Webinar II

HPP-Heterogeneous Processing Platform

Programming Models, Performance and Power Profiling for the HPP

Yassine Hariri
Senior Engineer, Platform Design
Hariri@cmc.ca

Hugh W. Pollitt-Smith
Senior System Design Engineer
Pollitt-smith@cmc.ca

2-12-2015

Agenda

- Overview
- emSYSCAN Development Systems Update
- HPP: Heterogeneous processing platform
- HPP GPU
  - GPU Programming
    - Libraries, OpenACC and Programming langages
  - CUDA Development Using NVIDIA Nsight, Eclipse Edition
    - Project Management, Edit, Build, Debug and Profile
  - Power profiling using nvidia-smi
  - Live demo
- HPP FPGA
  - OpenCL for FPGA
  - The AOCL FPGA Programming Flow
  - Power and performance profiling
  - Live demo
- HPP schedule
- Heterogeneous processing workshop

EMSYSCAN DEVELOPMENT SYSTEMS UPDATE

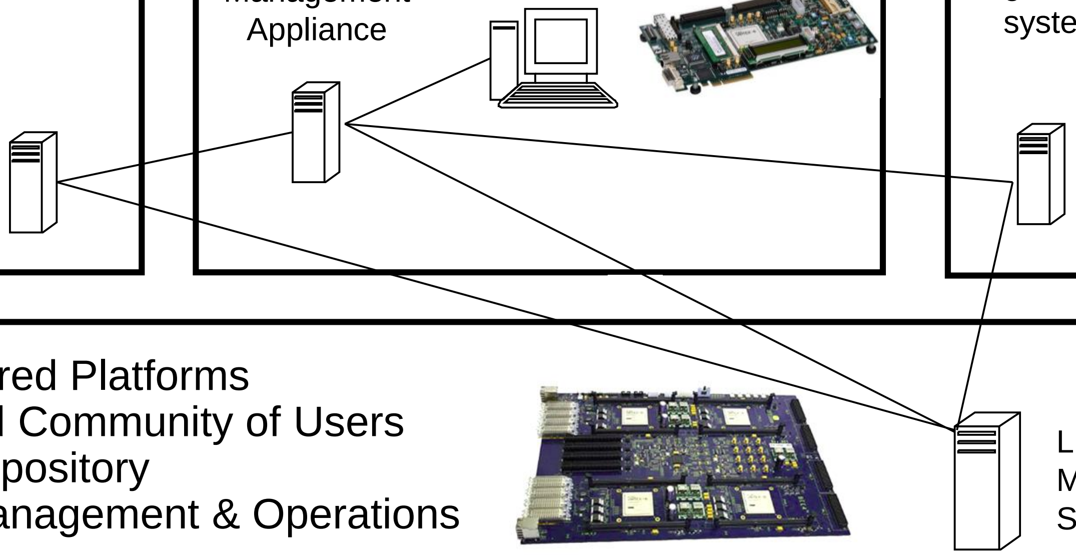
Embedded Systems Canada (emSYSCAN)
$54M investment in Canada's National Design Network
37+ universities, 250+ faculty, 5 years

Microsystems Rapid-Prototyping, Characterization and Integration Labs

4 Universities:
- UBC
- U Manitoba
- Queen's
- École Polytechnique

Multi-Technology Design Environment
- System architecture exploration
- Multi-technology simulation
- Design of custom devices for manufacturing

Development Systems
- System validation and proof-of-concept demonstration


License Management Appliance
Installed Design Environment
Development System Hardware
License Management Server (LMS)

Real-Time Embedded Software Lab

University of Waterloo

- Design, analysis, debug of real-time software on next-generation processor systems

Common, Shared Platforms
Interconnected Community of Users
Knowledge Repository
Centralized Management & Operations

Development Systems for Proof of Concept

Images courtesy of National Instruments, Xilinx, BEECube,, NVIDIA

National Research Platform: Enriched Projects; Results Sooner

- <u>Common</u> set of programmable research platforms with proof of concept features
- <u>Pooled</u> equipment timeshared among users
- <u>Sharing</u> of knowledge on equipment usage
- <u>Adaptive</u> over time in terms of equipment quantities and equipment features
- <u>Large</u> community of users, institutions
- <u>Leveraged</u> industrial partners (e.g., STMicro.)

National project scope and sizeable outcomes enabled by centralized project implementation and management by CMC Microsystems

Installation and usage

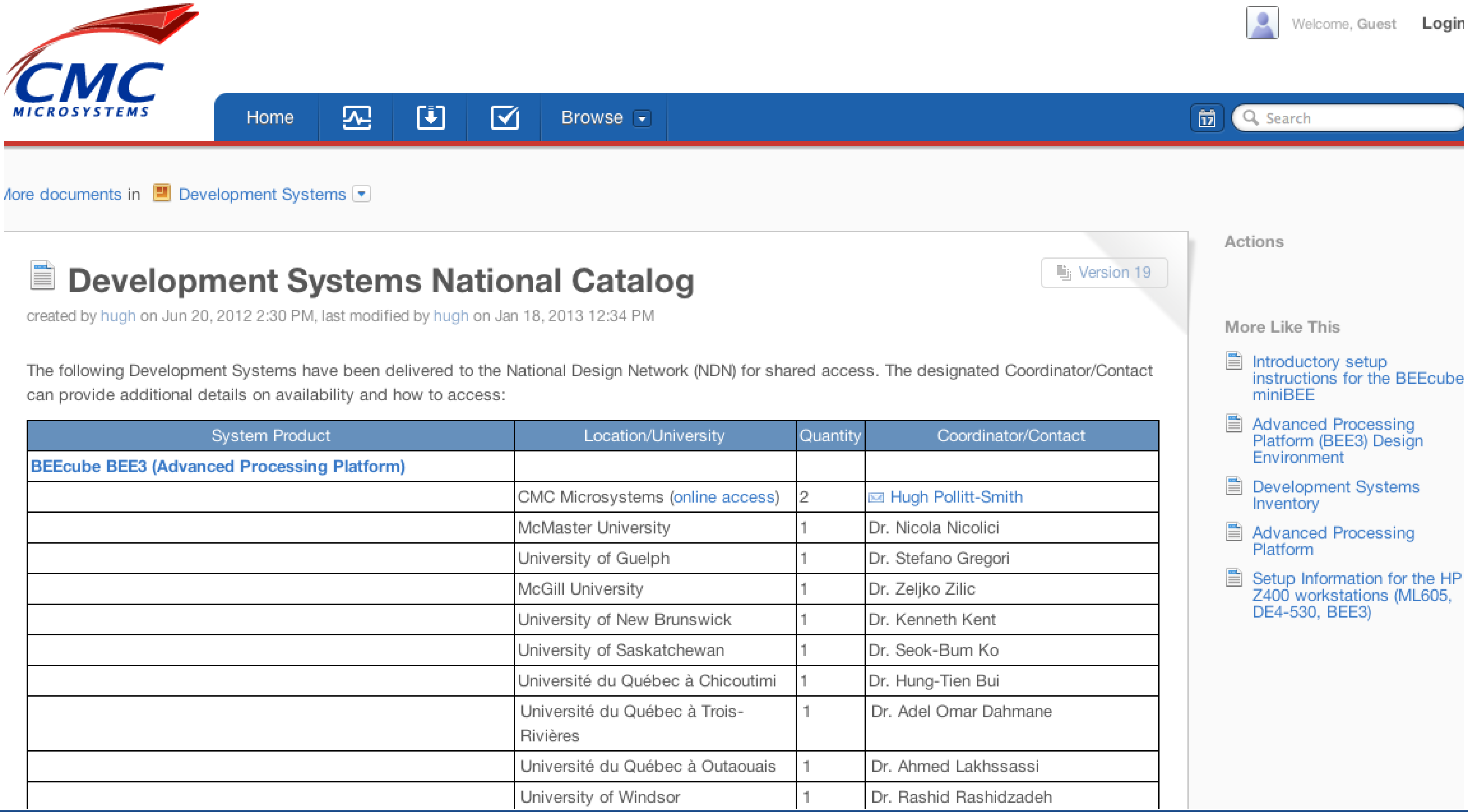
- Shared access systems can be accessed at no charge but require Designer level subscription
  - Subscription provides access to support, tools, reference designs, forums, workshops, travel, select/swap, training, additional discounts
- Systems delivered on site, remote access
- Designated Development System coordinator(s) at each site
  - Communicate institutional needs for purchase specifications
  - Local advocate, information source
  - Encourage participation in National Project

emSYSCAN Development Systems delivered (Gen1)

- Embedded Systems Platform:
  - Xilinx ML605, Altera DE4-530
- Advanced Processing Platform
  - BEEcube BEE3, BEE4, miniBEE
- Software-Defined Radio Platform
  - BEEcube miniBEE, RF daughtercard
- Simulation Acceleration Platform
  - Nallatech P385-D5 (Altera Stratix V, OpenCL)
- Multiprocessor Array Platform
  - NVIDIA Tesla K20 GPU
  - Intel Xeon Phi
- Microsystems Integration Platform
  - National Instruments PXI-based, FPGA, MEMS, microfluidics, RF, photonics features

NDN Development Systems Community

https://community.cmc.ca/community/development-systems



Development Systems National Catalog

created by hugh on Jun 20, 2012 2:30 PM, last modified by hugh on Jan 18, 2013 12:34 PM

The following Development Systems have been delivered to the National Design Network (NDN) for shared access. The designated Coordinator/Contact can provide additional details on availability and how to access:

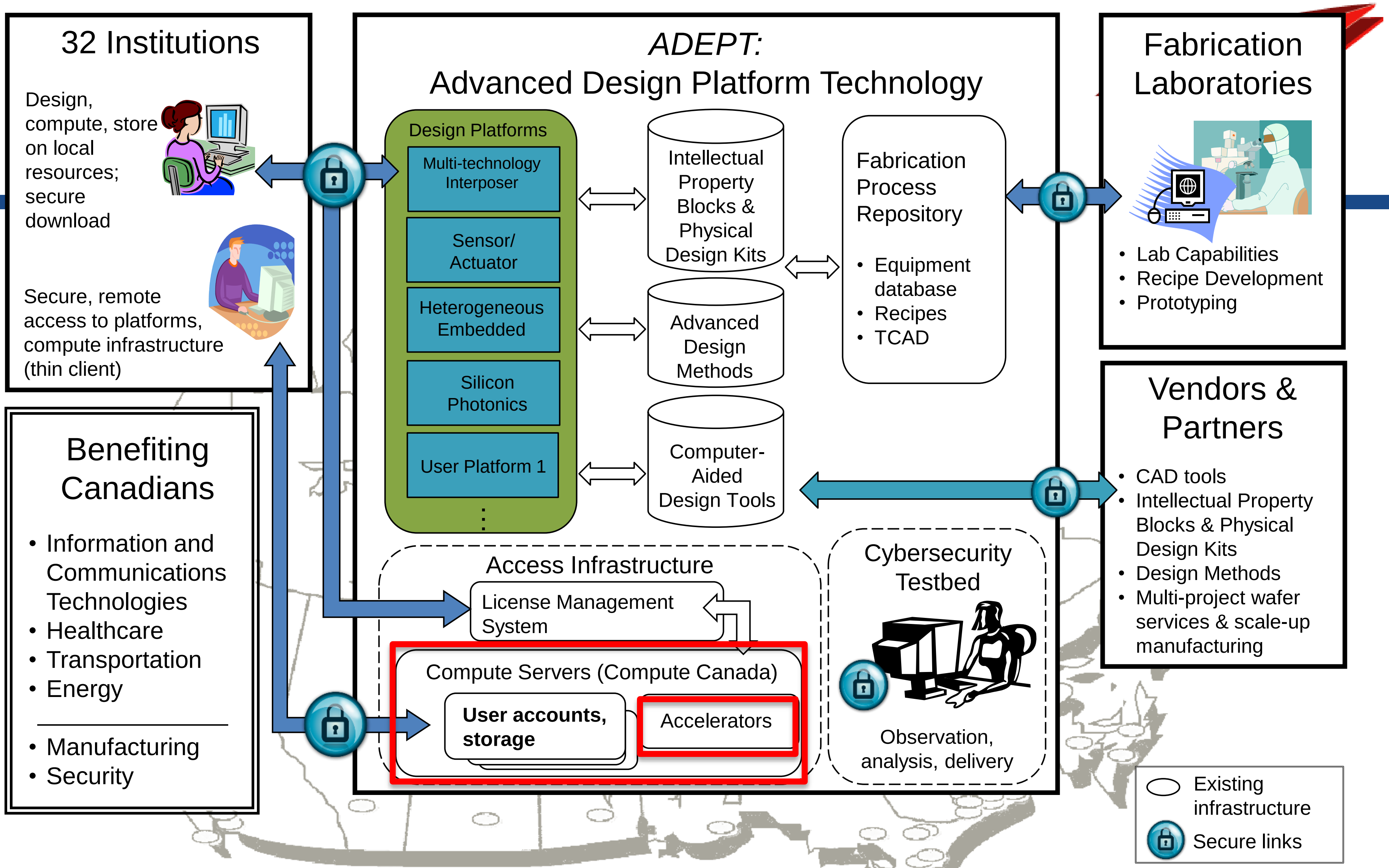
System Product	Location/University	Quantity	Coordinator/Contact
BEEcube BEE3 (Advanced Processing Platform)			
	CMC Microsystems (online access)	2	Hugh Pollitt-Smith
	McMaster University	1	Dr. Nicola Nicolici
	University of Guelph	1	Dr. Stefano Gregori
	McGill University	1	Dr. Zeljko Zilic
	University of New Brunswick	1	Dr. Kenneth Kent
	University of Saskatchewan	1	Dr. Seok-Bum Ko
	Université du Québec à Chicoutimi	1	Dr. Hung-Tien Bui
	Université du Québec à Trois-Rivières	1	Dr. Adel Omar Dahmane
	Université du Québec à Outaouais	1	Dr. Ahmed Lakhssassi
	University of Windsor	1	Dr. Rashid Rashidzadeh

More Like This

- Introductory setup instructions for the BEEcube miniBEE
- Advanced Processing Platform (BEE3) Design Environment
- Development Systems Inventory
- Advanced Processing Platform
- Setup Information for the HP Z400 workstations (ML605, DE4-530, BEE3)

Upcoming emSYSCAN Development Systems deployments

- Embedded Systems Platform
  - Xilinx Virtex-7, Ultrascale, Zynq options
  - Altera Arria 10 and Arria 10 SoC options
  - Shipping Q1 2016
- Advanced Processing Platform
  - RFP currently in evaluation
  - Shipping Q1/Q2 2016
- Software-defined Radio
  - BEEcube nano/megaBEE (2x2 up to 16x16 MIMO options)
  - Shipping Q1 2016


32 Institutions
Design, compute, store on local resources; secure download
Secure, remote access to platforms, compute infrastructure (thin client)
ADEPT:
Advanced Design Platform Technology
Design Platforms
Multi-technology Interposer
Sensor/ Actuator
Heterogeneous Embedded
Silicon Photonics
User Platform 1
⋮
Intellectual Property Blocks & Physical Design Kits
Advanced Design Methods
Computer-Aided Design Tools
Fabrication Process Repository
• Equipment database
• Recipes
• TCAD
Fabrication Laboratories
• Lab Capabilities
• Recipe Development
• Prototyping
Vendors & Partners
• CAD tools
• Intellectual Property Blocks & Physical Design Kits
• Design Methods
• Multi-project wafer services & scale-up manufacturing
Benefiting Canadians
• Information and Communications Technologies
• Healthcare
• Transportation
• Energy
• Manufacturing
• Security
Access Infrastructure
License Management System
Compute Servers (Compute Canada)
User accounts, storage
Accelerators
Cybersecurity Testbed
Observation, analysis, delivery
Existing infrastructure
Secure links

Canada's National Design Network – ADEPT Management & Operations

Includes software procurement, configuration, installation and delivery. Access and utilization management, engineering/technical support. Cybersecurity installations, secure testbed assistance and demonstrations. Train-the-trainer events. Advisory Group coordination. Governance, reporting, legal and financial administration.

HPP Distribution

- Based on Development Systems Coordinator consultations in April 2014:
  - Generation 1 (2014/15): 18 systems
    - USask, UQTR, Outaouais, McGill, York, Windsor, Waterloo, Western, Ottawa, Ryerson, RMC, Victoria
  - Generation 2 (2016/17): 12 systems
    - Memorial, Guelph, McMaster, Toronto, Polytechnique, UQTR, Outaouais

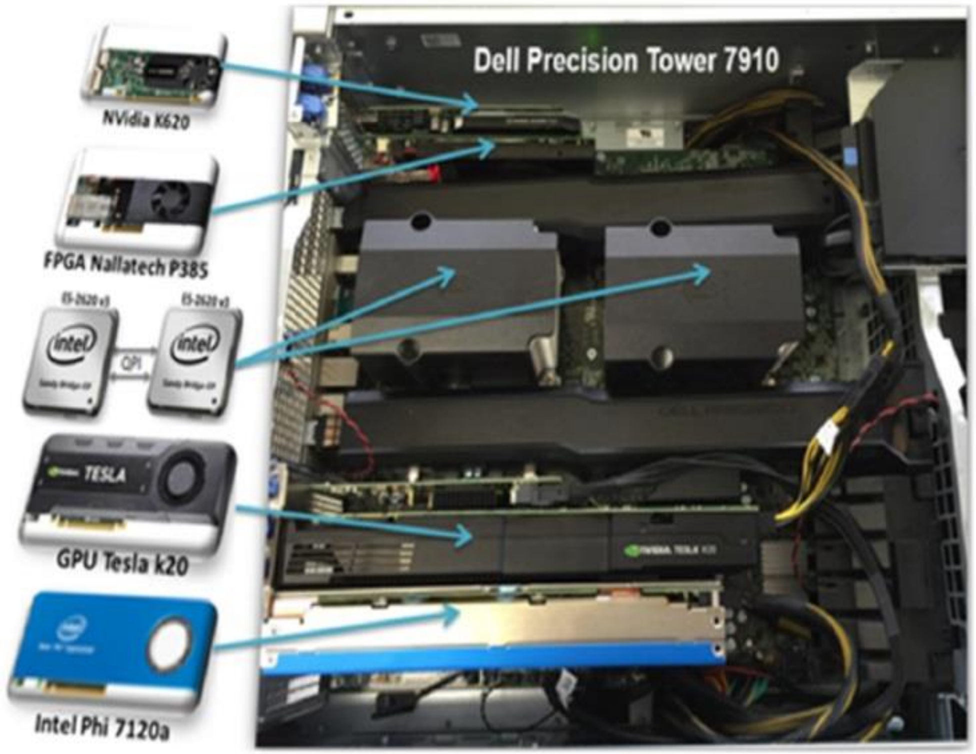
CMC
MICROSYSTEMS

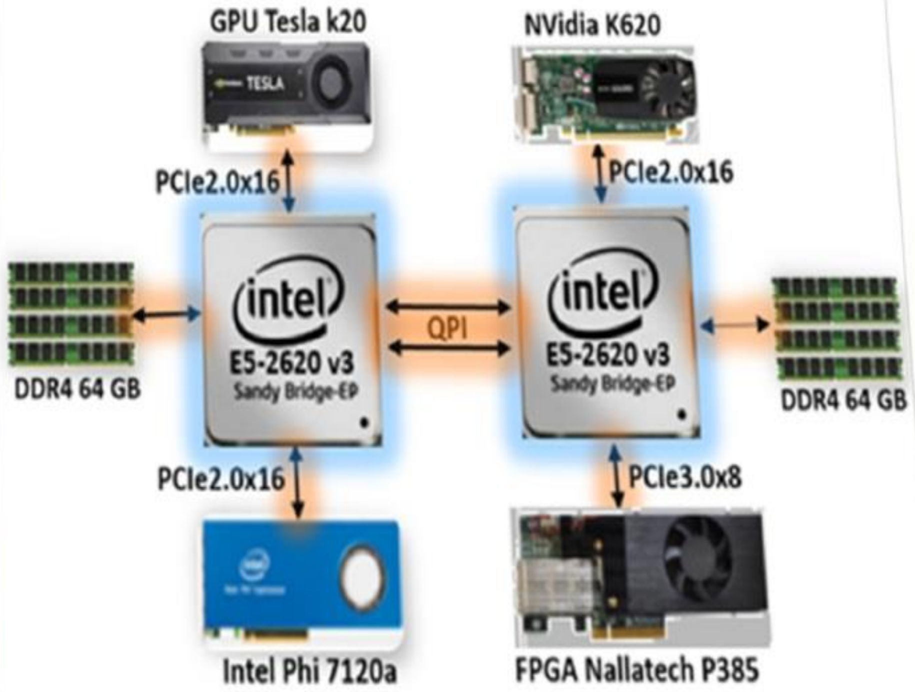
HPP: HETEROGENEOUS PROCESSING PLATFORM

HPP main components

- The HPP workstation integrates the following main components:
  - Dual core Intel Xeon E5-2620 V3
  - NVidia GPU (Tesla K20)
  - FPGA board (Nallatech P385-A72).
  - Xeon Phi 7120A
- Key Platform Benefits
  - Customizability: Select the right mix of accelerators for your application
  - Greater flexibility for HW/SW exploration
  - Scalability: Create one node and scale up by adding more nodes
  - Fast automated setup and configuration
  - Faster path to commercialization
  - Technical support and training from CMC Microsystems

HPP fully installed system





HPP Pre-Installed Software Components

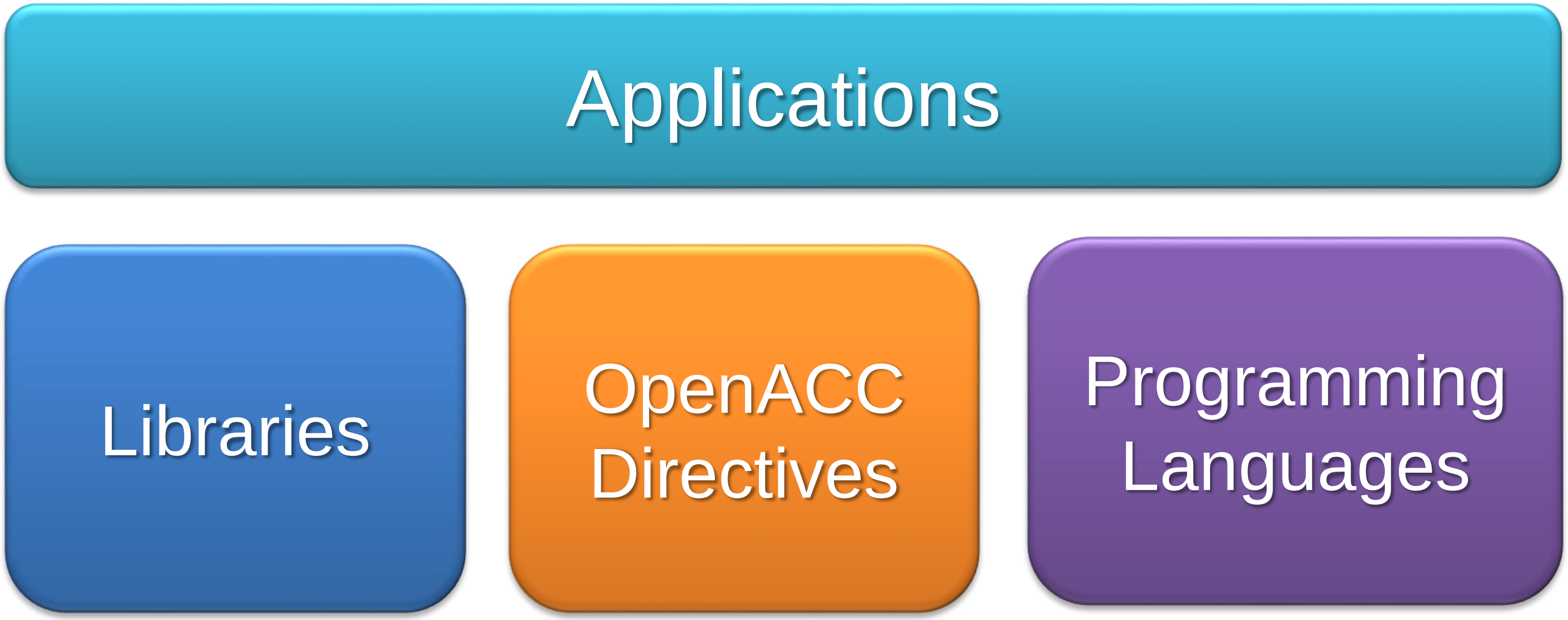
These software components required by the HPP are pre-installed on the workstation:

- RedHat Enterprise Linux 6.6 (Kernel version: 2.6.32-504.el6.x86_64)
- Java Runtime Environment (JRE)
- gcc compiler and toolchain

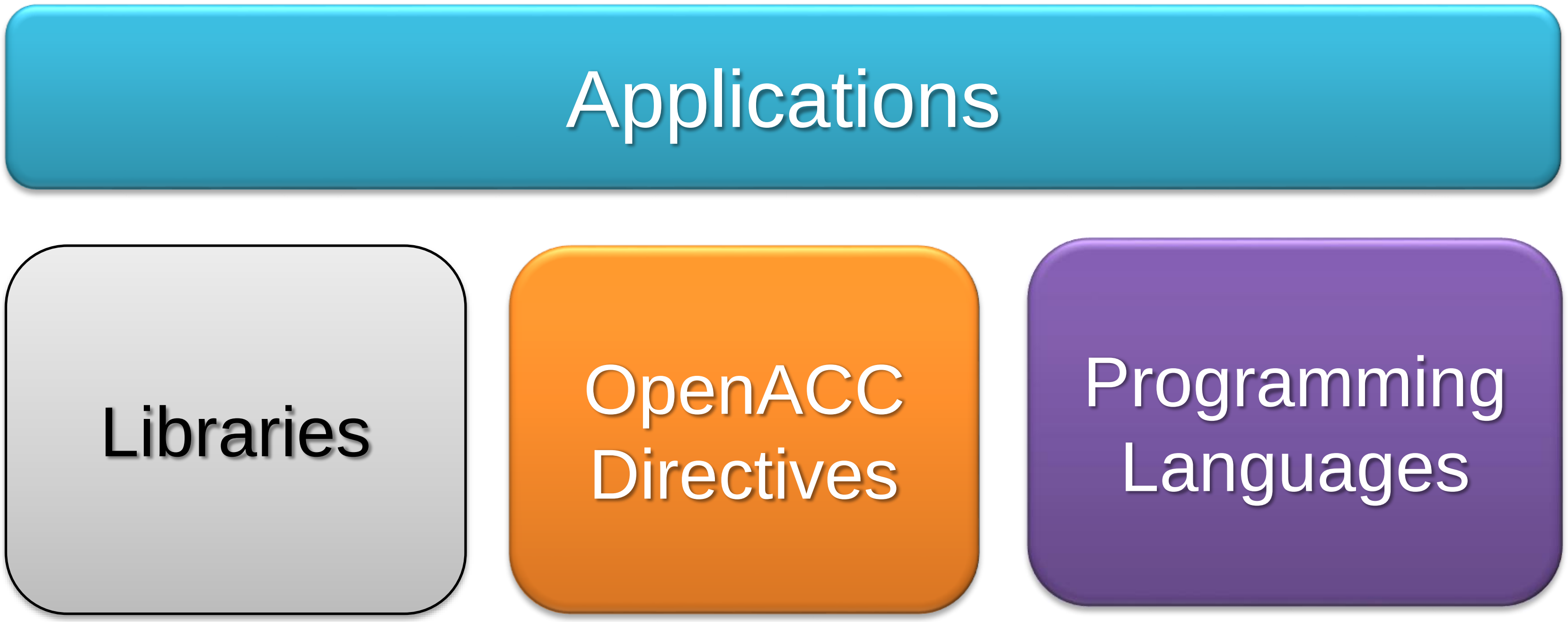
Accelerator	Software and tools
GPU Tesla K20	• NVIDIA CUDA 7 Toolkit
Nallatech FPGA P385	• Altera Quartus 15.0, Altera SDK for OpenCL 1.0 • Nallatech FPGA P385 Board Support Package
Xeon Phi 7120a	• Intel Manycore Platform Software Stack (MPSS) 3.5.1 for Linux • Intel Parallel Studio XE 2015, Professional Studio for C++, Linux version

HPP: GPU

Applications Acceleration



Applications Acceleration



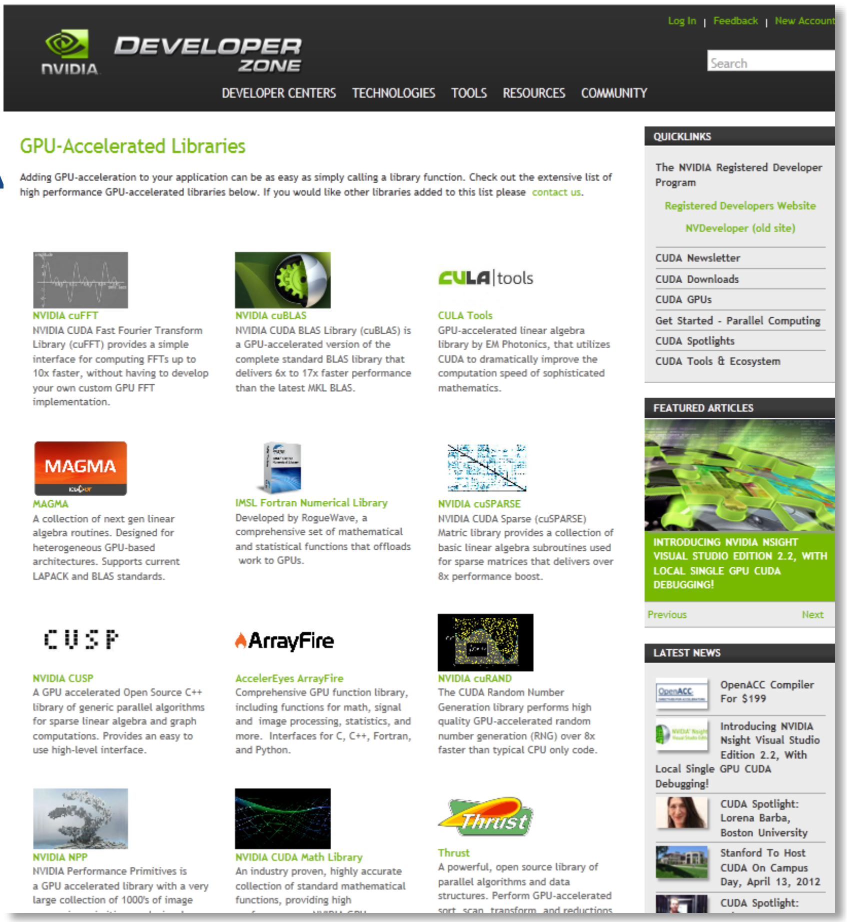
Libraries

- **Ease of use:** Deep knowledge of GPU programming is not required
- **“Drop-in”:** Standard APIs, minimal code changes
- **Quality:** High-quality implementations
- **Performance:** NVIDIA libraries are highly optimized

CUDA Libraries Ecosystem

- CUDA Tools and Ecosystem described in detail on NVIDIA Developer Zone:

  developer.nvidia.com/cuda-tools-ecosystem



GPU-Accelerated Libraries

Adding GPU-acceleration to your application can be as easy as simply calling a library function. Check out the extensive list of high performance GPU-accelerated libraries below. If you would like other libraries added to this list please contact us.

NVIDIA cuFFT
NVIDIA CUDA Fast Fourier Transform Library (cuFFT) provides a simple interface for computing FFTs up to 10x faster, without having to develop your own custom GPU FFT implementation.

NVIDIA cuBLAS
NVIDIA CUDA BLAS Library (cuBLAS) is a GPU-accelerated version of the complete standard BLAS library that delivers 6x to 17x faster performance than the latest MKL BLAS.

CULA Tools
GPU-accelerated linear algebra library by EM Photonics, that utilizes CUDA to dramatically improve the computation speed of sophisticated mathematics.

MAGMA
A collection of next gen linear algebra routines. Designed for heterogeneous GPU-based architectures. Supports current LAPACK and BLAS standards.

IMSL Fortran Numerical Library
Developed by RogueWave, a comprehensive set of mathematical and statistical functions that offloads work to GPUs.

NVIDIA cuSPARSE
NVIDIA CUDA Sparse (cuSPARSE) Matric library provides a collection of basic linear algebra subroutines used for sparse matrices that delivers over 8x performance boost.

NVIDIA CUSP
A GPU accelerated Open Source C++ library of generic parallel algorithms for sparse linear algebra and graph computations. Provides an easy to use high-level interface.

AccelerEyes ArrayFire
Comprehensive GPU function library, including functions for math, signal and image processing, statistics, and more. Interfaces for C, C++, Fortran, and Python.

NVIDIA cuRAND
The CUDA Random Number Generation library performs high quality GPU-accelerated random number generation (RNG) over 8x faster than typical CPU only code.

NVIDIA NPP
NVIDIA Performance Primitives is a GPU accelerated library with a very large collection of 1000's of image

NVIDIA CUDA Math Library
An industry proven, highly accurate collection of standard mathematical functions, providing high

Thrust
A powerful, open source library of parallel algorithms and data structures. Perform GPU-accelerated sort, scan, transform, and reductions

QUICKLINKS

The NVIDIA Registered Developer Program

Registered Developers Website

NVDeveloper (old site)

CUDA Newsletter
CUDA Downloads
CUDA GPUs
Get Started - Parallel Computing
CUDA Spotlights
CUDA Tools & Ecosystem

FEATURED ARTICLES

INTRODUCING NVIDIA NSIGHT VISUAL STUDIO EDITION 2.2, WITH LOCAL SINGLE GPU CUDA DEBUGGING!

Previous Next

LATEST NEWS

OpenACC Compiler For $199

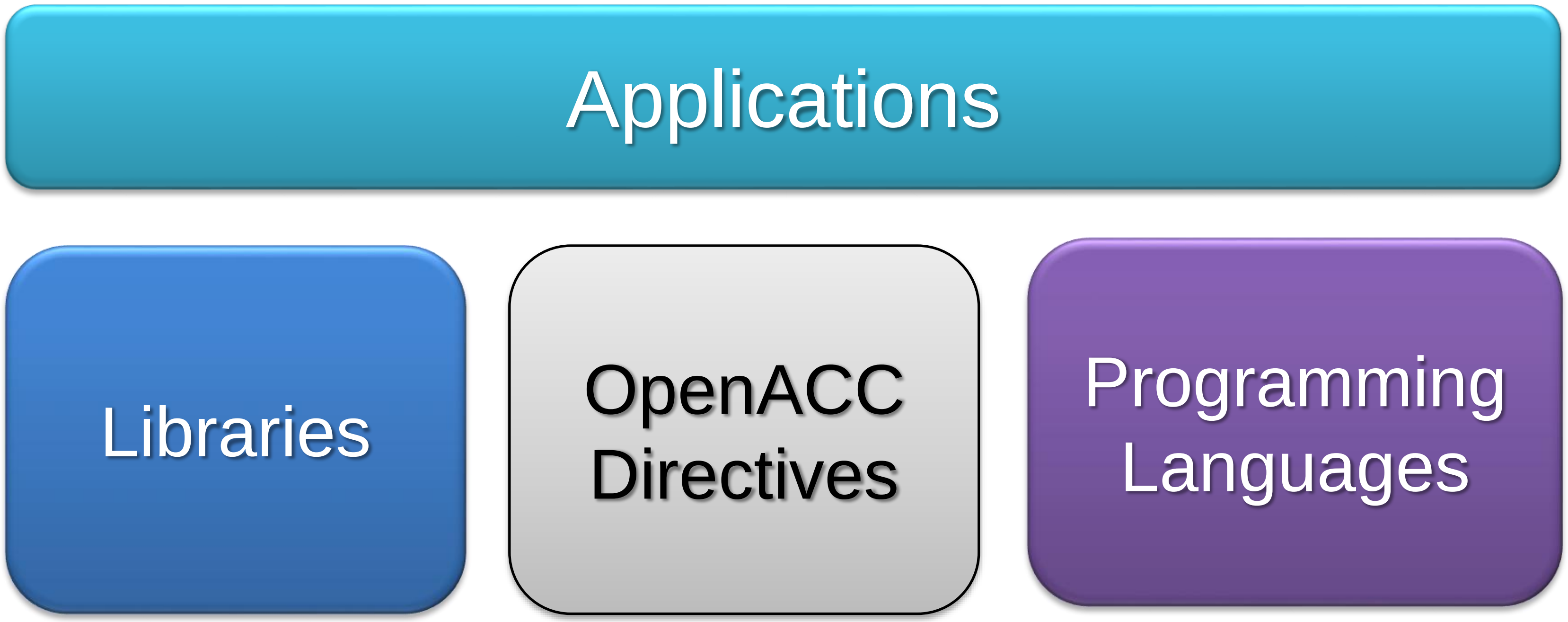
Introducing NVIDIA Nsight Visual Studio Edition 2.2, With Local Single GPU CUDA Debugging!

CUDA Spotlight: Lorena Barba, Boston University

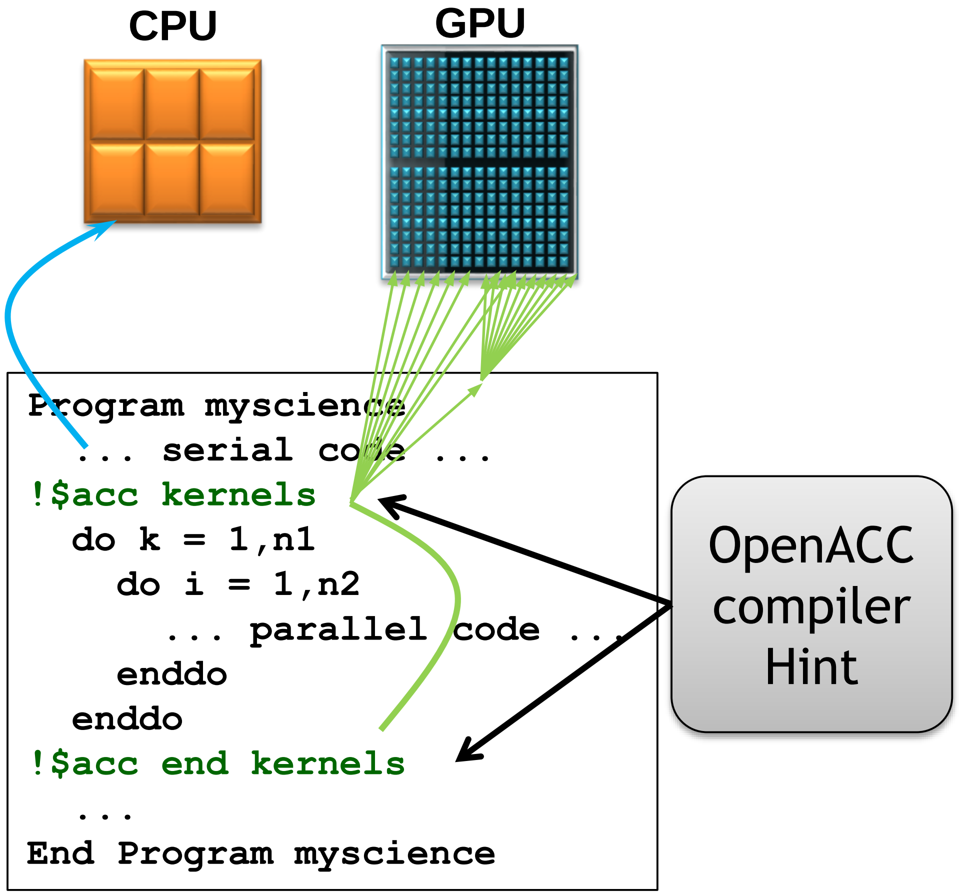
Stanford To Host CUDA On Campus Day, April 13, 2012

CUDA Spotlight:

Applications Acceleration



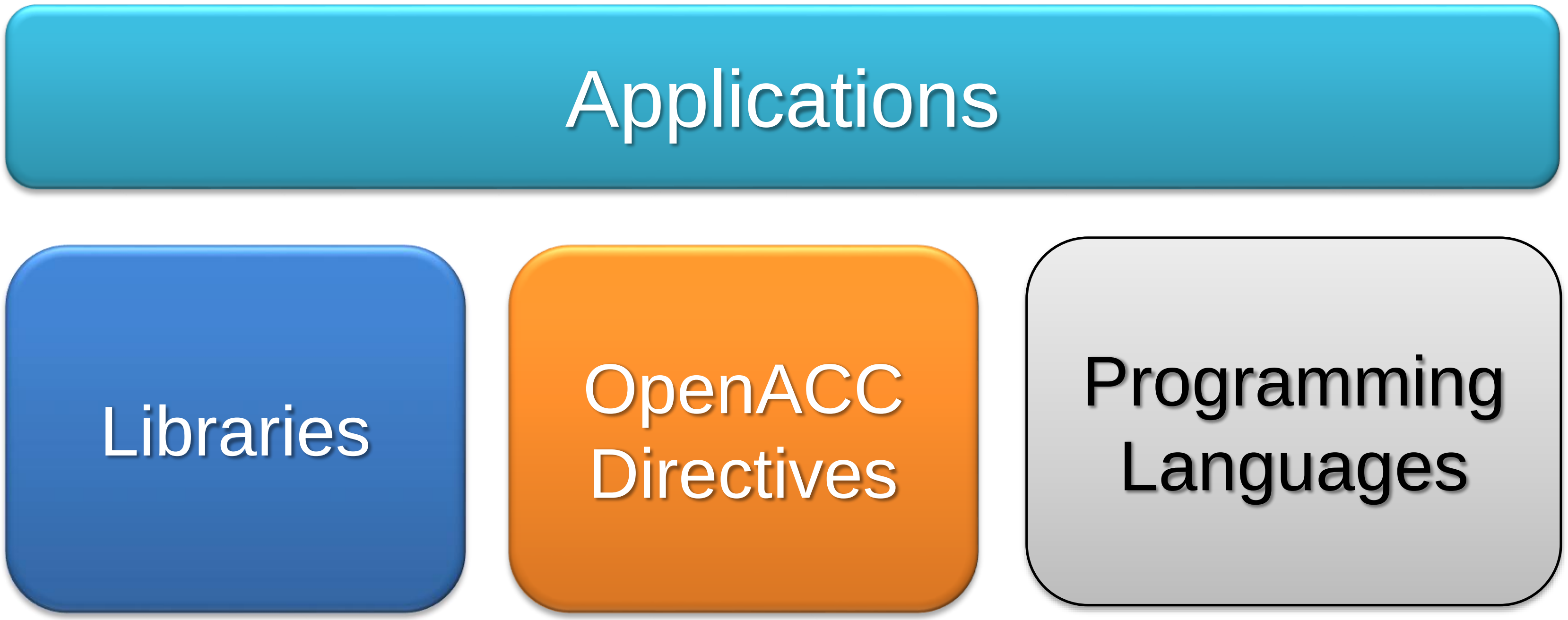
OpenACC Directives



```
Program myscience
   ... serial code ...
!$acc kernels
   do k = 1,n1
      do i = 1,n2
         ... parallel code ...
      enddo
   enddo
!$acc end kernels
   ...
End Program myscience
```

Original Code

Applications Acceleration



GPU Programming Languages

- Numerical analytics: MATLAB, Mathematica, LabVIEW
- Fortran: OpenACC, CUDA Fortran
- C: OpenACC, CUDA C
- C++: Thrust, CUDA C++
- Python: PyCUDA, Copperhead

HPP GPU : CUDA DEVELOPMENT USING NVIDIA NSIGHT, ECLIPSE EDITION

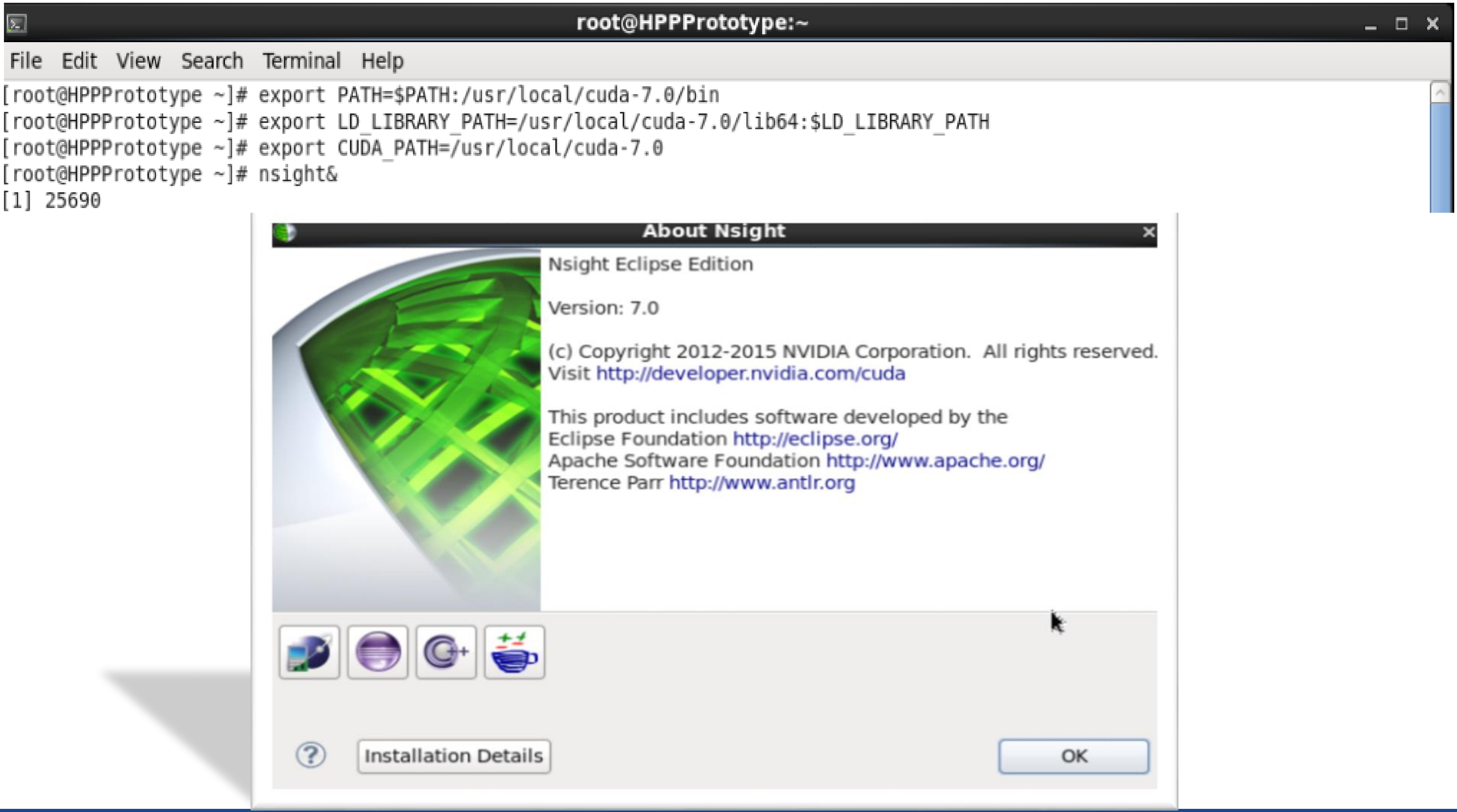
NVIDIA® Nsight™ Eclipse Edition

- CUDA Integrated Development Environment
- Project Management
- Edit
- Build
- Debug
- Profile

Powered By Eclipse

- Extensible via robust selection of open-source and commercial plugins
- Revision control: CVS, SVN, Git, Perforce, …
- Issue tracking
- …
- Strong cross-platform support
- Nsight Eclipse Edition available for Linux and Mac OSX

Included In CUDA Toolkit



```
[root@HPPPrototype ~]# export PATH=$PATH:/usr/local/cuda-7.0/bin
[root@HPPPrototype ~]# export LD_LIBRARY_PATH=/usr/local/cuda-7.0/lib64:$LD_LIBRARY_PATH
[root@HPPPrototype ~]# export CUDA_PATH=/usr/local/cuda-7.0
[root@HPPPrototype ~]# nsight&
[1] 25690
```

About Nsight

Nsight Eclipse Edition

Version: 7.0

(c) Copyright 2012-2015 NVIDIA Corporation. All rights reserved.
Visit http://developer.nvidia.com/cuda

This product includes software developed by the
Eclipse Foundation http://eclipse.org/
Apache Software Foundation http://www.apache.org/
Terence Parr http://www.antlr.org

NVIDIA® Nsight™ Eclipse Edition

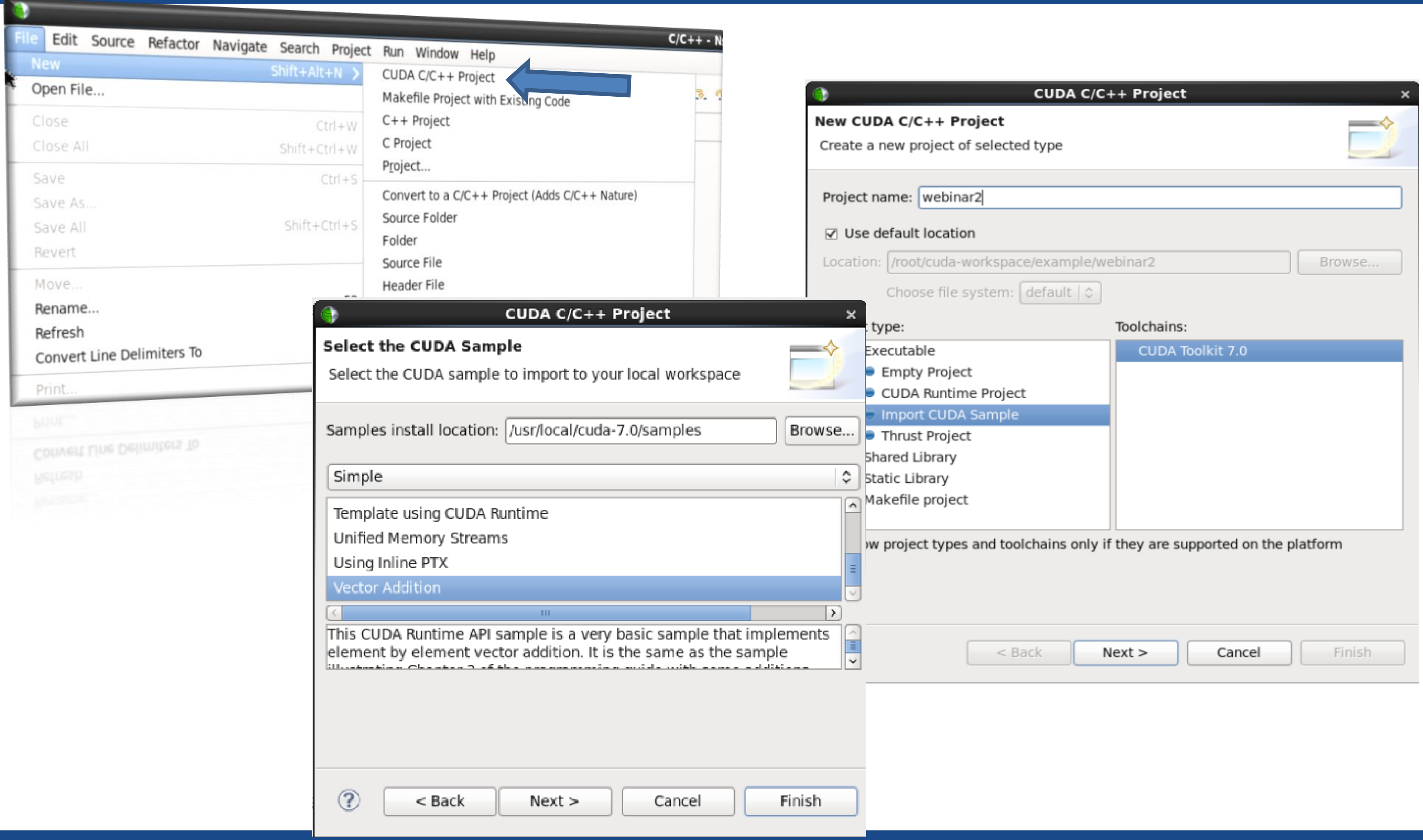
- CUDA Integrated Development Environment
  - Project Management
  - Edit
  - Build
  - Debug
  - Profile

Nsight Project Support

- CUDA C / C++
- Project Types
  - Executable
  - Shared Library
  - Static Library

- New vs. Existing
  - New project, build managed by Nsight
  - Existing project, Nsight can use your Makefile

Creating A New CUDA Project



CUDA C/C++ Project

New CUDA C/C++ Project

Create a new project of selected type

Project name: webinar2

☑ Use default location

Location: /root/cuda-workspace/example/webinar2 Browse...

Choose file system: default

type:
- Executable
  - Empty Project
  - CUDA Runtime Project
  - Import CUDA Sample
  - Thrust Project
- Shared Library
- Static Library
- Makefile project

Toolchains:
- CUDA Toolkit 7.0

w project types and toolchains only if they are supported on the platform

< Back | Next > | Cancel | Finish

CUDA C/C++ Project

Select the CUDA Sample

Select the CUDA sample to import to your local workspace

Samples install location: /usr/local/cuda-7.0/samples Browse...

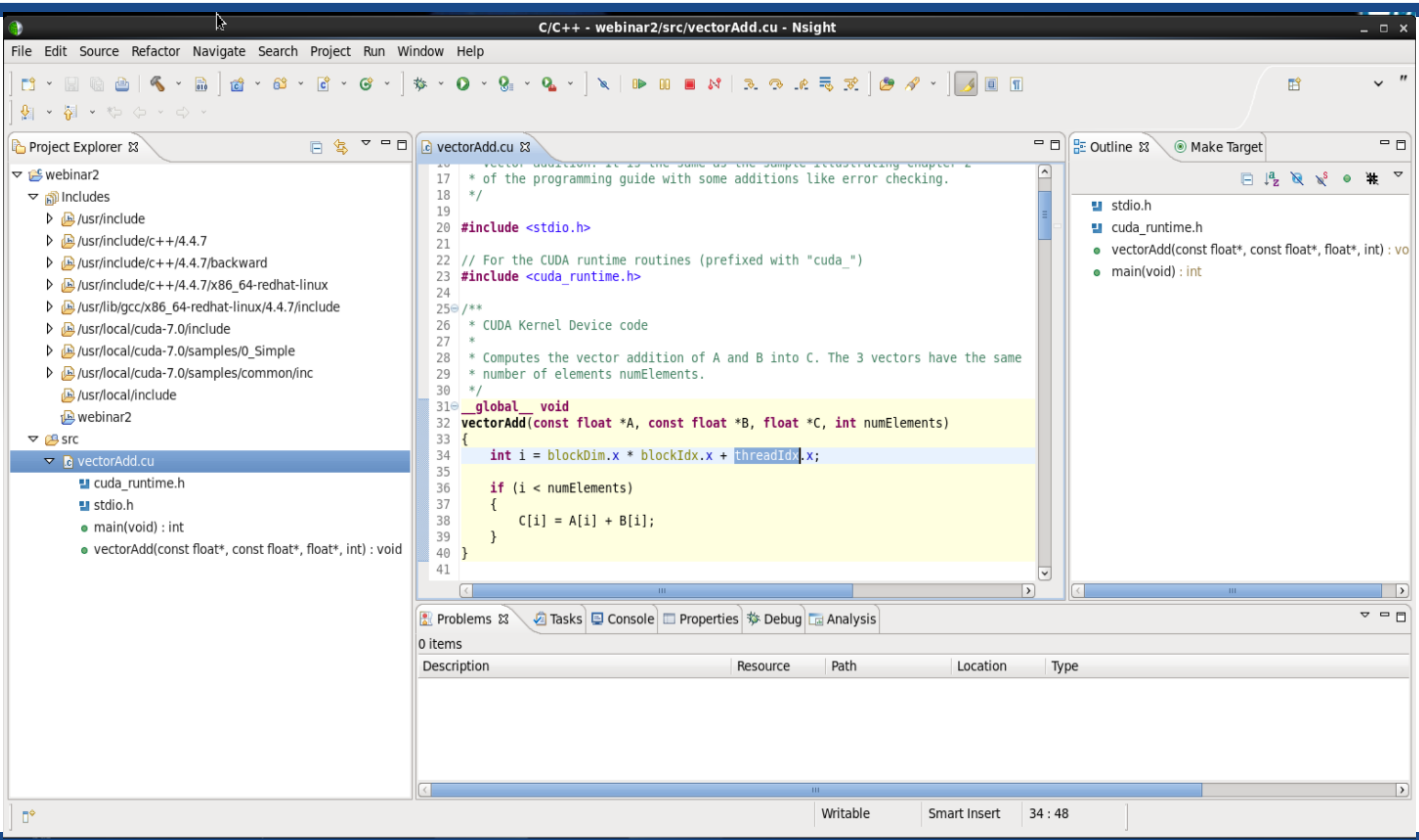
Simple

- Template using CUDA Runtime
- Unified Memory Streams
- Using Inline PTX
- Vector Addition

This CUDA Runtime API sample is a very basic sample that implements element by element vector addition. It is the same as the sample

< Back | Next > | Cancel | Finish

Nsight main window after creating a new project



C/C++ - webinar2/src/vectorAdd.cu - Nsight
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
webinar2
Includes
/usr/include
/usr/include/c++/4.4.7
/usr/include/c++/4.4.7/backward
/usr/include/c++/4.4.7/x86_64-redhat-linux
/usr/lib/gcc/x86_64-redhat-linux/4.4.7/include
/usr/local/cuda-7.0/include
/usr/local/cuda-7.0/samples/0_Simple
/usr/local/cuda-7.0/samples/common/inc
/usr/local/include
webinar2
src
vectorAdd.cu
cuda_runtime.h
stdio.h
main(void) : int
vectorAdd(const float*, const float*, float*, int) : void
vectorAdd.cu
Outline
Make Target
stdio.h
cuda_runtime.h
vectorAdd(const float*, const float*, float*, int) : vo
main(void) : int
Problems
Tasks
Console
Properties
Debug
Analysis
0 items
Description
Resource
Path
Location
Type
Writable
Smart Insert
34 : 48

```
 * of the programming guide with some additions like error checking.
 */

#include <stdio.h>

// For the CUDA runtime routines (prefixed with "cuda_")
#include <cuda_runtime.h>

/**
 * CUDA Kernel Device code
 *
 * Computes the vector addition of A and B into C. The 3 vectors have the same
 * number of elements numElements.
 */
__global__ void
vectorAdd(const float *A, const float *B, float *C, int numElements)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements)
    {
        C[i] = A[i] + B[i];
    }
}
```

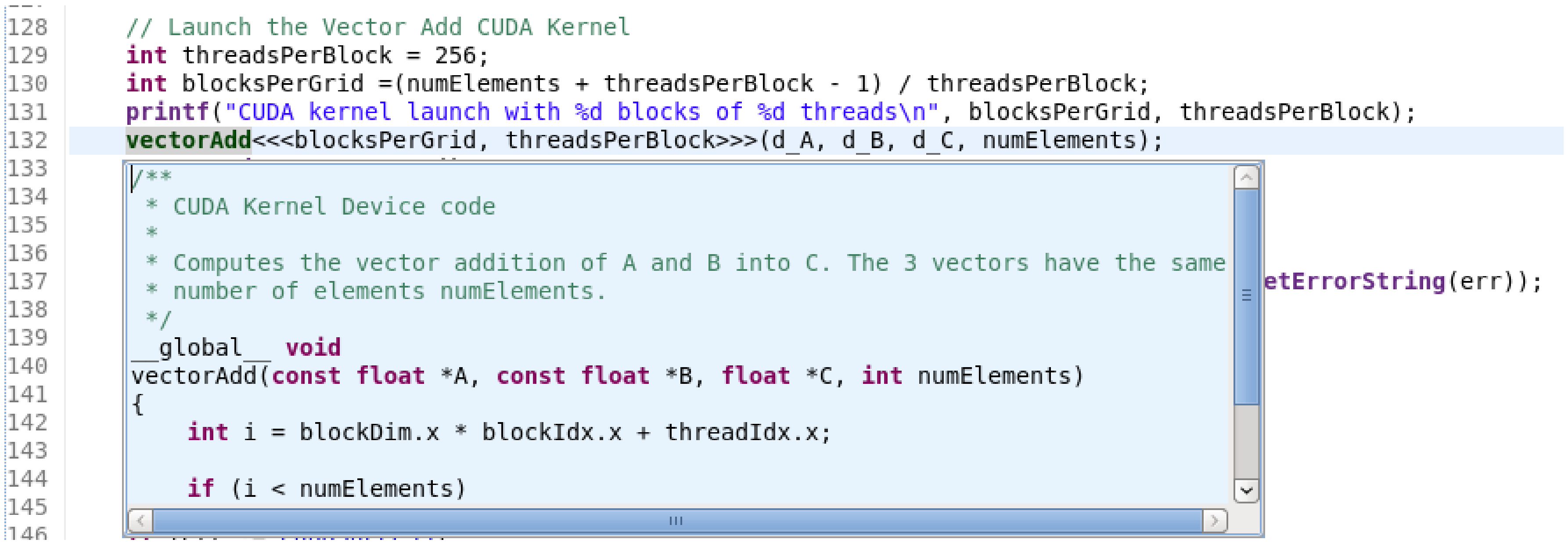
12/16/2015

Edit

- CUDA Integrated Development Environment
  - Project Management
  - Edit
  - Build
  - Debug
  - Profile

CUDA Editor

- CUDA-aware syntax highlighting
- Host / Device code highlighting
- Smart code assist
- As-you-type error detection
- CUDA API documentation pop-ups
- Automatic code refactoring



```
    // Launch the Vector Add CUDA Kernel
    int threadsPerBlock = 256;
    int blocksPerGrid =(numElements + threadsPerBlock - 1) / threadsPerBlock;
    printf("CUDA kernel launch with %d blocks of %d threads\n", blocksPerGrid, threadsPerBlock);
    vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, numElements);
```

```
/**
 * CUDA Kernel Device code
 *
 * Computes the vector addition of A and B into C. The 3 vectors have the same
 * number of elements numElements.
 */
__global__ void
vectorAdd(const float *A, const float *B, float *C, int numElements)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements)
```

```
etErrorString(err));
```

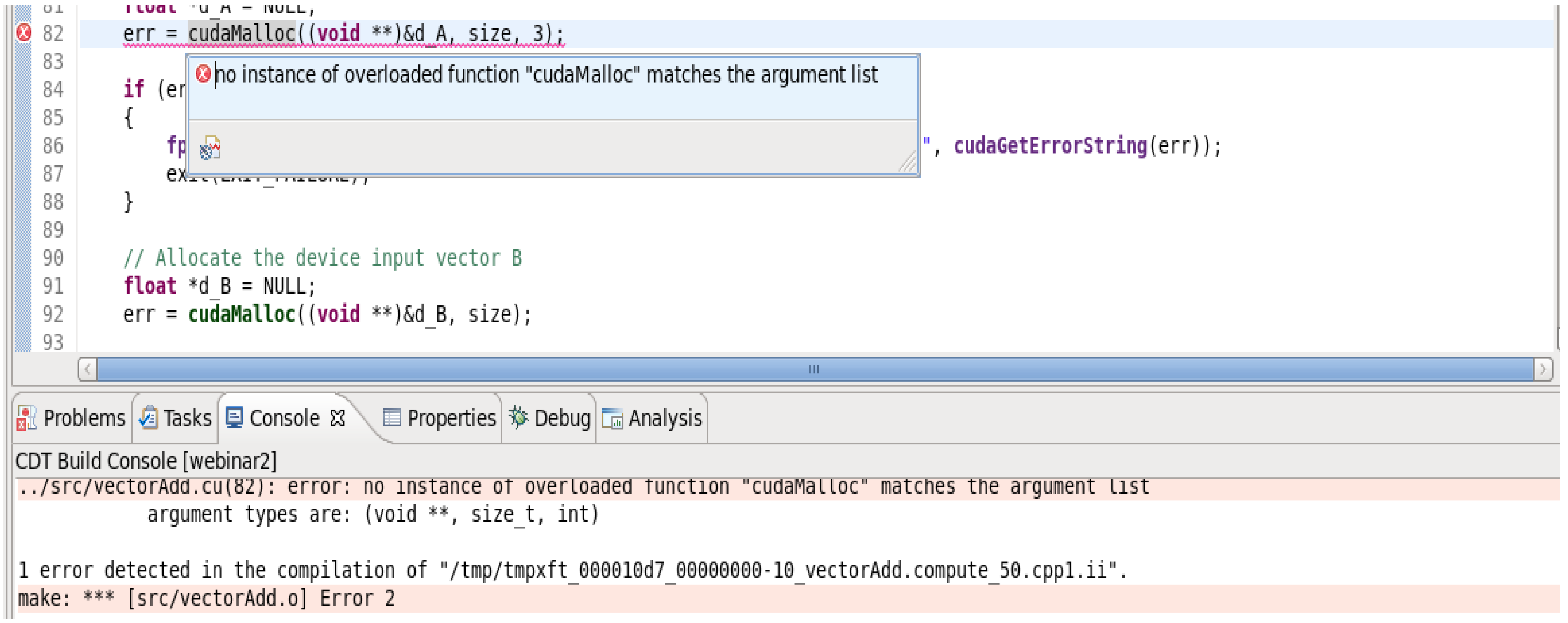
Edit

- CUDA Integrated Development Environment
  - Project Management
  - Edit
  - Build
  - Debug
  - Profile

CUDA Builder

- Full CUDA toolchain support
  - All nvcc features
  - Debug, release, and custom build configurations
- Dependent project support
  - Static libraries
  - Shared libraries
  - Manages all build dependencies
- Source-correlated error reporting

Build Error Reporting



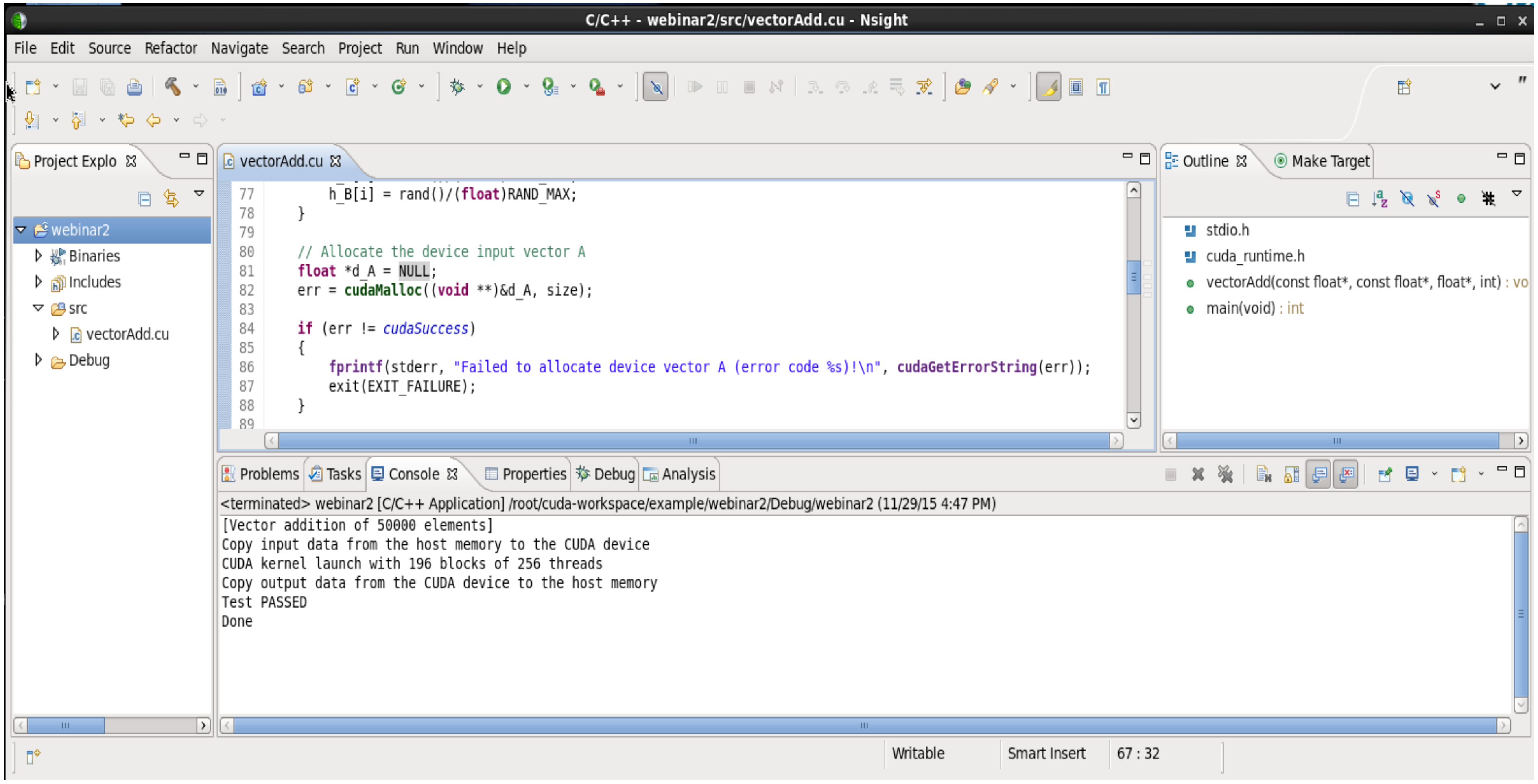
```
    float *d_A = NULL;
    err = cudaMalloc((void **)&d_A, size, 3);

    if (er
    {
        fp                                                              ", cudaGetErrorString(err));
        ex
    }

    // Allocate the device input vector B
    float *d_B = NULL;
    err = cudaMalloc((void **)&d_B, size);
```

no instance of overloaded function "cudaMalloc" matches the argument list

CDT Build Console [webinar2]

```
../src/vectorAdd.cu(82): error: no instance of overloaded function "cudaMalloc" matches the argument list
            argument types are: (void **, size_t, int)

1 error detected in the compilation of "/tmp/tmpxft_000010d7_00000000-10_vectorAdd.compute_50.cpp1.ii".
make: *** [src/vectorAdd.o] Error 2
```

Run Application



```
        h_B[i] = rand()/(float)RAND_MAX;
    }

    // Allocate the device input vector A
    float *d_A = NULL;
    err = cudaMalloc((void **)&d_A, size);

    if (err != cudaSuccess)
    {
        fprintf(stderr, "Failed to allocate device vector A (error code %s)!\n", cudaGetErrorString(err));
        exit(EXIT_FAILURE);
    }
```

<terminated> webinar2 [C/C++ Application] /root/cuda-workspace/example/webinar2/Debug/webinar2 (11/29/15 4:47 PM)

```
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

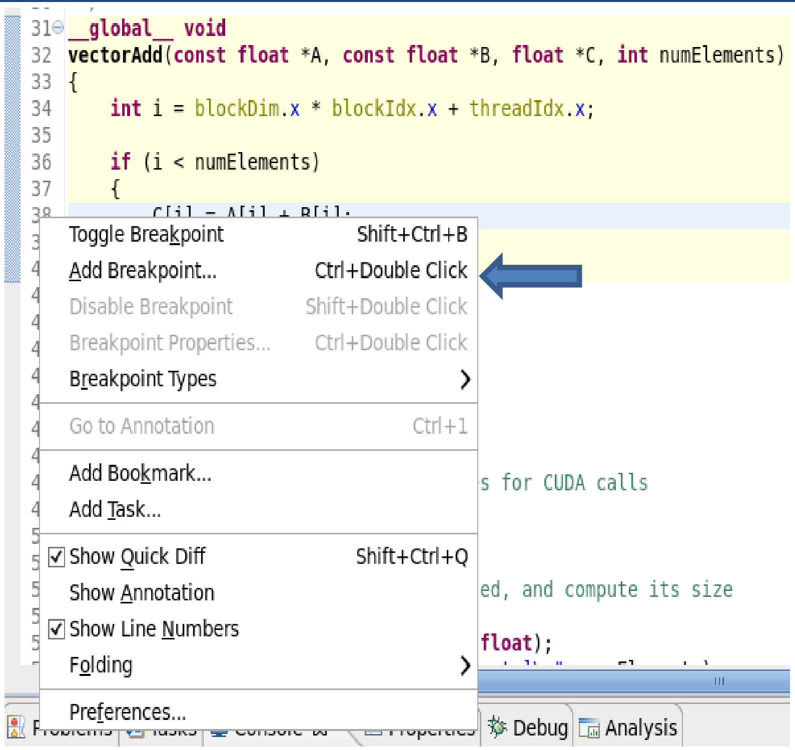
Edit

- CUDA Integrated Development Environment
  - Project Management
  - Edit
  - Build
  - Debug
  - Profile

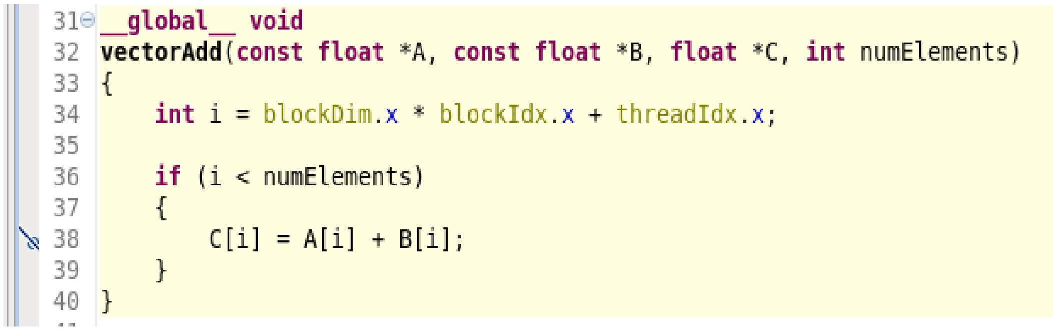
CUDA Debugger

- Unified CPU / GPU Debugging
  - Simultaneous visibility into both CPU and GPU state
  - Multi-GPU support
- Full GPU debugging
  - Set kernel breakpoints
  - Single-step, run until, etc.
  - View variables, registers, and expression values across multiple GPU threads at the same time
  - Examine thread, warp, block state
  - Source and assembly level debugging

Add a break point in the code

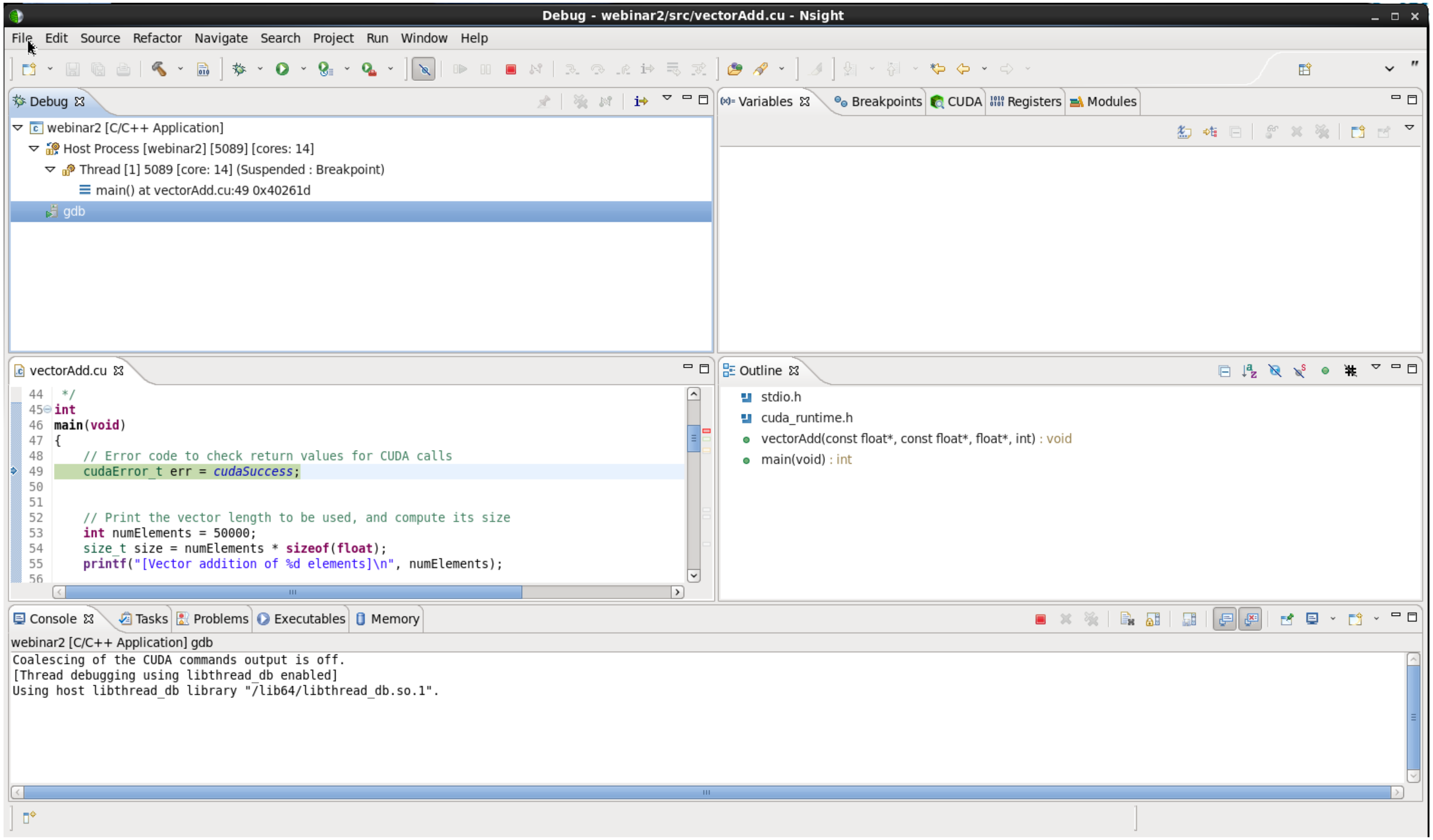


```
__global__ void
vectorAdd(const float *A, const float *B, float *C, int numElements)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements)
    {
        C[i] = A[i] + B[i];
```



```
__global__ void
vectorAdd(const float *A, const float *B, float *C, int numElements)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements)
    {
        C[i] = A[i] + B[i];
    }
}
```

GPU / CPU Threads, Call Stacks


Debug - webinar2/src/vectorAdd.cu - Nsight
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Variables
Breakpoints
CUDA
Registers
Modules
vectorAdd.cu
Outline
Console
Tasks
Problems
Executables
Memory

webinar2 [C/C++ Application]
Host Process [webinar2] [5089] [cores: 14]
Thread [1] 5089 [core: 14] (Suspended : Breakpoint)
main() at vectorAdd.cu:49 0x40261d
gdb

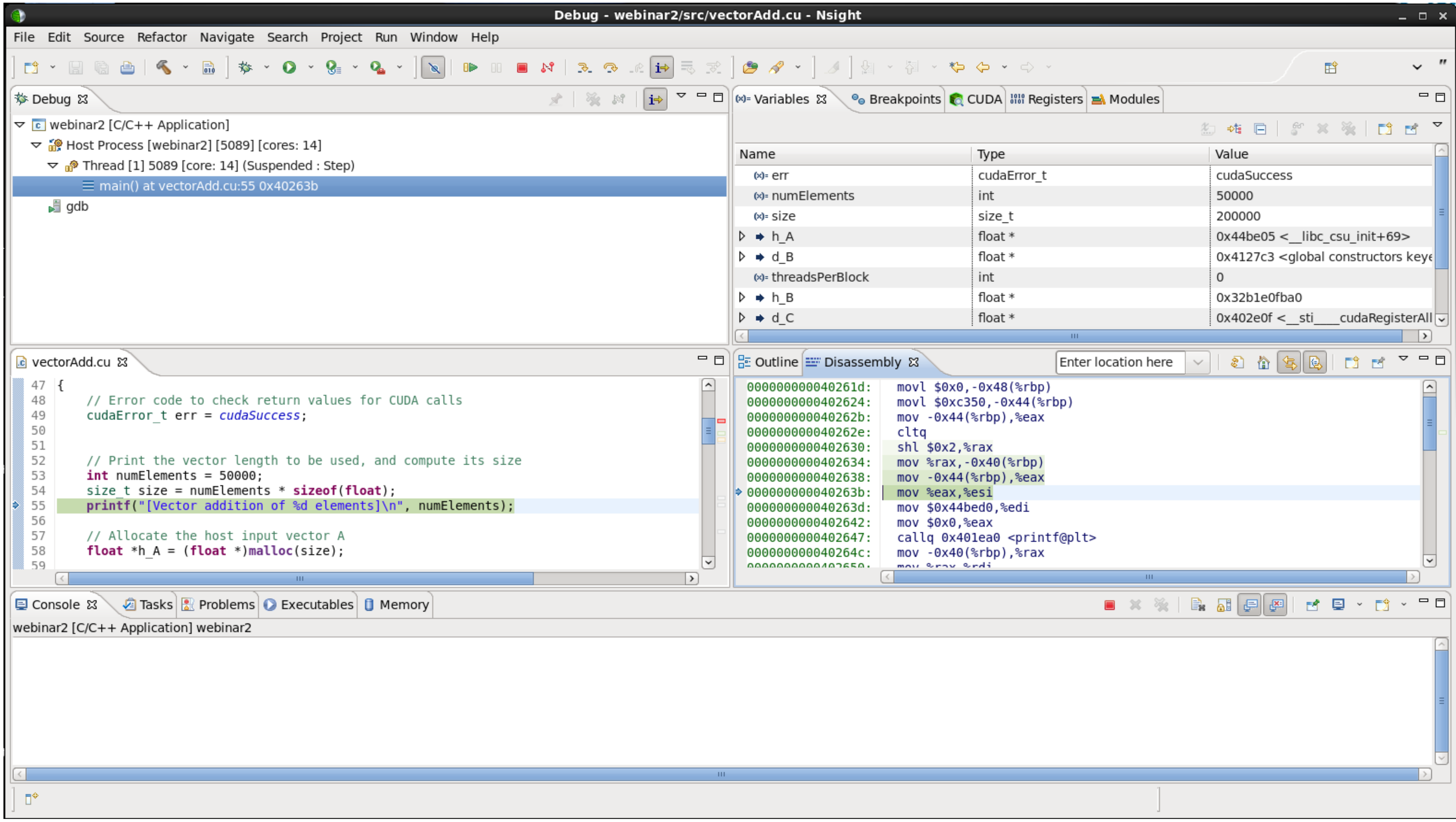
```
 */
int
main(void)
{
    // Error code to check return values for CUDA calls
    cudaError_t err = cudaSuccess;


    // Print the vector length to be used, and compute its size
    int numElements = 50000;
    size_t size = numElements * sizeof(float);
    printf("[Vector addition of %d elements]\n", numElements);
```

stdio.h
cuda_runtime.h
vectorAdd(const float*, const float*, float*, int) : void
main(void) : int

webinar2 [C/C++ Application] gdb

```
Coalescing of the CUDA commands output is off.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
```

Stepping



Debug - webinar2/src/vectorAdd.cu - Nsight

File Edit Source Refactor Navigate Search Project Run Window Help

```
webinar2 [C/C++ Application]
  Host Process [webinar2] [5089] [cores: 14]
    Thread [1] 5089 [core: 14] (Suspended : Step)
      main() at vectorAdd.cu:55 0x40263b
  gdb
```

Name	Type	Value
err	cudaError_t	cudaSuccess
numElements	int	50000
size	size_t	200000
h_A	float *	0x44be05 <__libc_csu_init+69>
d_B	float *	0x4127c3 <global constructors keye
threadsPerBlock	int	0
h_B	float *	0x32b1e0fba0
d_C	float *	0x402e0f <__sti____cudaRegisterAll

vectorAdd.cu

```
{
    // Error code to check return values for CUDA calls
    cudaError_t err = cudaSuccess;


    // Print the vector length to be used, and compute its size
    int numElements = 50000;
    size_t size = numElements * sizeof(float);
    printf("[Vector addition of %d elements]\n", numElements);

    // Allocate the host input vector A
    float *h_A = (float *)malloc(size);
```

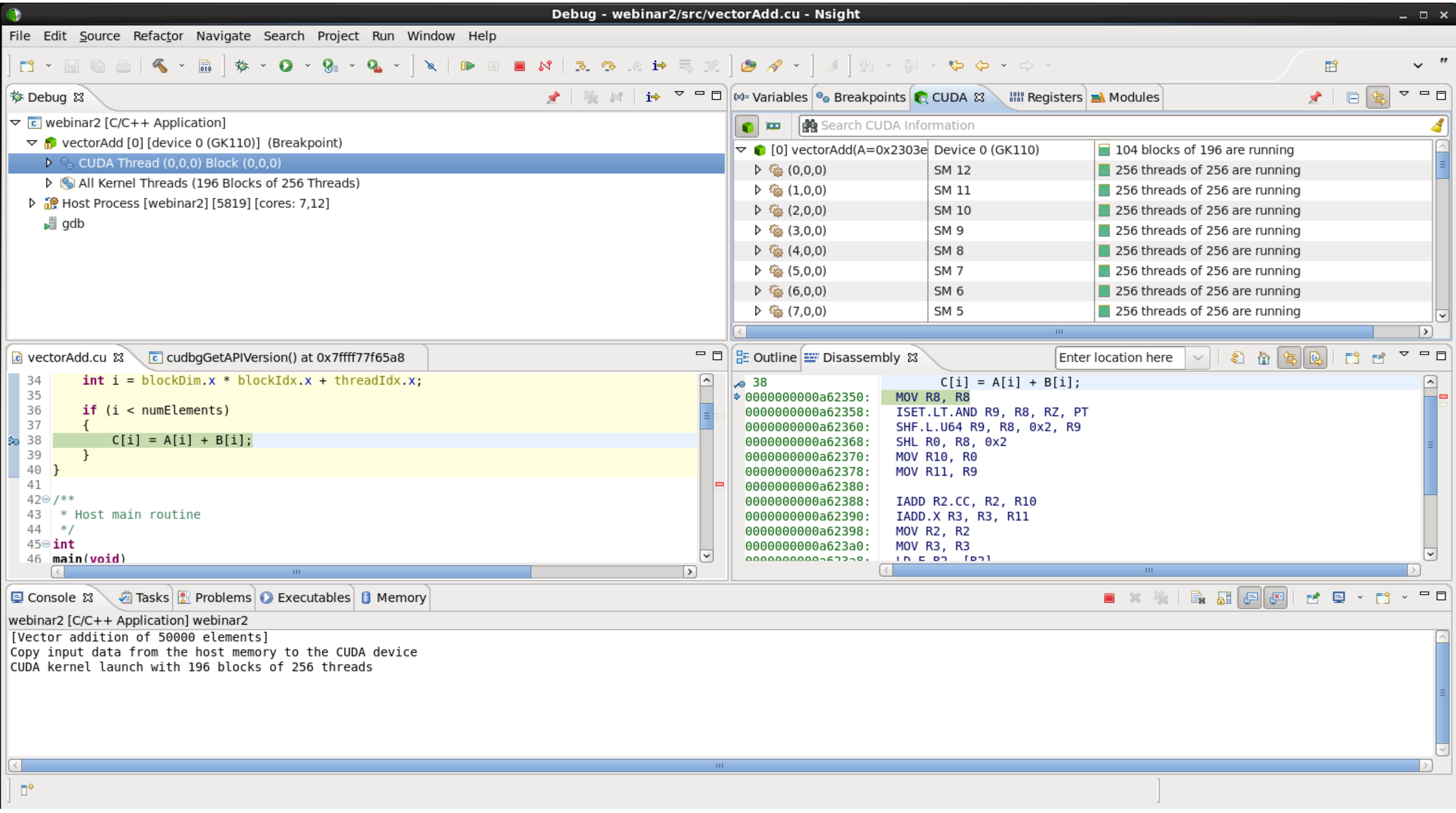
Disassembly

```
000000000040261d:   movl $0x0,-0x48(%rbp)
0000000000402624:   movl $0xc350,-0x44(%rbp)
000000000040262b:   mov -0x44(%rbp),%eax
000000000040262e:   cltq
0000000000402630:   shl $0x2,%rax
0000000000402634:   mov %rax,-0x40(%rbp)
0000000000402638:   mov -0x44(%rbp),%eax
000000000040263b:   mov %eax,%esi
000000000040263d:   mov $0x44bed0,%edi
0000000000402642:   mov $0x0,%eax
0000000000402647:   callq 0x401ea0 <printf@plt>
000000000040264c:   mov -0x40(%rbp),%rax
```

Console

webinar2 [C/C++ Application] webinar2

GPU / CPU Threads, Call Stacks


Debug - webinar2/src/vectorAdd.cu - Nsight
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Variables Breakpoints CUDA Registers Modules
Search CUDA Information
vectorAdd.cu cudbgGetAPIVersion() at 0x7ffff77f65a8
Outline Disassembly
Enter location here
Console Tasks Problems Executables Memory

```
webinar2 [C/C++ Application]
  vectorAdd [0] [device 0 (GK110)]  (Breakpoint)
    CUDA Thread (0,0,0) Block (0,0,0)
    All Kernel Threads (196 Blocks of 256 Threads)
  Host Process [webinar2] [5819] [cores: 7,12]
  gdb
```

[0] vectorAdd(A=0x2303e	Device 0 (GK110)	🟩 104 blocks of 196 are running
(0,0,0)	SM 12	🟩 256 threads of 256 are running
(1,0,0)	SM 11	🟩 256 threads of 256 are running
(2,0,0)	SM 10	🟩 256 threads of 256 are running
(3,0,0)	SM 9	🟩 256 threads of 256 are running
(4,0,0)	SM 8	🟩 256 threads of 256 are running
(5,0,0)	SM 7	🟩 256 threads of 256 are running
(6,0,0)	SM 6	🟩 256 threads of 256 are running
(7,0,0)	SM 5	🟩 256 threads of 256 are running

```
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements)
    {
        C[i] = A[i] + B[i];
    }
}

/**
 * Host main routine
 */
int
main(void)
```

```
38                  C[i] = A[i] + B[i];
0000000000a62350:  MOV R8, R8
0000000000a62358:  ISET.LT.AND R9, R8, RZ, PT
0000000000a62360:  SHF.L.U64 R9, R8, 0x2, R9
0000000000a62368:  SHL R0, R8, 0x2
0000000000a62370:  MOV R10, R0
0000000000a62378:  MOV R11, R9
0000000000a62380:
0000000000a62388:  IADD R2.CC, R2, R10
0000000000a62390:  IADD.X R3, R3, R11
0000000000a62398:  MOV R2, R2
0000000000a623a0:  MOV R3, R3
```

```
webinar2 [C/C++ Application] webinar2
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
```

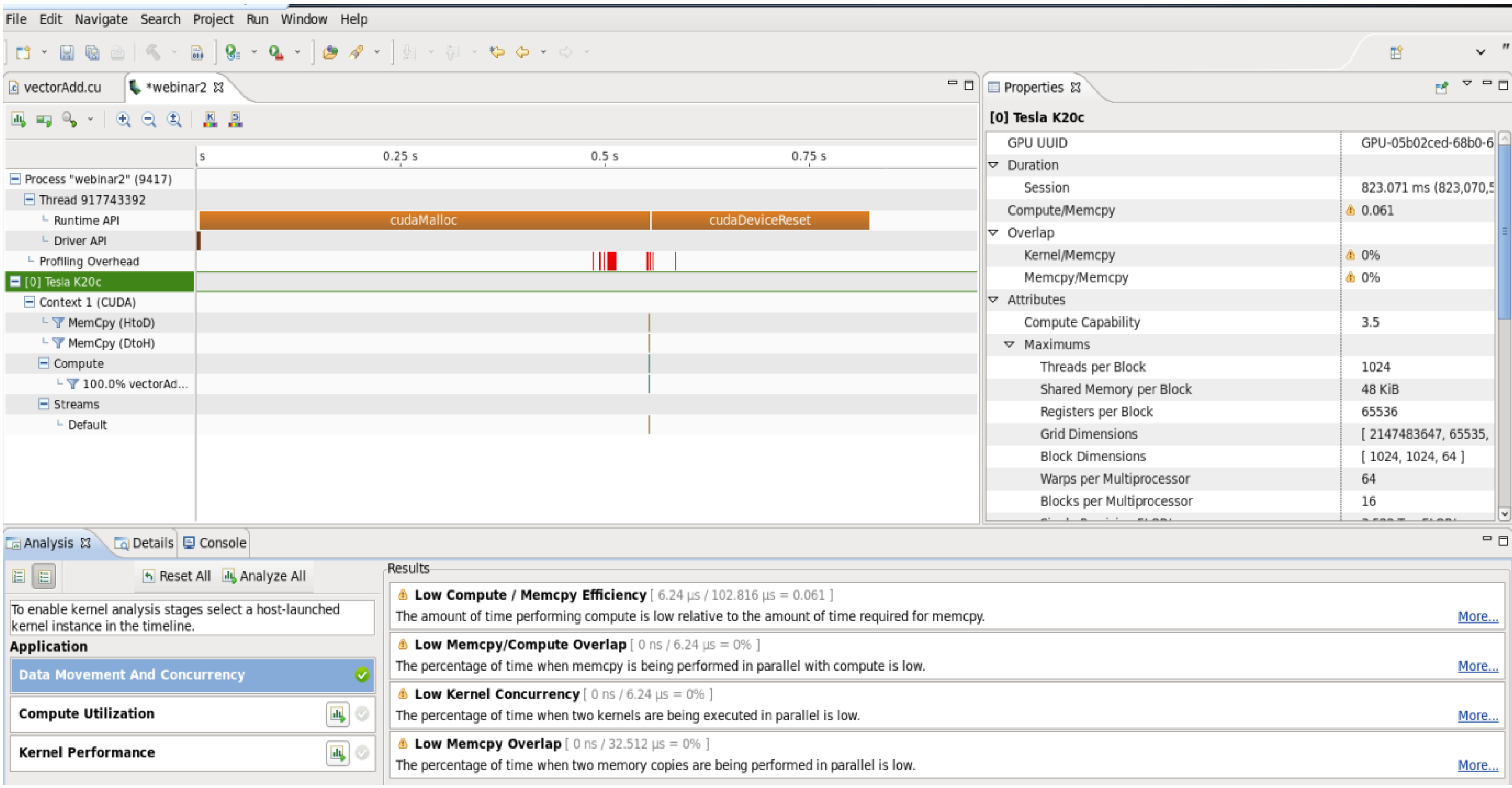
Edit

- CUDA Integrated Development Environment
  - Project Management
  - Edit
  - Build
  - Debug
  - Profile

CUDA Profiler

- Unified GPU / CPU profiler
- Visualize GPU / CPU interactions Identify GPU utilization and efficiency bottlenecks
- View low-level counters and metrics
- Multi-GPU support
- Automated application analysis identifies optimization opportunities
- Online documentation gives direction on how to exploit opportunities to get performance improvement

Unified GPU / CPU Timeline



[0] Tesla K20c

GPU UUID	GPU-05b02ced-68b0-6
Duration	
Session	823.071 ms (823,070,5
Compute/Memcpy	0.061
Overlap	
Kernel/Memcpy	0%
Memcpy/Memcpy	0%
Attributes	
Compute Capability	3.5
Maximums	
Threads per Block	1024
Shared Memory per Block	48 KiB
Registers per Block	65536
Grid Dimensions	[2147483647, 65535,
Block Dimensions	[1024, 1024, 64]
Warps per Multiprocessor	64
Blocks per Multiprocessor	16

To enable kernel analysis stages select a host-launched kernel instance in the timeline.

Application

- Data Movement And Concurrency
- Compute Utilization
- Kernel Performance

Results

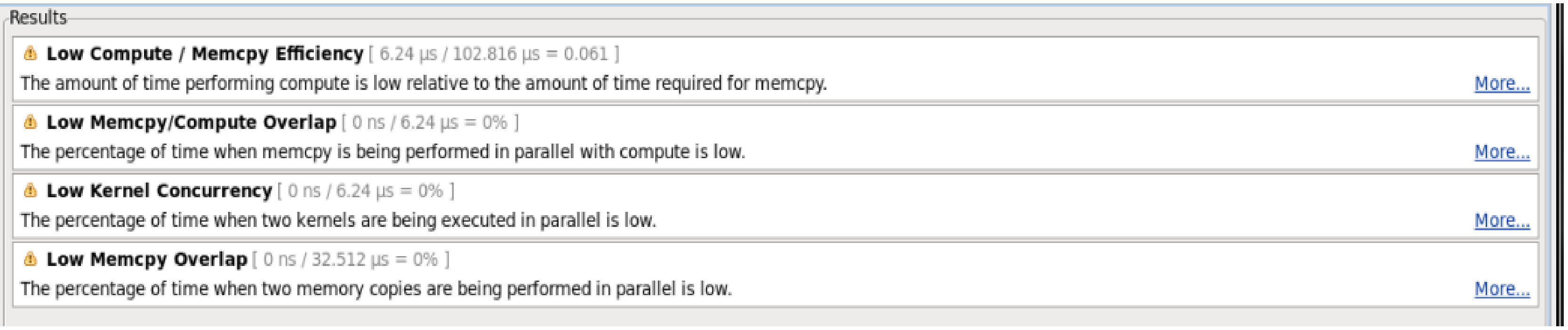
Low Compute / Memcpy Efficiency [6.24 µs / 102.816 µs = 0.061]
The amount of time performing compute is low relative to the amount of time required for memcpy. More...

Low Memcpy/Compute Overlap [0 ns / 6.24 µs = 0%]
The percentage of time when memcpy is being performed in parallel with compute is low. More...

Low Kernel Concurrency [0 ns / 6.24 µs = 0%]
The percentage of time when two kernels are being executed in parallel is low. More...

Low Memcpy Overlap [0 ns / 32.512 µs = 0%]
The percentage of time when two memory copies are being performed in parallel is low. More...

Analysis Documentation



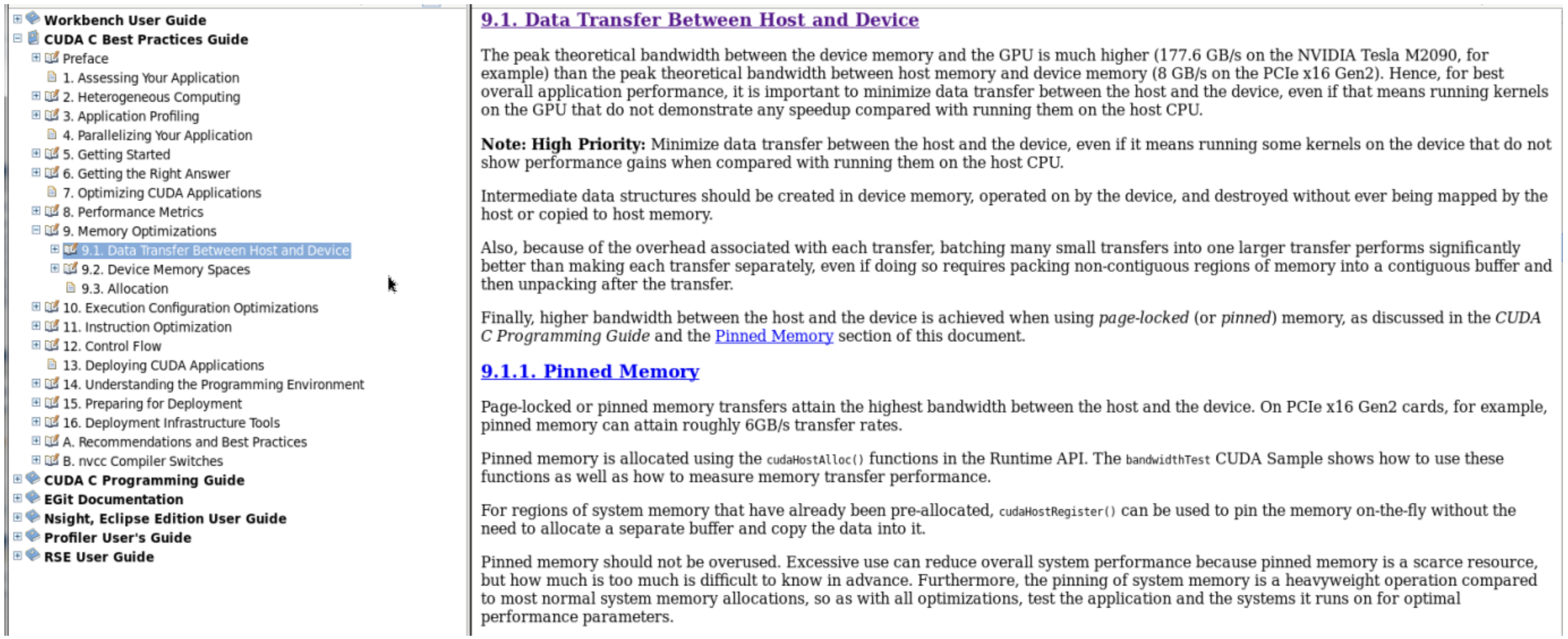
Results

Low Compute / Memcpy Efficiency [6.24 µs / 102.816 µs = 0.061]
The amount of time performing compute is low relative to the amount of time required for memcpy. More...

Low Memcpy/Compute Overlap [0 ns / 6.24 µs = 0%]
The percentage of time when memcpy is being performed in parallel with compute is low. More...

Low Kernel Concurrency [0 ns / 6.24 µs = 0%]
The percentage of time when two kernels are being executed in parallel is low. More...

Low Memcpy Overlap [0 ns / 32.512 µs = 0%]
The percentage of time when two memory copies are being performed in parallel is low. More...



Workbench User Guide
CUDA C Best Practices Guide
- Preface
- 1. Assessing Your Application
- 2. Heterogeneous Computing
- 3. Application Profiling
- 4. Parallelizing Your Application
- 5. Getting Started
- 6. Getting the Right Answer
- 7. Optimizing CUDA Applications
- 8. Performance Metrics
- 9. Memory Optimizations
  - 9.1. Data Transfer Between Host and Device
  - 9.2. Device Memory Spaces
  - 9.3. Allocation
- 10. Execution Configuration Optimizations
- 11. Instruction Optimization
- 12. Control Flow
- 13. Deploying CUDA Applications
- 14. Understanding the Programming Environment
- 15. Preparing for Deployment
- 16. Deployment Infrastructure Tools
- A. Recommendations and Best Practices
- B. nvcc Compiler Switches

CUDA C Programming Guide
EGit Documentation
Nsight, Eclipse Edition User Guide
Profiler User's Guide
RSE User Guide

9.1. Data Transfer Between Host and Device

The peak theoretical bandwidth between the device memory and the GPU is much higher (177.6 GB/s on the NVIDIA Tesla M2090, for example) than the peak theoretical bandwidth between host memory and device memory (8 GB/s on the PCIe x16 Gen2). Hence, for best overall application performance, it is important to minimize data transfer between the host and the device, even if that means running kernels on the GPU that do not demonstrate any speedup compared with running them on the host CPU.

Note: High Priority: Minimize data transfer between the host and the device, even if it means running some kernels on the device that do not show performance gains when compared with running them on the host CPU.

Intermediate data structures should be created in device memory, operated on by the device, and destroyed without ever being mapped by the host or copied to host memory.

Also, because of the overhead associated with each transfer, batching many small transfers into one larger transfer performs significantly better than making each transfer separately, even if doing so requires packing non-contiguous regions of memory into a contiguous buffer and then unpacking after the transfer.

Finally, higher bandwidth between the host and the device is achieved when using *page-locked* (or *pinned*) memory, as discussed in the *CUDA C Programming Guide* and the Pinned Memory section of this document.

9.1.1. Pinned Memory

Page-locked or pinned memory transfers attain the highest bandwidth between the host and the device. On PCIe x16 Gen2 cards, for example, pinned memory can attain roughly 6GB/s transfer rates.

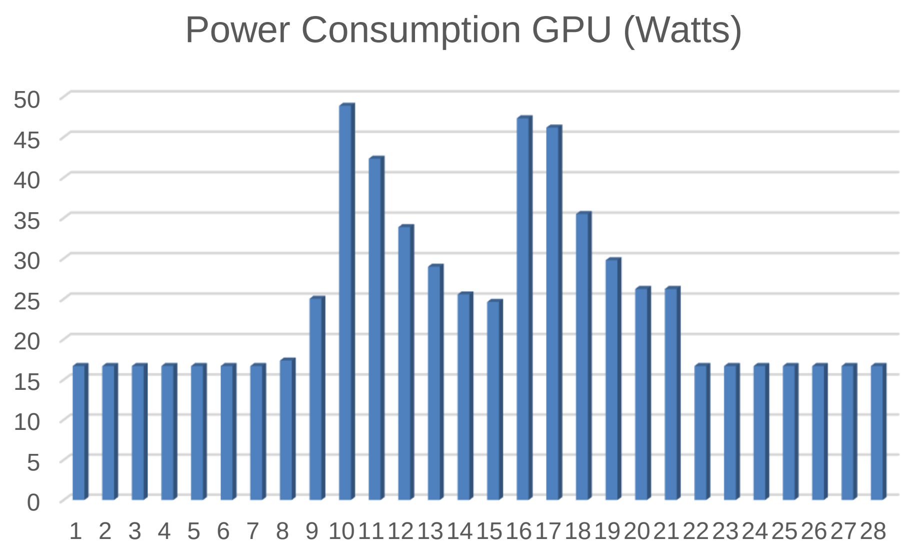
Pinned memory is allocated using the `cudaHostAlloc()` functions in the Runtime API. The `bandwidthTest` CUDA Sample shows how to use these functions as well as how to measure memory transfer performance.

For regions of system memory that have already been pre-allocated, `cudaHostRegister()` can be used to pin the memory on-the-fly without the need to allocate a separate buffer and copy the data into it.

Pinned memory should not be overused. Excessive use can reduce overall system performance because pinned memory is a scarce resource, but how much is too much is difficult to know in advance. Furthermore, the pinning of system memory is a heavyweight operation compared to most normal system memory allocations, so as with all optimizations, test the application and the systems it runs on for optimal performance parameters.

Power profiling using nvidia-smi

- **nvidia-smi -q -d POWER -i 0 -l 1 -f out.log**
  - **−q, −−query** Display GPU or Unit info
  - **−d TYPE, −−display=TYPE** Display only selected information: MEMORY, UTILIZATION, ECC, TEMPERATURE, POWER, CLOCK, COMPUTE…
  - **−i, −−id=ID** Display data for a single specified GPU or Unit.
  - **−f FILE, −−filename=FILE** Redirect query output to the specified file in place of the default stdout. The specified file will be overwritten.



Live Demo

HPP: FPGA

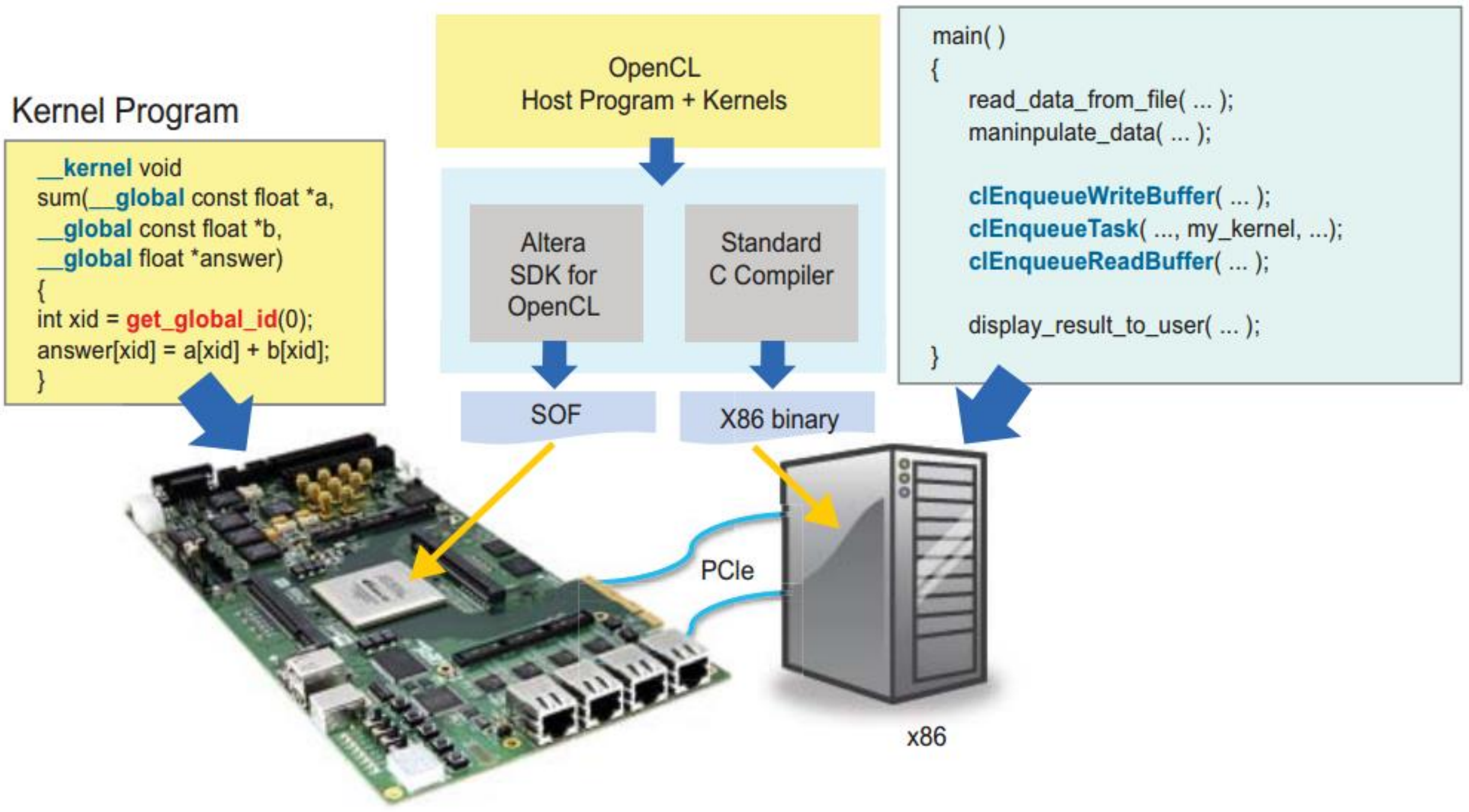
FPGA pains

- Programmability is an issue
  - Hardware Description Languages (HDLs) are complexe
  - At Register transfer level (RTL) abstraction
  - Designer needs to :
    - Create Custom Memory Hierarchies
    - Manage Communication with PC
    - Create PC side SW that plays nice with all this

OpenCL for FPGA

- Unified programming model
  - – More accessible to the software developpers
  - – Host CPU-FPGA communication
  - – C based programming language
  - – Memory Hierarchy auto generated
  - – Potential to integrate existing VHDL/Verilog IP

OpenCL for FPGA



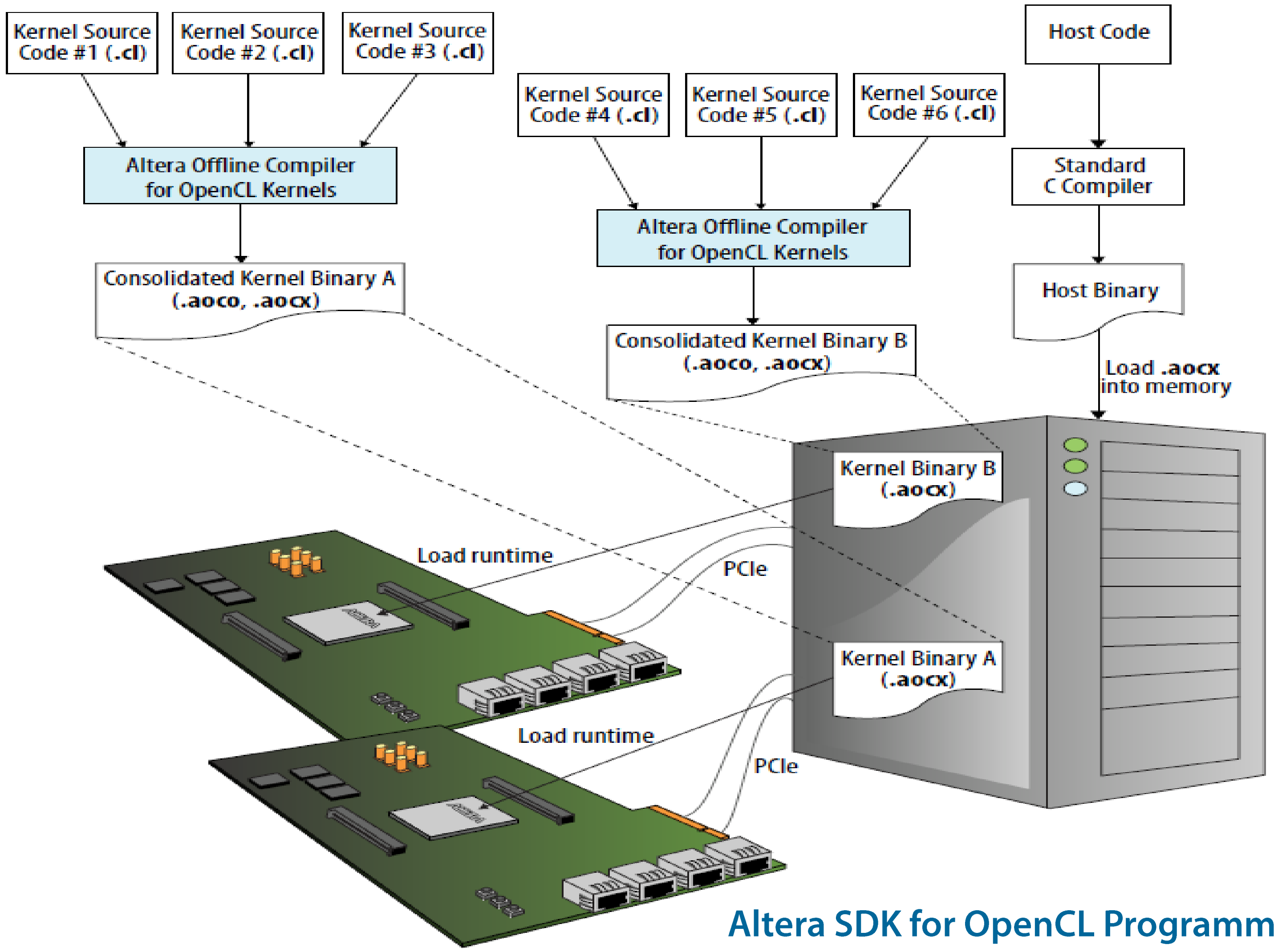
Kernel Program

```
__kernel void
sum(__global const float *a,
__global const float *b,
__global float *answer)
{
int xid = get_global_id(0);
answer[xid] = a[xid] + b[xid];
}
```

```
main( )
{
    read_data_from_file( ... );
    maninpulate_data( ... );

    clEnqueueWriteBuffer( ... );
    clEnqueueTask( ..., my_kernel, ...);
    clEnqueueReadBuffer( ... );

    display_result_to_user( ... );
}
```

White Paper: Implementing FPGA Design with the OpenCL Standard (Figure 4)

The AOCL FPGA Programming Flow

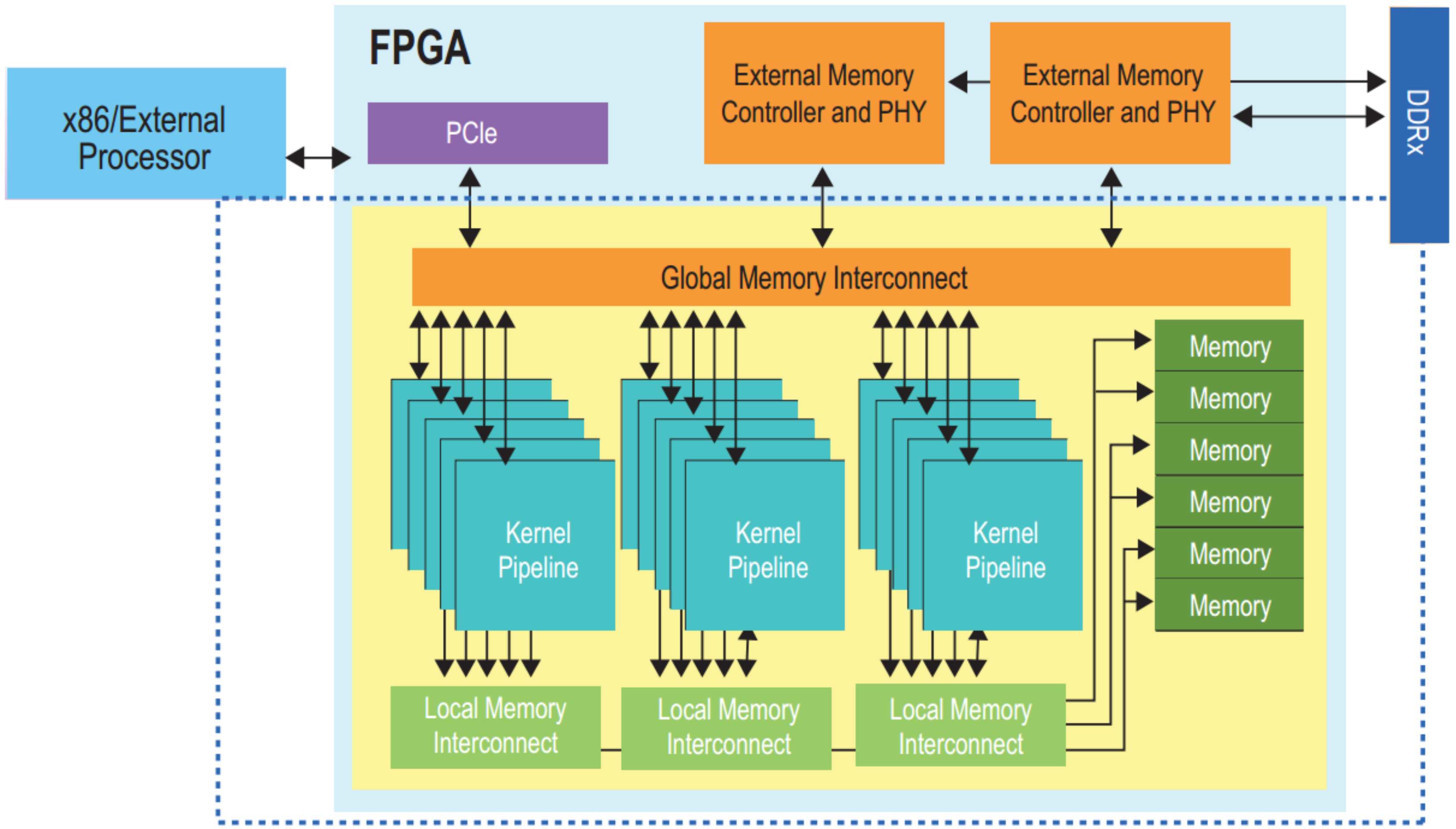

Kernel Source Code #1 (.cl)
Kernel Source Code #2 (.cl)
Kernel Source Code #3 (.cl)
Altera Offline Compiler for OpenCL Kernels
Consolidated Kernel Binary A (.aoco, .aocx)
Kernel Source Code #4 (.cl)
Kernel Source Code #5 (.cl)
Kernel Source Code #6 (.cl)
Altera Offline Compiler for OpenCL Kernels
Consolidated Kernel Binary B (.aoco, .aocx)
Host Code
Standard C Compiler
Host Binary
Load .aocx into memory
Kernel Binary B (.aocx)
Kernel Binary A (.aocx)
Load runtime
PCIe
Load runtime
PCIe

Altera SDK for OpenCL Programming Guide

AOCL design flow steps

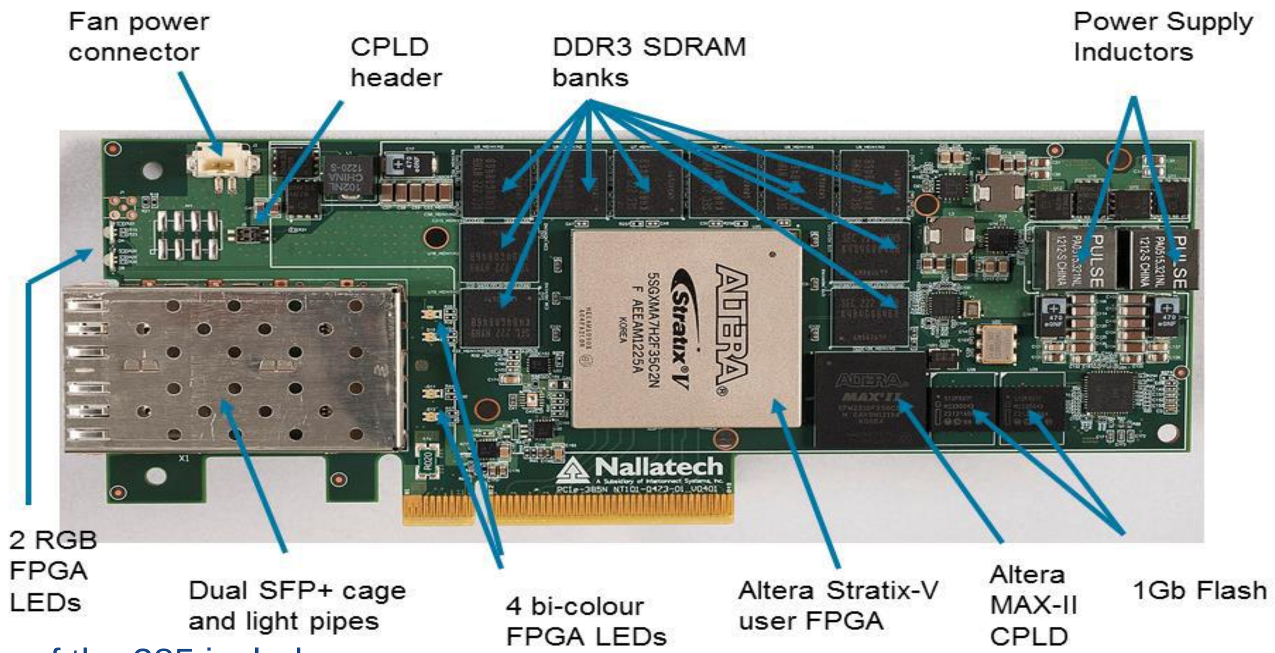
1. **Intermediate compilation (aoc –c [-g] <*your_kernel_filename*>.cl)**
   - Checks for syntatic errors
   - Generates a **.aoco** file without building the hardware configuration file
   - Generate estimated resource usage summary **<*your_kernel_filename*>.log**
2. **Emulation (aoc -g <your_kernel_filename>.cl)**
   - The AOCL Emulator generates a **.aocx** file that executes on x86-64 Windows or Linux host
   - Assess the functionality of your OpenCL kernel
3. **Profiling (aoc --profile <your_kernel_filename>.cl)**
   - aocl report <*your_kernel_filename*>.aocx profile.mon
   - Instruct the Altera Offline Compiler to instrument performance counters in the Verilog code in the **.aocx** file
   - During execution, the performance counters collect performance information which you can then review in the Profiler GUI.
4. **Full deployment**
   - Execute the .aocx file on the FPGA

OpenCL Overview


x86/External Processor
FPGA
PCIe
External Memory Controller and PHY
External Memory Controller and PHY
DDRx
Global Memory Interconnect
Kernel Pipeline
Kernel Pipeline
Kernel Pipeline
Memory
Memory
Memory
Memory
Memory
Memory
Local Memory Interconnect
Local Memory Interconnect
Local Memory Interconnect

White Paper: Implementing FPGA Design with the OpenCL Standard (Page: 7)

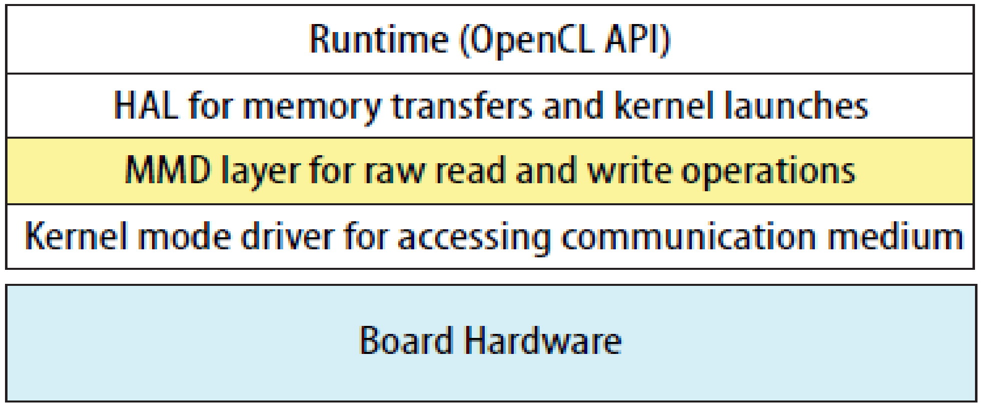
Nallatech 385 Hardware Overview



- The key features of the 385 include:
    - PCI Express form factor
        - 8-lane PCI Express up to 3.0 host interface1
    - FPGA options
        - Altera Stratix-V 5SGXMA7H2F35C2N
        - Altera Stratix-V 5SGSMD5H2F35C2N
    - Two 10G LAN/WAN/FC Ethernet channels accessed via two SFP+ ports2
    - Two banks of SDRAM memory, each bank with 4GByte, x72, DDR3 SDRAM running at 1600 MT/s

BSP and Altera SDK for OpenCL

- Four layers of the Altera Software Development Kit (SDK) for OpenCL :


Runtime (OpenCL API)
HAL for memory transfers and kernel launches
MMD layer for raw read and write operations
Kernel mode driver for accessing communication medium
Board Hardware

- The Nallatech OpenCL BSPs provide a memory-mapped device (MMD) layer necessary for communication with the accelerator board and the lower level kernel mode driver.
- All other upper software layers are provided by the Altera SDK for OpenCL installation.

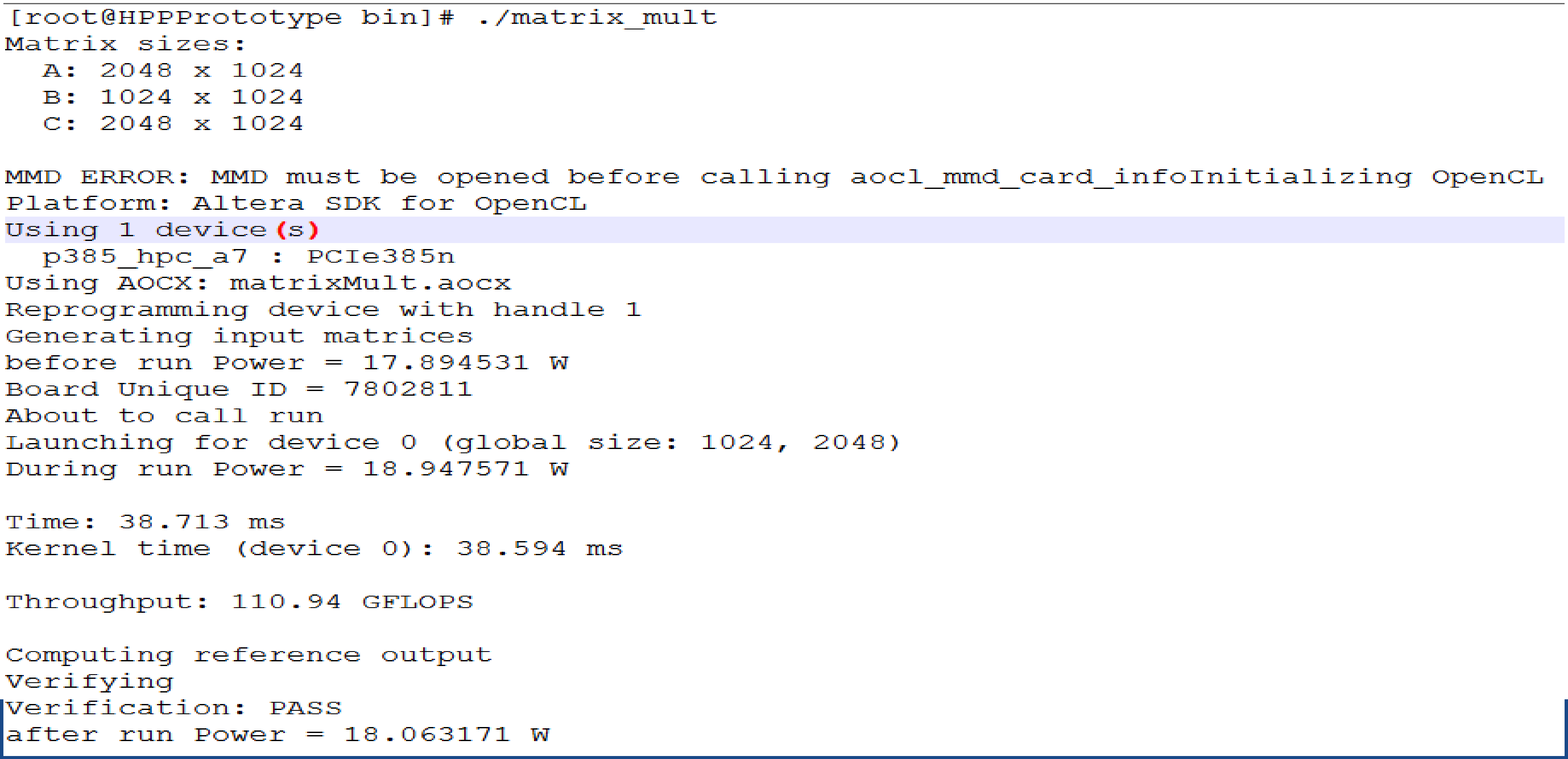
Linux Vector Addition Walkthrough

- Step 1 Compile
  - aoc –v --board p395_hpc_ab vectorAdd.cl
- Step 2 Build the Host code
  - make
- Step 3 Execute
  - ./vector_add

- Generated files:
  - vectorAdd.log – kernel compilation with estimated resource usage and Qsys generation.
  - quartus_sh_compile.log – Quartus tools build log for generating the actual FPGA hardware aocx file.
  - acl_quartus_report.txt – Final results for the generated design including final resource usage figures.

Power consumption reading

- Note that this requires that you have already opened the card using the normal opencl SDK routines (i.e. in init_opencl in this case).
- **aocl_mmd_card_info**("acl0", AOCL_MMD_POWER, sizeof(float),(void*) &power, &returnedSize);
- **printf**("before run Power = %f W\n", power);
- **aocl_mmd_card_info**("acl0", AOCL_MMD_BOARD_UNIQUE_ID, sizeof(int), (void*) &uniqueId, &returnedSize);
- **printf**("Board Unique ID = %d\n", uniqueId);



```
[root@HPPPrototype bin]# ./matrix_mult
Matrix sizes:
  A: 2048 x 1024
  B: 1024 x 1024
  C: 2048 x 1024

MMD ERROR: MMD must be opened before calling aocl_mmd_card_infoInitializing OpenCL
Platform: Altera SDK for OpenCL
Using 1 device(s)
  p385_hpc_a7 : PCIe385n
Using AOCX: matrixMult.aocx
Reprogramming device with handle 1
Generating input matrices
before run Power = 17.894531 W
Board Unique ID = 7802811
About to call run
Launching for device 0 (global size: 1024, 2048)
During run Power = 18.947571 W

Time: 38.713 ms
Kernel time (device 0): 38.594 ms

Throughput: 110.94 GFLOPS

Computing reference output
Verifying
Verification: PASS
after run Power = 18.063171 W
```

CMC MICROSYSTEMS

Live Demo

Project status

Status (12 universities selected 18 systems G1, 7 universities selected 8 G2)

- Assembled, cloned, tested and shipped 18/18 units
- (1) Quick start guide : Heterogeneous Parallel Platform (HPP)
- Introduction to the HPP-Heterogeneous Parallel Platform: A combination of Multicores, GPUs, FPGAs and Many-cores accelerators (August 26th)

In progress

- (2) User Guide: Performance and Power profiling for the HPP
- Programming models, performance and power profiling for the HPP-Heterogeneous Parallel Platform (December 2nd)

Next (Webinars series for the HPP)

  - Computer vision using OpenCV/OpenCL targeting the HPP- Heterogeneous Processing Platform (January 13th)

Workshop on Heterogeneous Computing Platforms (Scope)

Location: Toronto

Date: TBD (February 15 - February 19, 2016)

Objective

- Bring researchers from academia and expert from industry to discuss about heterogeneous computing and explore collaboration opportunities

Technical session: presentations

- Applications and Algorithms
- Software stack and tools
- Heterogeneous systems Architecture

Breakout session: Discuss and report

- What are the next generation platforms specifications?
  - What is the ranked list of platform features that would enable this research?
- What are the key research problems that needs acceleration?
- What are the top three barriers preventing access and effective use of these platforms?
- What are the challenges and Opportunities
- What barriers could CMC help lower ?
- What would enable and encourage a vibrant community of researchers sharing platform configuration files and other knowledge?
- Any other feedback and guidance

Q&A

Yassine Hariri
Hariri@cmc.ca