL



Standard OpenGL API

GLSL language

Easiest integration for applications already using OpenGL or OpenGL ES API

OpenGL 4.4 and OpenGL ES 3.1 with compute shaders on Tegra K1

COMPUTE SHADERS

- Execute algorithmically general purpose GLSL shaders
  - Operate on buffers, images and textures
- Process graphics data in the context of the graphics pipeline
  - Easier than interoperating with a compute API for graphics apps
- Standard part of all OpenGL 4.3+ implementations
  - And now OpenGL ES 3.1!


Image processing

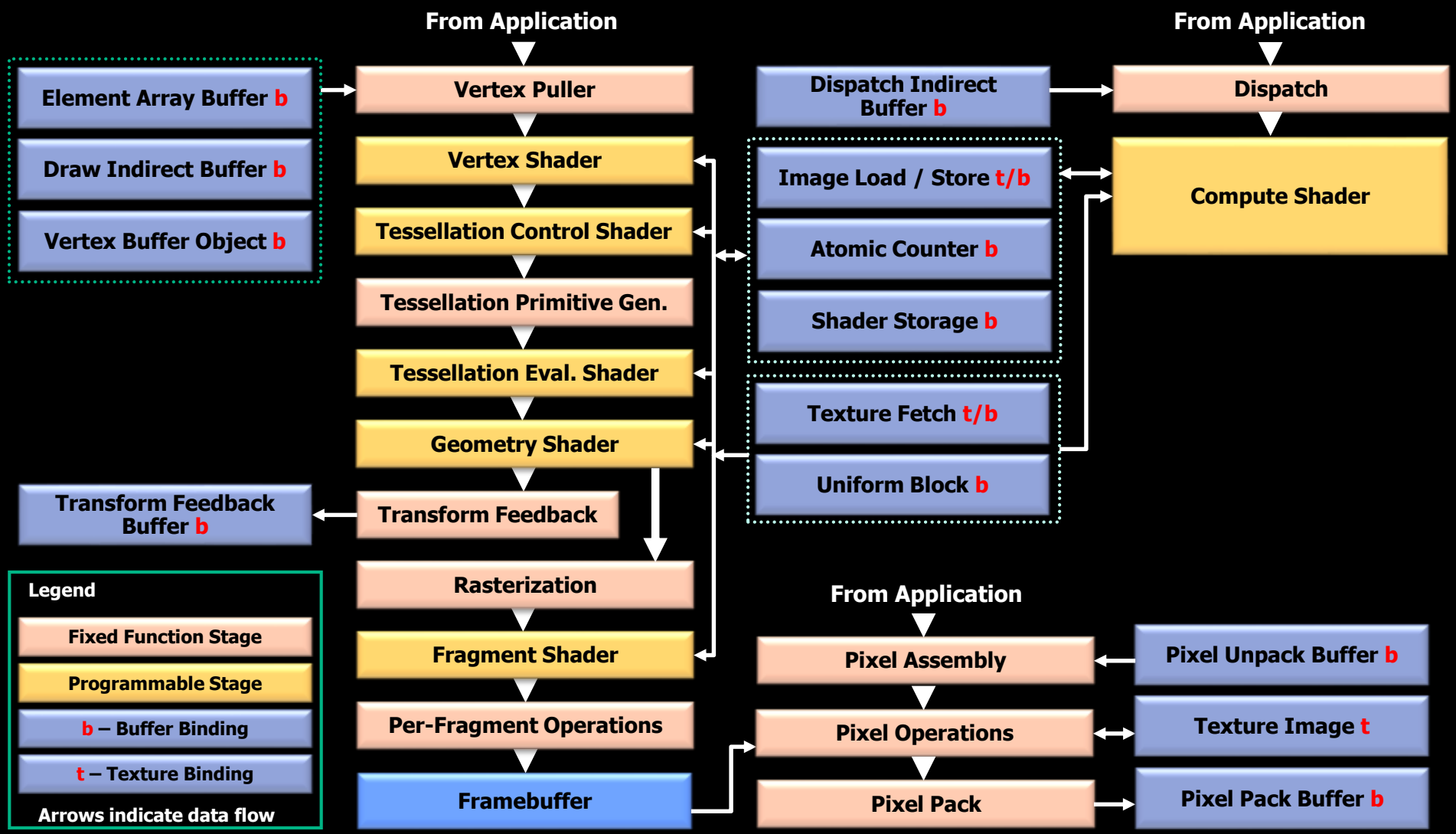

AI Simulation


Ray Tracing

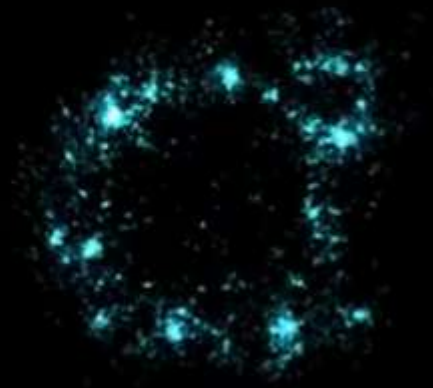

Wave Simulation


Global Illumination

OPENGL COMPUTE SHADERS



OPENCL



NVIDIA December 2008 – First OpenCL demo on a GPU

OpenCL 1.2 full profile capable

C99 based OpenCL C kernel language

Device portability with ability to tune for specific device types

OpenCL 1.1 full profile for registered developers

TEGRA K1 FOR OPENCL

- Full profile support
  - True portability from desktop
  - Higher precision, higher limits
- Awesome performance
- Related Session:

S4808 - Real-Time Facial Motion Capture and Animation on Mobile
Emiliano Gambaretto
Tuesday, 5:00 P.M.

CUDA

Language integrated
C/C++

Easy programmability
and low level access
to Kepler architecture

CUDA 6.0, Unified
Memory, Unified
Virtual Addressing,
NPP, cuFFT, cuBLAS,
cuda-gdb, memcheck,
nvprof

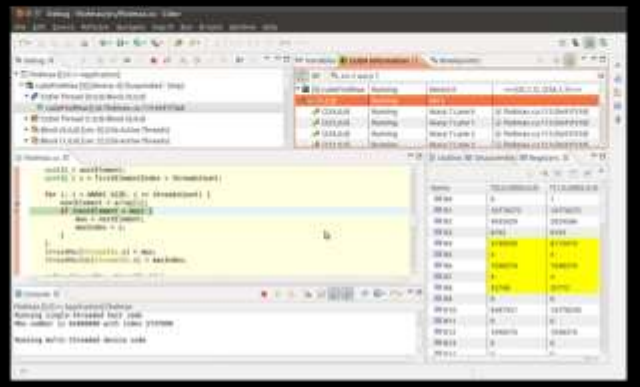
NVIDIA® NSIGHT™

eclipse

Full featured IDE for developing CUDA-C apps on X86, ARM & POWER8.

CUDA-Aware Editor

- Automated CPU to GPU code refactoring
- Semantic highlighting of CUDA code
- Integrated code samples & docs
- Cross-compilation for Linux target



Nsight Debugger

- Simultaneously debug CPU and GPU code
- Inspect variables across CUDA threads
- Use breakpoints & single-step debugging
- Integrated CUDA memory checker

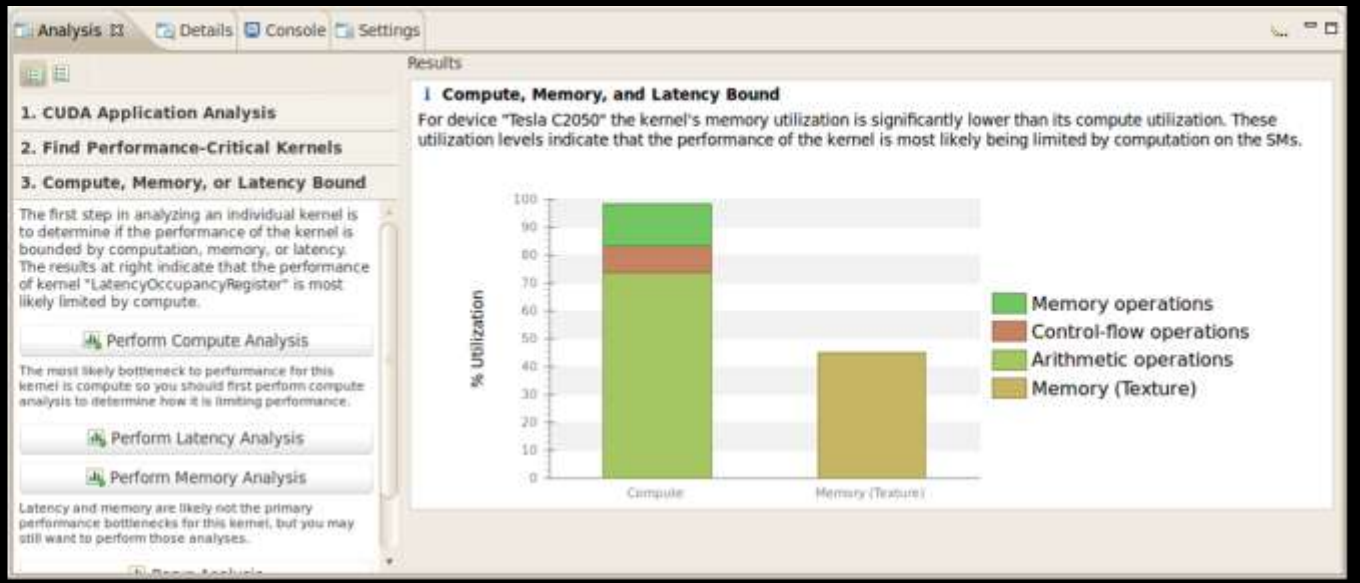
Nsight Profiler

- Quickly identifies performance issues
- Guided expert analysis
- Source line correlation

CUDA STANDALONE TOOLS

Visual Profiler

- Trace CUDA activities
- Kernel Profiler
- Performance instrumentation with source code correlation
- Guided Expert Analysis



NVPROF

- Generates execution summary
- Gather Performance events and metrics

CUDA-MEMCHECK

- Out of bounds memory access detection
- Detects Race Condition

CUDA-GDB

- Command line CUDA debugging
- Debug CPU and GPU code

* Android new in CUDA 6.0

TEGRA K1 FOR COMPUTE

Kepler Architecture
192 CUDA Cores

Cortex-A15
4-Plus-1

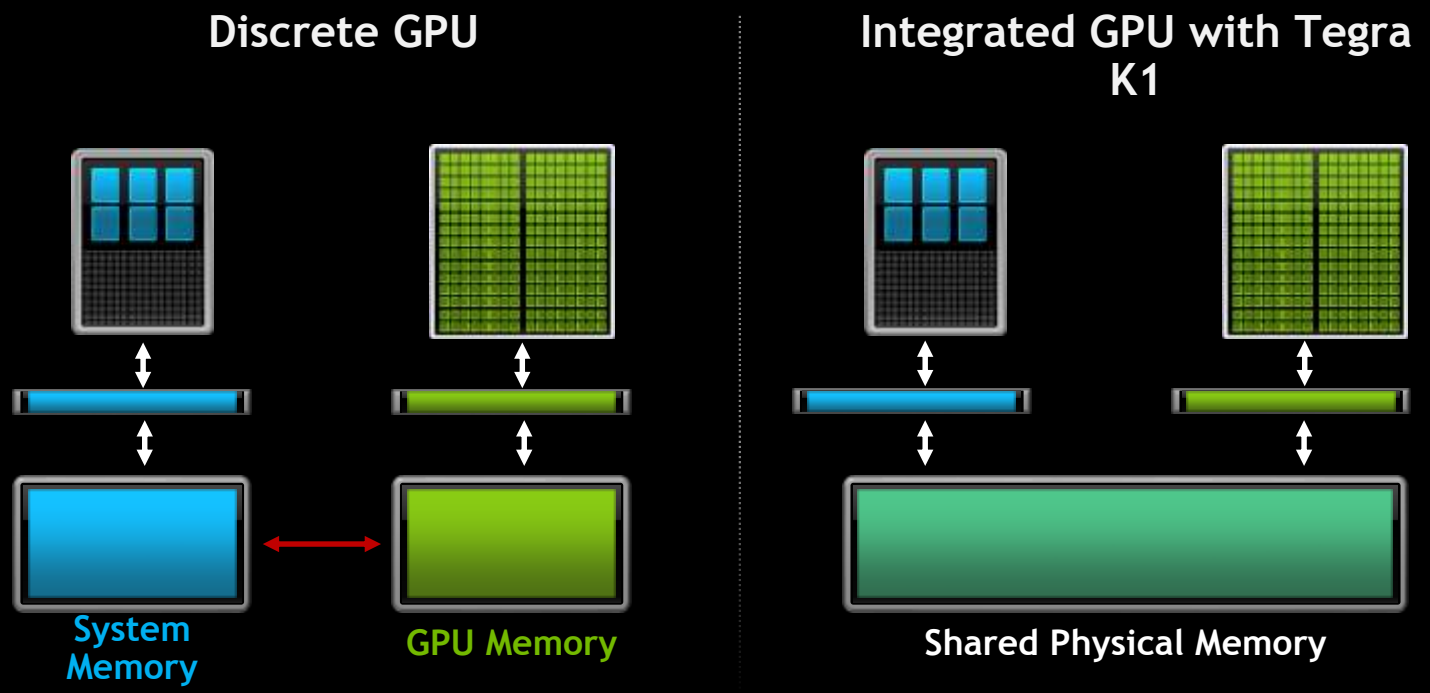
Shared Physical Memory

2D Engine / ISP

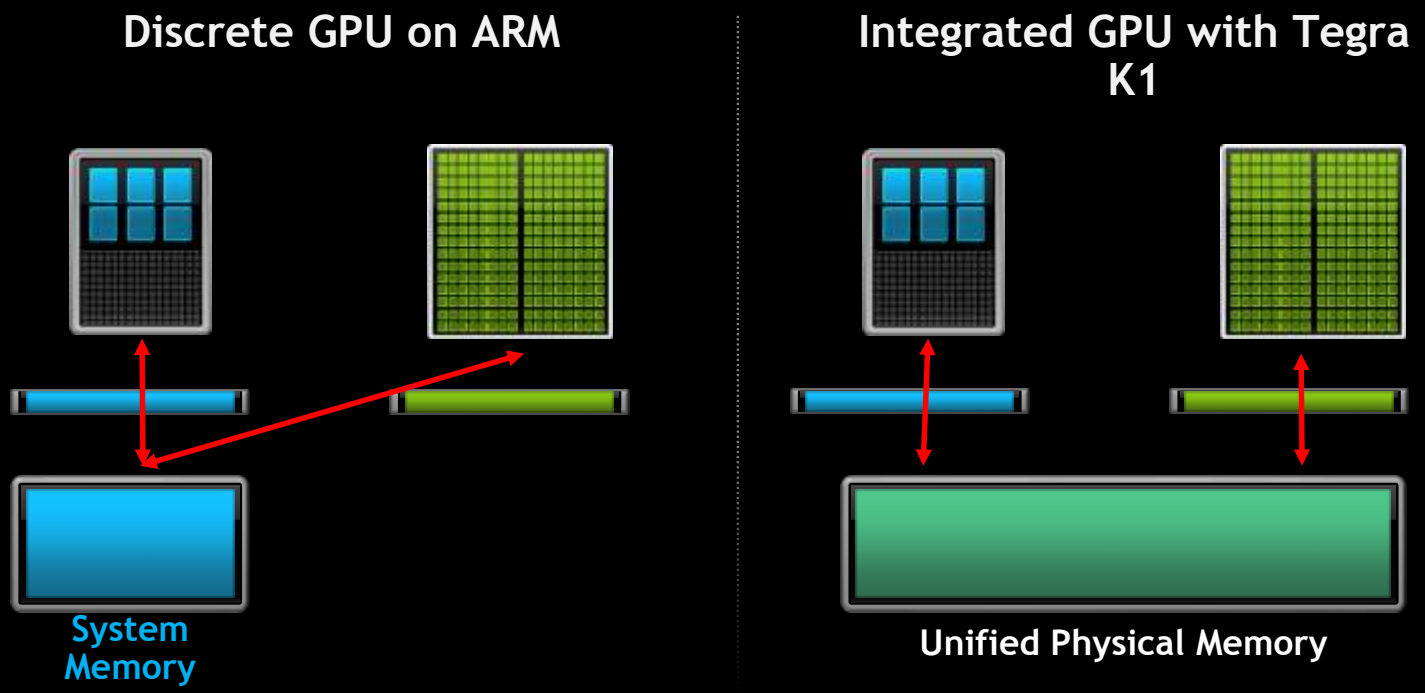
SHARING DATA BETWEEN HOST AND DEVICE

- Pageable + Memory Copy
- Page-locked/Pinned + Memory Copy
- Zero Copy

UNIFIED PHYSICAL MEMORY ON TEGRA K1



ZERO COPY



Bypassing many cache benefits

CUDA 6.0 FOR TEGRA – CLOSER LOOK

- Same programming model and features from superphone to supercomputer
- There are some differences
  - 32-bit support for CUDA Unified Virtual Addressing
  - 32-bit support for CUDA Unified Memory
  - Zero Copy support (no cudaHostRegister)

SHARING DATA BETWEEN HOST AND DEVICE

- Pageable + Memory Copy
- Page-locked/Pinned + Memory Copy
- Zero Copy
- Managed Memory

MANAGED MEMORY ON CUDA 6.0

Traditional

```
#define N (2048*2048)
#define TPBLOCK 512
int main(void) {
    // host copies of a, b, c
    int *a, *b, *c;
    // device copies of a, b, c
    int *d_a, *d_b, *d_c;
    int size = N * sizeof(int);
    int result = 0;

    // Alloc space for device copies of a, b, c
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);

    // Alloc space for host copies of a, b, c and
setup input values
    a = (int *)malloc(size); random_ints(a, N);
    b = (int *)malloc(size); random_ints(b, N);
    c = (int *)malloc(size);
```

Unified Memory

```
#define N (2048*2048)
#define TPBLOCK 512
int main(void) {
    // unified copies of a, b, c
    int *a, *b, *c;

    int size = N * sizeof(int);
    int result = 0;

    // Alloc space for unified copies of a, b, c
    cudaMallocManaged((void **)&a, size);
    cudaMallocManaged((void **)&b, size);
    cudaMallocManaged((void **)&c, size);

    // Setup input values, avoid duplicate data alloc
    random_ints(a, N);
    random_ints(b, N);
```

MANAGED MEMORY ON CUDA 6.0

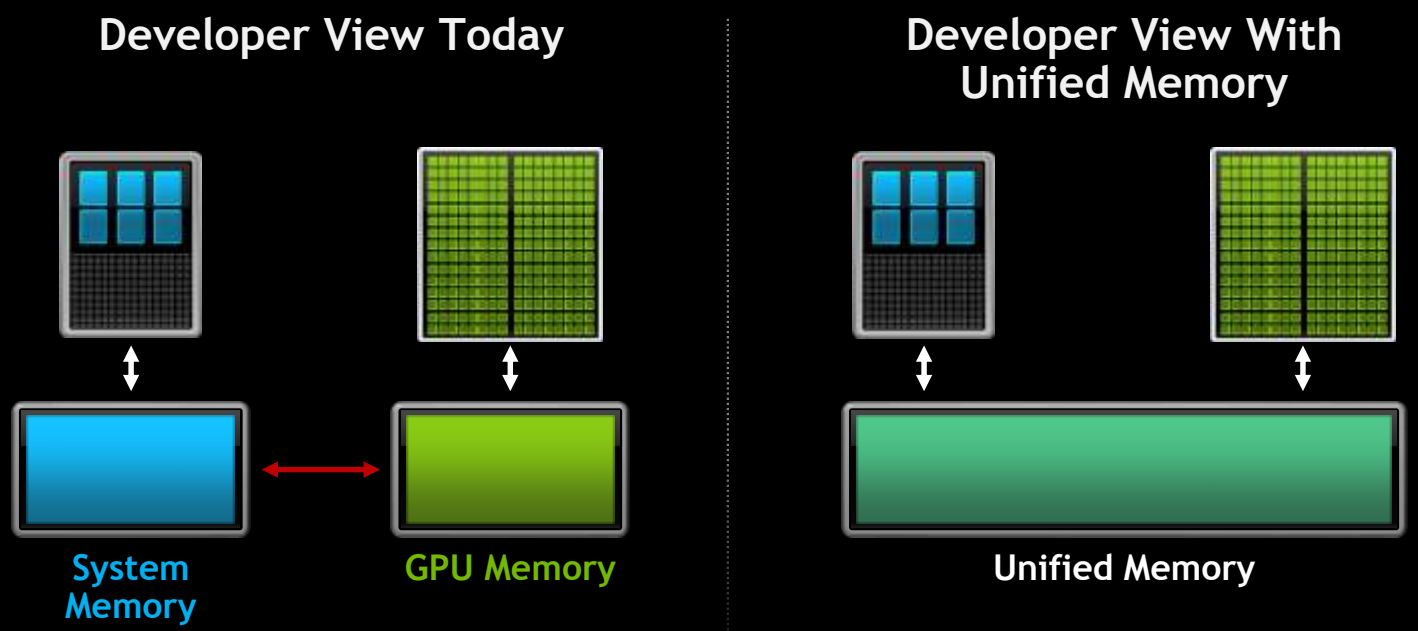
Traditional

```
// Copy inputs to device
cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);

// Launch add() kernel on GPU
add<<<N/TPBLOCK,TPBLOCK>>>(d_a, d_b, d_c);

// Copy result back to host
cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);

// Use c, do cleanup
result = verify_add(c, a, b);
free(a); free(b); free(c);
cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
return result;
}
```

Unified Memory

```
// Implicit mapping of inputs to device
// Results in CPU cache flush for managed memory
// Results in GPU cache invalidate

// Launch add() kernel on GPU
add<<<N/TPBLOCK,TPBLOCK>>>(a, b, c);

// Implicit mapping of result back to host
// Results in GPU cache flush
// Results in CPU cache invalidate for managed memory

cudaDeviceSynchronize();

// Use c, do cleanup
result = verify_add(c, a, b);
cudaFree(a); cudaFree(b); cudaFree(c);
return result;
}
```

CUDA UNIFIED MEMORY



Dramatically Lower Developer Effort
Faster performance on SoC

UNIFIED MEMORY IN CUDA 6.0

- Simplifies many common usage patterns
- Asynchronous binding to device/stream/host
- Related Session:

S4801 - Using Unified Memory in CUDA 6
Mark Ebersole
Wednesday, 9:00 A.M.

GPU TECHNOLOGY CONFERENCE

NVIDIA

GETTING UP AND RUNNING TADP FOR GPU COMPUTE

Navjot Garg (ngarg@nvidia.com)

CONTENTS

- Introduction to TADP compute
- Installation steps
- Getting Started with CUDA Samples
- Setting up CUDA Makefile
  - Sample Walkthrough
- Setting up Android Makefile
  - Sample Walkthrough

INTRODUCTION TO TADP COMPUTE

- One stop shop for all components
- Quick and easy cross-compilation and linking
- Profiling CUDA applications running on TK1
- Easy update to new versions of CUDA Toolkit, keeps previous work intact
- Includes samples to get started with CUDA development on TegraK1
- Limited distribution. Register at the URL below for access and specify the "**Reason for application**" as "**Access to TADP Compute**"

- http://developer.nvidia.com/register-tegra-registered-developer-program

INSTALLATION STEPS

STEP 1	Run the TADP compute installer and follow the instructions on installation wizard.
STEP 2	source ~/.bashrc
STEP 3	In the same terminal, change to <installation location>/NVPACK/eclipse and launch eclipse (this makes the environment variables, set during installation, visible to eclipse)
STEP 4	Specify the location of Android NDK in eclipse. Go to Windows->preference... Expand "Android" option in left pane and select NDK. Set the NDK location to the “<installation location>/NVPACK/android-ndk-r9b” in the right pane.

TADP COMPUTE

The latest version TADP-2.0c-R7 installs following:

- **CUDA-6.0 (ARM compatible)**
- **Android-NDK-r9b**
- **Android OS image (build includes CUDA driver)**
- Android-SDK
- Apache-ant-1.8.2
- Eclipse(with ADT plugins installed)
- TDK_Samples
- Tegra_Graphics_Debugger
- Tegra_Profiler_2.0
- CUDA libraries, headers and tools
- CUDA Samples for TK1

GETTING STARTED WITH CUDA SAMPLES

STEP 1	Open eclipse
STEP 2	Choose <TADP install dir>/nvsample_workspace as workspace. The samples are already imported to this workspace
STEP 3	Make a static library from the cuda code using the makefile in CUDA folder, which is linked against the android project in Android.mk
STEP 4	Build and run the android project as Android Application.

SETTING UP CUDA MAKEFILE

- Make a compatible ARM library for android, cross-compile CUDA project as follows:

```
GCC=$(NDK_ROOT)/toolchains/arm-linux-androideabi-4.6/gen_standalone/linux-x86_64/bin/arm-linux-androideabi-g++

NVCC=$(CUDA_TOOLKIT_ROOT)/bin/nvcc -ccbin $(GCC) -target-cpu-arch=ARM -m32 -arch=sm_30 -O3 -Xptxas '-dlcm=ca' -target-os-variant=Android

lib_boxfilter.a: $(OBJS)
	$(NVCC) -lib -o "$@" $(OBJS)
```

SETTING UP ANDROID MAKEFILE

STEP 1

For building the apk in android project, add following to Android.mk under jni folder:

```
include $(CLEAR_VARS)
LOCAL_MODULE := lib_boxfilter
LOCAL_SRC_FILES := ../cuda/lib_boxfilter.a
include $(PREBUILT_STATIC_LIBRARY)
```

SETTING UP ANDROID MAKEFILE

STEP 2

Configuration to use CUDA run-time:

```
include $(CLEAR_VARS)
LOCAL_MODULE := libcudart_static
LOCAL_SRC_FILES := $(CUDA_TOOLKIT_ROOT)/lib/libcudart_static.a
include $(PREBUILT_STATIC_LIBRARY)
```

SETTING UP ANDROID MAKEFILE

STEP 3

Configuration to use CUDA headers and libs:

```
include $(CLEAR_VARS)
LOCAL_STATIC_LIBRARIES := lib_boxfilter libcudart_static

#CUDA project includes
LOCAL_C_INCLUDES += <path/to/cuda/project/headers>
LOCAL_C_INCLUDES += $(CUDA_TOOLKIT_ROOT)/include
include $(BUILD_SHARED_LIBRARY)
```

THANK YOU

- Registration URL:
- http://developer.nvidia.com/register-tegra-registered-developer-program

- Don’t forget to specify the “**Reason for application**” as “**Access to TADP Compute**”

THE PROBLEM

Lots of data available…

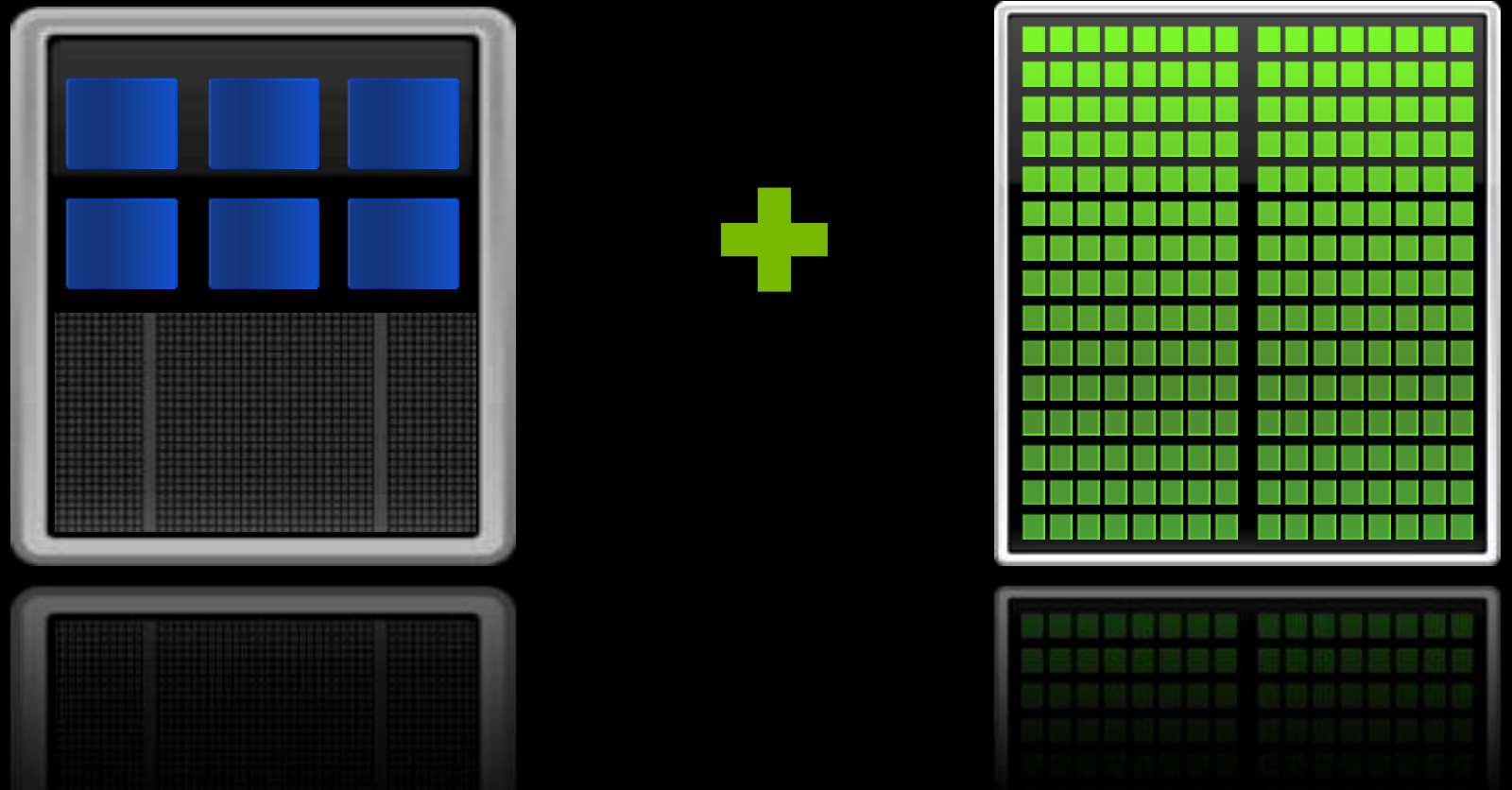
No good way to process it!

PROCESSING MUST BE POWER-EFFICIENT!

HETEROGENEOUS COMPUTING

CPU

Accelerator



SMALLER FOOTPRINT, LOWER POWER & COST

Google Datacenter

1000 CPU Servers

600 kWatts

$5,000,000

Stanford AI Lab

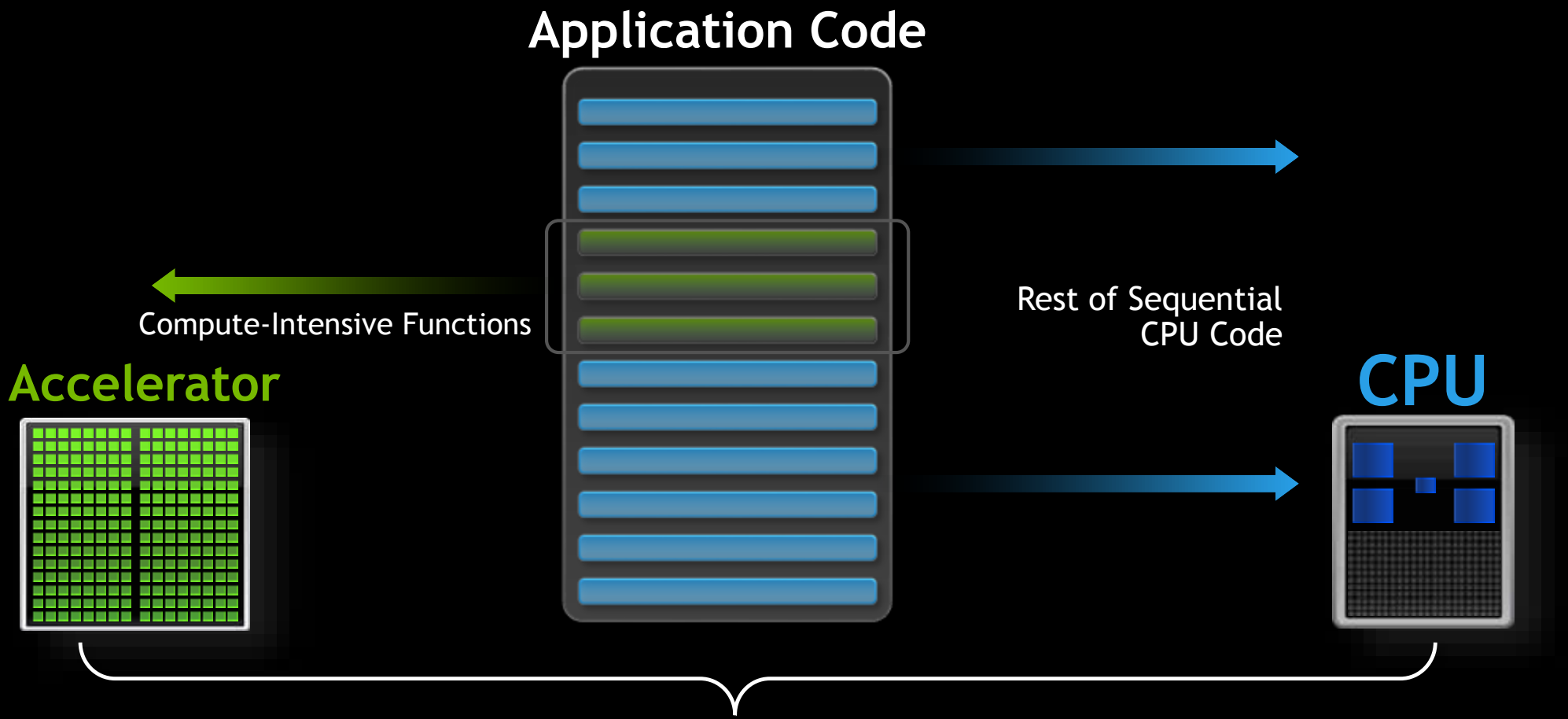
3 GPU-Accelerated Servers

3.6 kWatts

$21,000

Deep learning with COTS HPC systems, A Coates, B Huval, T Wang, D Wu, A Ng, B Catanzaro, NIPS 2013

THE BASIC IDEA



HOW DO I USE GPUS?

CUDA PARALLEL COMPUTING PLATFORM

www.nvidia.com/getcuda

Programming Approaches

- **Libraries** — "Drop-in" Acceleration
- **Directives** — Easily Accelerate Apps
- **Programming Languages** — Maximum Flexibility

Development Environment

Nsight IDE
Linux, Mac and Windows
GPU Debugging and Profiling

CUDA-GDB debugger
NVIDIA Visual Profiler

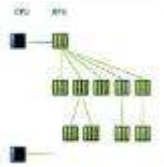
Open Compiler Tool Chain

Enables compiling new languages to CUDA platform, and CUDA languages to other architectures

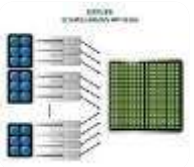
Hardware Capabilities

SMX

Dynamic Parallelism



HyperQ



GPUDirect

GPU COMPUTING IN THE PALM OF YOUR HAND

3 WAYS TO ACCELERATE APPLICATIONS

Applications

Libraries

Directives

Programming Languages

COMPUTER VISION – COIN COUNTING

= $$$?

FUTURE APPLICATIONS

- Realistically animate hair instead of pre-recording it?

FUTURE APPLICATIONS

- Low-cost GPU-equipped Drones

ROBOTICS

From Aldebaran Robotics

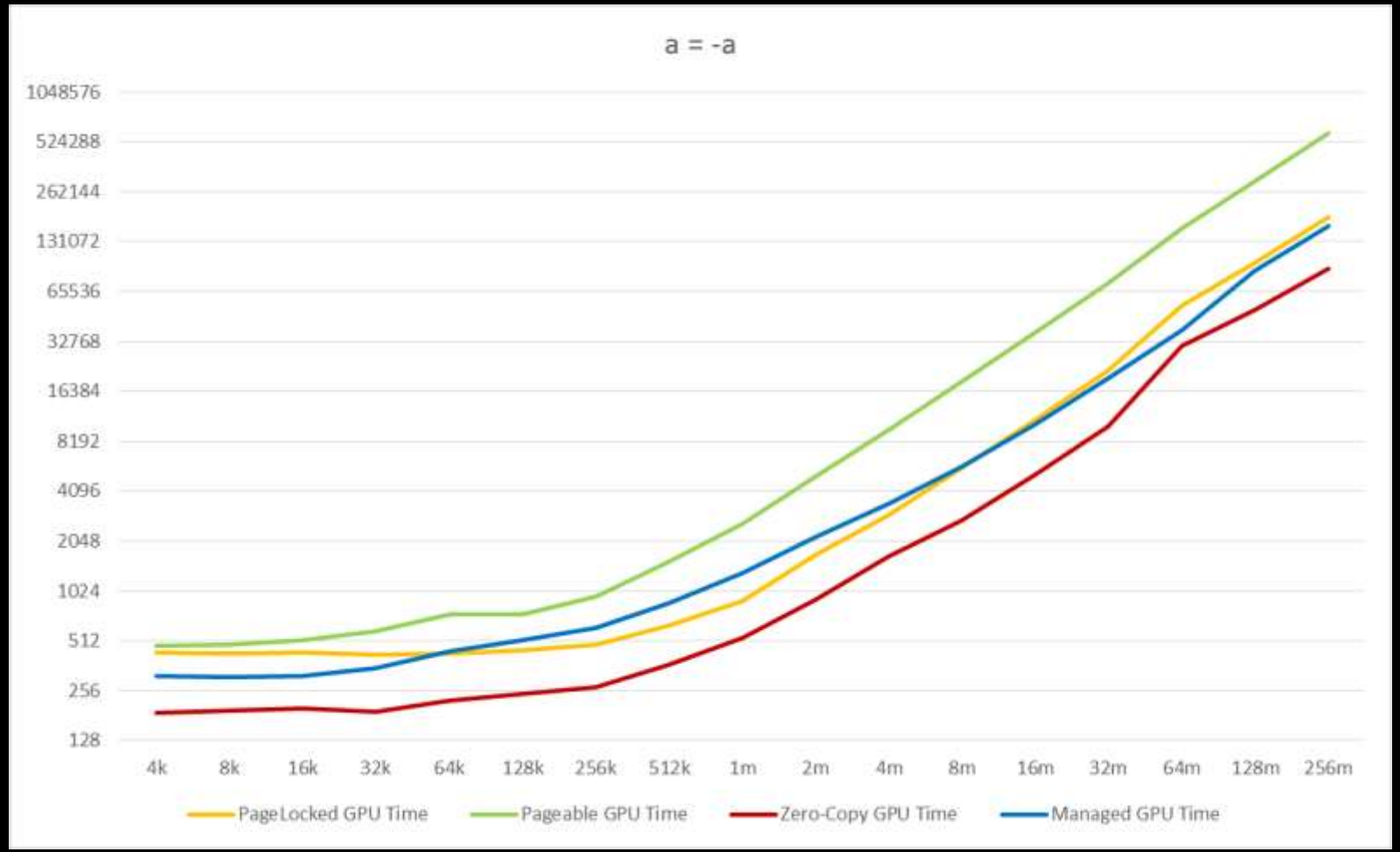
Q & A

- Registration URL:
- http://developer.nvidia.com/register-tegra-registered-developer-program

- Don’t forget to specify the “**Reason for application**” as “**Access to TADP Compute**”

BACKUP

KERNEL (A=-A) – GPU TIME



KERNEL (A=-A) - TOTAL TIME