

Approaches to GPU computing

Manuel Ujaldon

Nvidia CUDA Fellow

Computer Architecture Department
University of Malaga (Spain)

Talk outline [40 slides]

1. Programming choices. [30]
 1. CUDA libraries and tools. [10]
 2. Targeting CUDA to other platforms. [5]
 3. Accessing CUDA from other languages. [4]
 4. Using directives: OpenACC. [11]
2. Examples: Six ways to implement SAXPY on GPUs. [9]
3. Summary. [1]

I. Programming choices

CUDA Parallel Computing Platform

Programming Approaches

Libraries

“Drop-in”
Acceleration

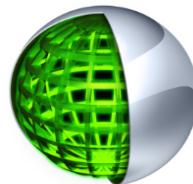
OpenACC
Directives

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

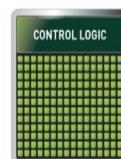
Open Compiler Tool Chain



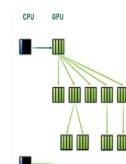
Enables compiling new languages to CUDA
platform, and CUDA languages to other
architectures

Hardware Capabilities

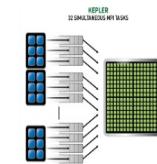
SMX



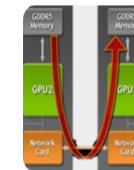
Dynamic
Parallelism



HyperQ



GPUDirect



I. 1. CUDA Libraries and tools



Libraries: Easy, high-quality acceleration

- ➊ **Ease of use:** Using libraries enables GPU acceleration without in-depth knowledge of GPU programming.
- ➋ **"Drop-in":** Many GPU-accelerated libraries follow standard APIs, thus enabling accel. with minimal changes.
- ➌ **Quality:** Libraries offer high-quality implementations of functions encountered in a broad range of applications.
- ➍ **Performance:** Nvidia libraries are tuned by experts.

Three steps to CUDA-accelerated applications

- Step 1: Substitute library calls with equivalent CUDA library calls.

 - `saxpy(...)` --> `cublasSaxpy(...)`

- Step 2: Manage data locality.

 - With CUDA: `cudaMalloc()`, `cudaMemcpy()`, etc.
 - With CUBLAS: `cublasAlloc()`, `cublasSetVector()`, etc.

- Step 3: Rebuild and link the CUDA-accelerated library.

 - `nvcc myobj.o -l cublas`

A linear algebra example

```
int N = 1 << 20;  
  
// Perform SAXPY on 1M elements: y[] = a*x[] + y[]  
saxpy(N, 2.0, x, y, 1);
```

A linear algebra example

```
int N = 1 << 20;
```

```
// Perform SAXPY on 1M elements: d_y[] = a * d_x[] + d_y[]
cublasSaxpy(N, 2.0, d_x, d_y, 1);
```

Add "cublas" prefix and
use device variables

A linear algebra example

```
int N = 1 << 20;  
cUBLASInit();
```

Initialize CUBLAS



```
// Perform SAXPY on 1M elements: d_y[] = a * d_x[] + d_y[]  
cUBLASSaxpy(N, 2.0, d_x, d_y, 1);
```

```
cUBLASShutdown();
```

Shut down CUBLAS



A linear algebra example

```
int N = 1 << 20;  
cublasInit();  
cublasAlloc(N, sizeof(float), (void**)&d_x);  
cublasAlloc(N, sizeof(float), (void**)&d_y);
```

Allocate device vectors

```
// Perform SAXPY on 1M elements: d_y[] = a * d_x[] + d_y[]  
cublasSaxpy(N, 2.0, d_x, d_y, 1);
```

```
cublasFree(d_x);  
cublasFree(x_y);
```

Deallocate device vectors

```
cublasShutdown();
```

A linear algebra example

```
int N = 1 << 20;  
cublasInit();  
cublasAlloc(N, sizeof(float), (void**)&d_x);  
cublasAlloc(N, sizeof(float), (void**)&d_y);  
  
cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);  
cublasSetVector(N, sizeof(x[0]), y, 1, d_y, 1);  
  
// Perform SAXPY on 1M elements: d_y[] = a * d_x[] + d_y[]  
cublasSaxpy(N, 2.0, d_x, d_y, 1);  
  
cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);  
  
cublasFree(d_x);  
cublasFree(x_y);  
  
cublasShutdown();
```

Transfer data to GPU

Read data back GPU

CUDA Math Libraries

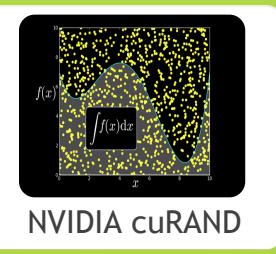
- ➊ High performance math routines for your applications:
 - ➊ **cuFFT:** Fast Fourier Transforms Library.
 - ➊ **cuBLAS:** Complete BLAS (Basic Linear Algebra Subroutines) Library.
 - ➊ **cuSPARSE:** Sparse Matrix Library.
 - ➊ **cuRAND:** RNG (Random Number Generation) Library.
 - ➊ **NPP:** Performance Primitives for Image & Video Processing.
 - ➊ **Thrust:** Templated Parallel Algorithms & Data Structures.
 - ➊ **math.h:** C99 floating-point library.
- ➊ All included in the CUDA Toolkit. Free download at:
 - ➊ <https://developer.nvidia.com/cuda-downloads>

GPU accelerated libraries

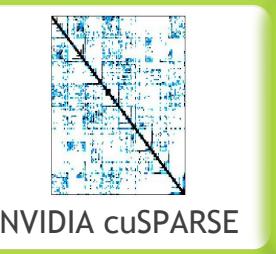
- Many other libraries outside the CUDA Toolkit...



NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE



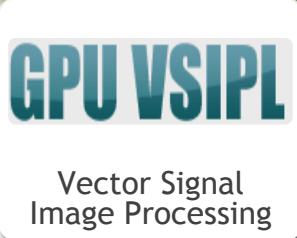
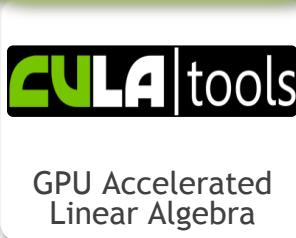
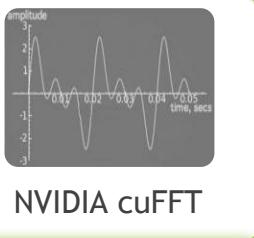
NVIDIA NPP



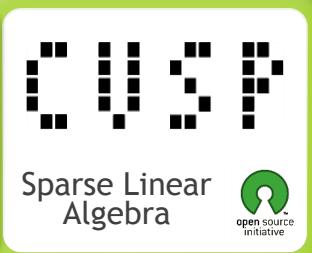
Developed by Nvidia.



Open source libraries.

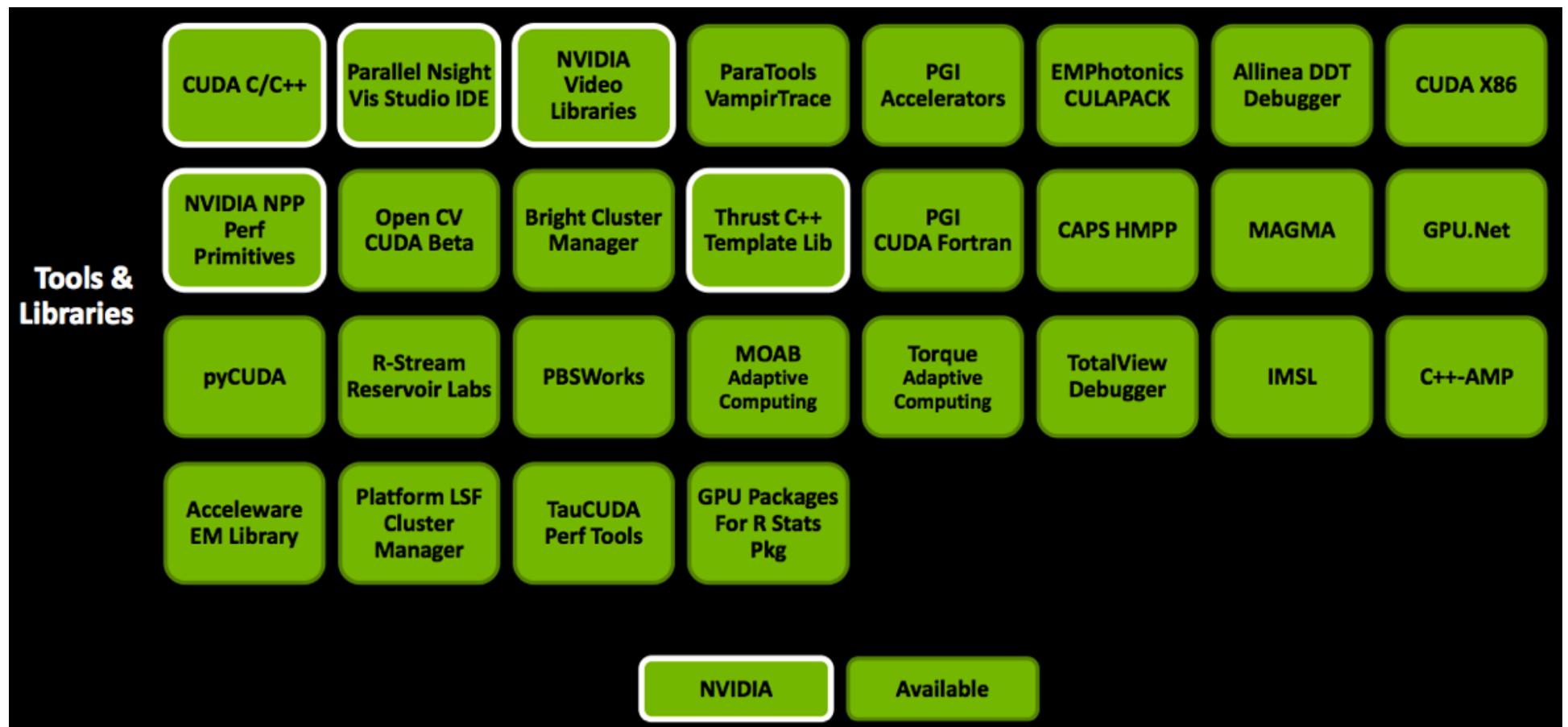
Vector Signal
Image ProcessingGPU Accelerated
Linear AlgebraMatrix Algebra
on GPU and
Multicore

NVIDIA cuFFT

ArrayFire Matrix
ComputationsSparse Linear
AlgebraC++ STL
Features for
CUDA

- ... not to mention all programs that are available on the Web thanks to the generosity of tough programmers.

Tools and Libraries: Developer ecosystem enables the application growth

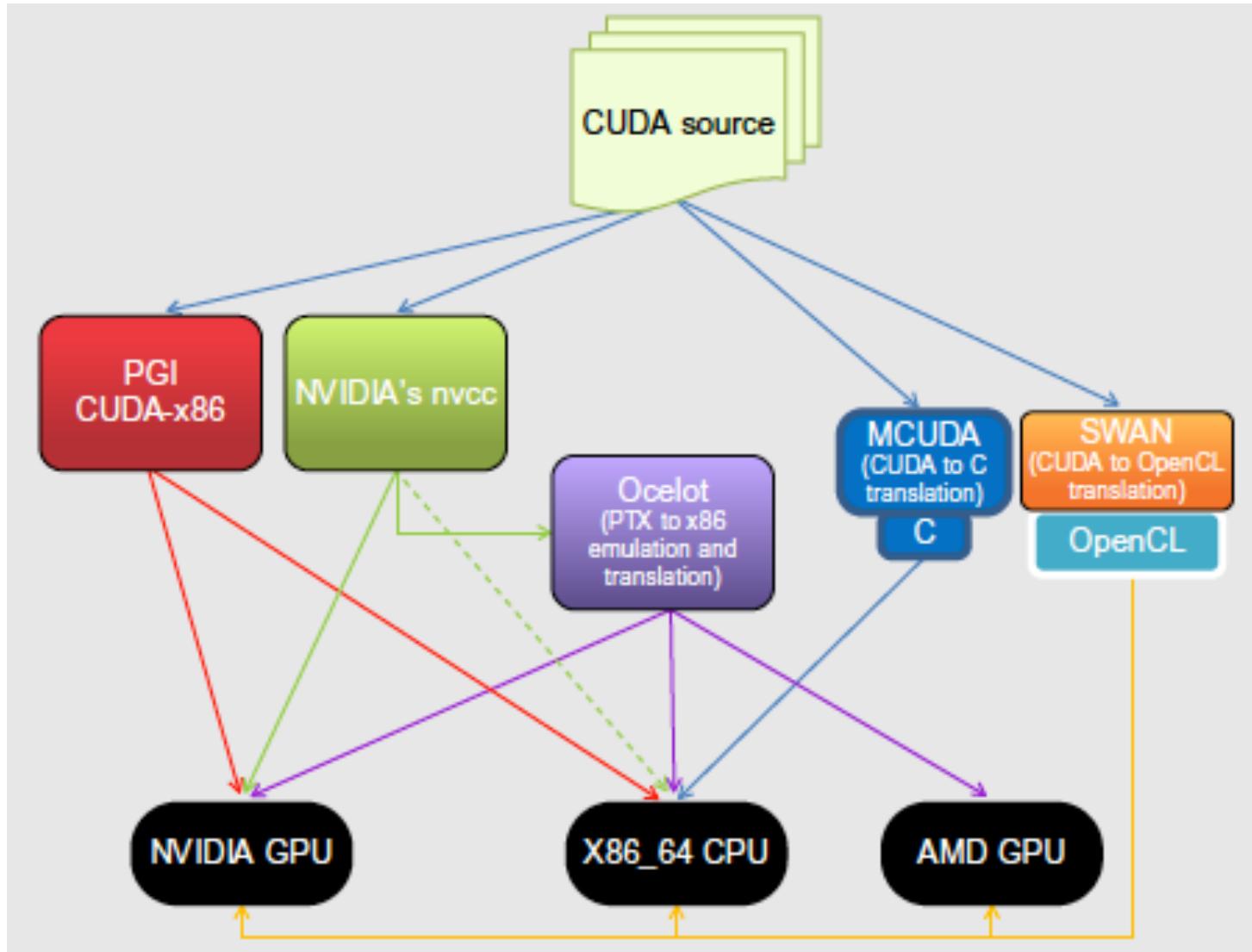


- Described in detail on Nvidia Developer Zone:
 - <http://developer.nvidia.com/cuda-tools-ecosystem>

I. 2. Targeting CUDA to other platforms



Compiling for other target platforms

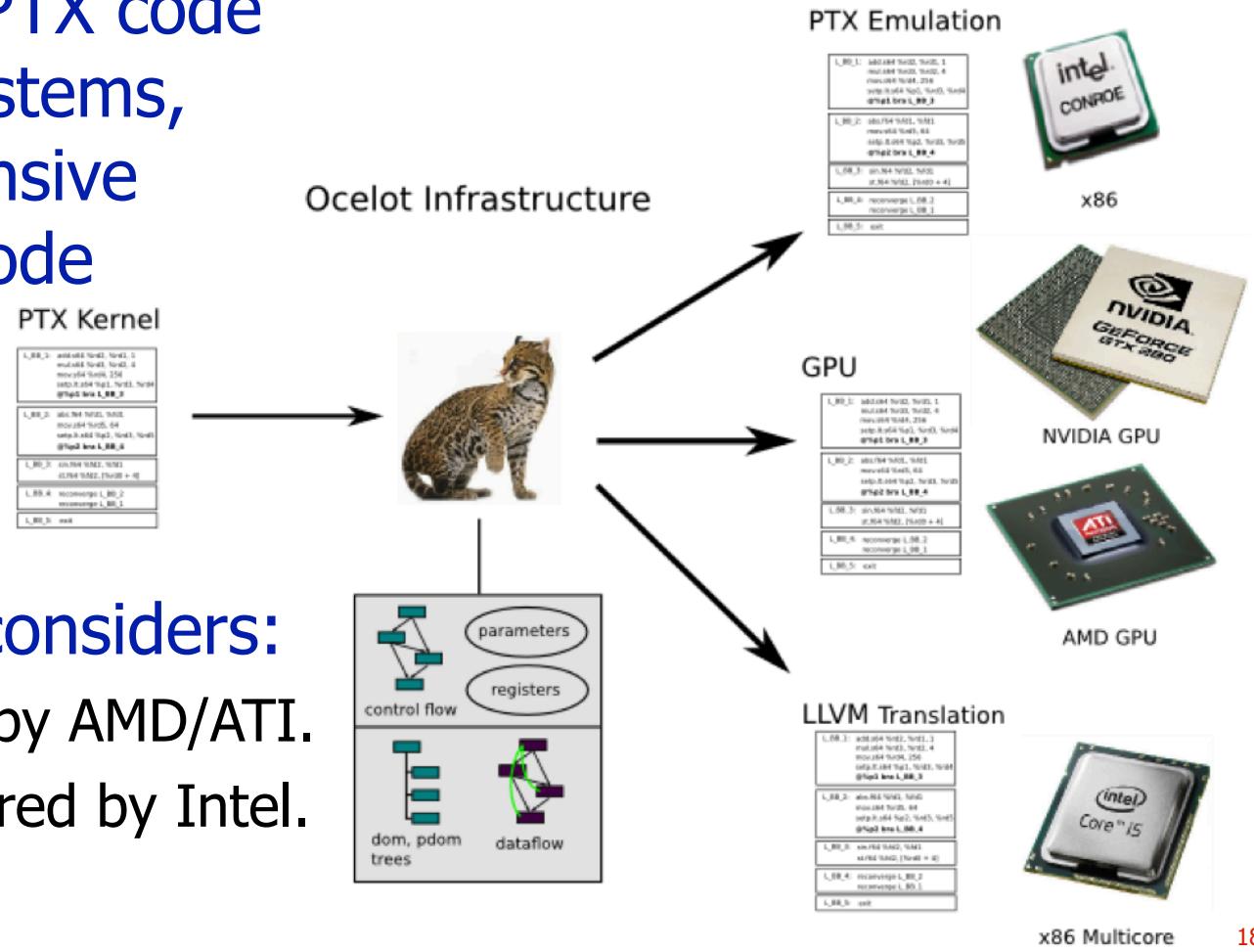


Ocelot

<http://code.google.com/p/gpuocelot>

- It is a dynamic compilation environment for the PTX code on heterogeneous systems, which allows an extensive analysis of the PTX code and its migration to other platforms.

- From Feb'11, also considers:
 - GPUs manufactured by AMD/ATI.
 - CPUs x86 manufactured by Intel.



Swan

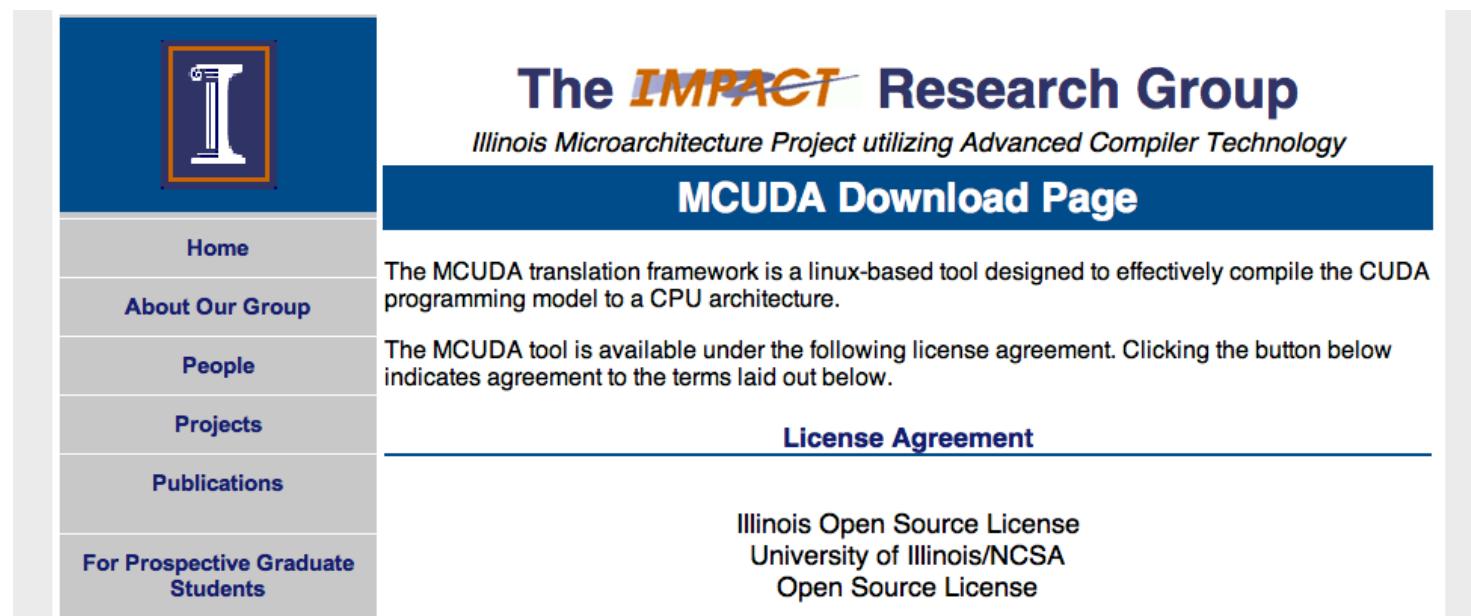
<http://www.multiscalelab.org/swan>

- ➊ It is a source-to-source translator from CUDA to OpenCL:
 - ➊ It provides a common API which abstracts the runtime support of CUDA and OpenCL.
 - ➊ It preserves the convenience of launching CUDA kernels (`<<<blocks,threads>>>`), generating source C code for the entry point kernel functions.
 - ➋ ... but the conversion process requires human intervention.
- ➋ Useful for:
 - ➊ Evaluate OpenCL performance for an already existing CUDA code.
 - ➋ Reduce the dependency from `nvcc` when we compile host code.
 - ➋ Support multiple CUDA compute capabilities on a single binary.
 - ➋ As runtime library to manage OpenCL kernels on new developments.

MCUDA

<http://impact.crhc.illinois.edu/mcuda.php>

- Developed by the IMPACT research group at the University of Illinois.
- It is a working environment based on Linux which tries to migrate CUDA codes efficiently to multicore CPUs.
- Available for free download ...



The screenshot shows the 'MCUDA Download Page' of the IMPACT Research Group website. The page features a sidebar with links to Home, About Our Group, People, Projects, Publications, and For Prospective Graduate Students. The main content area includes the IMPACT logo and text about the MCUDA translation framework and license agreement.

The *IMPACT* Research Group
Illinois Microarchitecture Project utilizing Advanced Compiler Technology

MCUDA Download Page

The MCUDA translation framework is a linux-based tool designed to effectively compile the CUDA programming model to a CPU architecture.

The MCUDA tool is available under the following license agreement. Clicking the button below indicates agreement to the terms laid out below.

License Agreement

Illinois Open Source License
University of Illinois/NCSA
Open Source License

PGI CUDA x86 compiler

<http://www.pgroup.com>

● Major differences with previous tools:

- It is not a translator from the source code, it works at runtime. It allows to build a unified binary which simplifies the software distribution.

● Main advantages:

- **Speed:** The compiled code can run on a x86 platform even without a GPU. This enables the compiler to vectorize code for SSE instructions (128 bits) or the most recent AVX (256 bits).
- **Transparency:** Even those applications which use GPU native resources like texture units will have an identical behavior on CPU and GPU.
- **Availability:** License free for one month if you register as CUDA developer.

I. 3. Accessing CUDA from other languages



Wrappers and interface generators

- CUDA can be incorporated into any language that provides a mechanism for calling C/C++. To simplify the process, we can use general-purpose interface generators.
- SWIG [<http://swig.org>] (Simplified Wrapper and Interface Generator) is the most renowned approach in this respect. Actively supported, widely used and already successful with: AllegroCL, C#, CFFI, CHICKEN, CLISP, D, Go language, Guile, Java, Lua, MxScheme/Racket, Ocaml, Octave, Perl, PHP, Python, R, Ruby, Tcl/Tk.
- A connection with Matlab interface is also available:
 - On a single GPU: Use Jacket, a numerical computing platform.
 - On multiple GPUs: Use MatWorks Parallel Computing Toolbox.

Entry point to CUDA from most popular languages

- Tools available for six different programmer profiles.

1. C programmer

CUDA C, OpenACC.

2. Fortran programmer

CUDA Fortran, OpenACC.

3. C++ programmer

Thrust, CUDA C++.

4. Maths programmer

MATLAB, Mathematica, LabVIEW.

5. C# programmer

GPU.NET.

6. Python programmer

PyCUDA.

Get started today

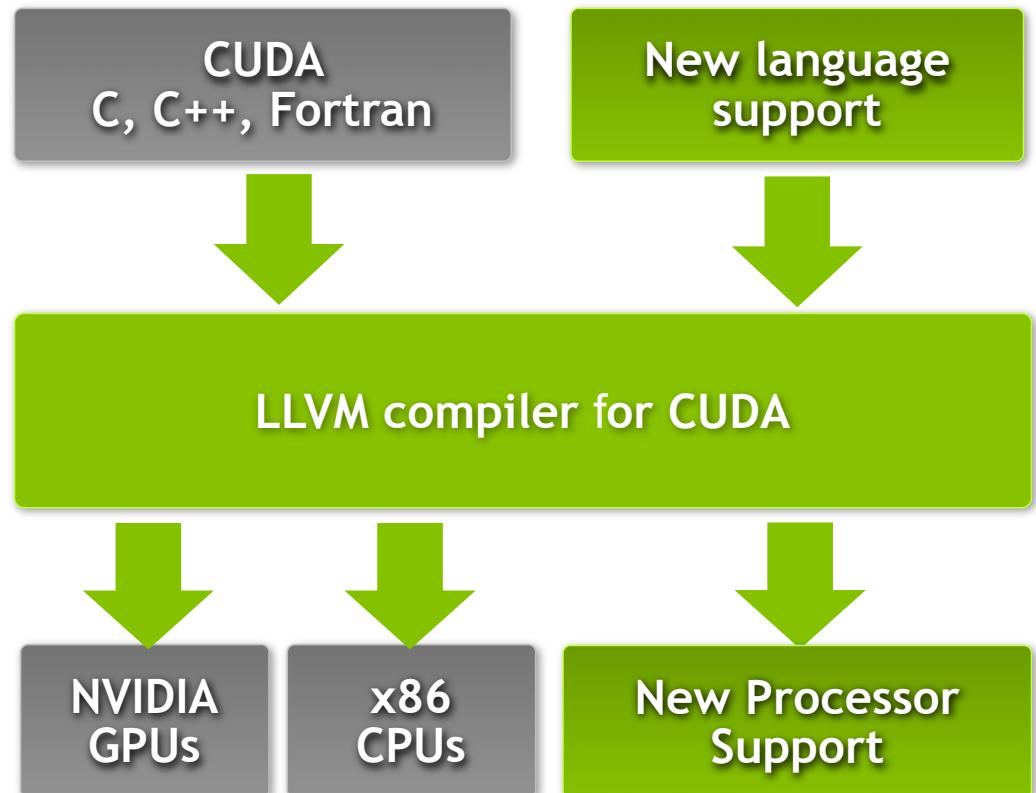
- ➊ These languages are supported on all CUDA GPUs.
 - ➋ It is very likely that you already have a CUDA capable GPU in your laptop or desktop PC (remember IGPs, EPGs, HPUs).
- ➌ Web pages:
- ➍ **CUDA C/C++:** <http://developer.nvidia.com/cuda-toolkit>
 - ➍ **Thrust C++ Template Lib:** <http://developer.nvidia.com/thrust>
 - ➍ **CUDA Fortran:** <http://developer.nvidia.com/cuda-toolkit>
 - ➍ **GPU.NET:** <http://tidepowerd.com>
 - ➍ **PyCUDA (Python):** <http://mathematician.de/software/pycuda>
 - ➍ **MATLAB:** <http://www.mathworks.com/discovery/matlab-gpu.html>
 - ➍ **Mathematica:** <http://www.wolfram.com/mathematica/new-in-8/cuda-and-opencl-support>

A wild card for languages: On Dec'11, source code of the CUDA compiler was accessible

- This does very convenient and efficient to connect with a whole world of:

- Languages on top. For example, adding front-ends for Java, Python, R, DSLs.
- Hardwares underneath. For example, ARM, FPGA, x86.

- CUDA compiler contributed to Open Source LLVM.



I. 4. Using directives: OpenACC



OpenACC: A corporative effort for standardization

OpenACC: Open Programming Standard for Parallel Computing



<http://www.openacc-standard.org>

The OpenACC™ API QUICK REFERENCE GUIDE

The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators.

Most OpenACC directives apply to the immediately following structured block or loop; a structured block is a single statement or a compound statement (C or C++) or a sequence of statements (Fortran) with a single entry point at the top and a single exit at the bottom.



Version 1.0, November 2011

OpenACC: An alternative to computer scientist's CUDA for an average programmer

- ➊ It is a parallel programming standard for accelerators based on directives (like OpenMP), which:
 - ➊ Are inserted into C, C++ or Fortran programs.
 - ➊ Drive the compiler to parallelize certain code sections.
- ➋ Goal: Targeted to an average programmer, code portable across parallel and multicore processors.
- ➌ Early development and commercial effort:
 - ➊ The Portland Group (PGI).
 - ➊ Cray.
- ➍ First supercomputing customers:
 - ➊ United States: Oak Ridge National Lab.
 - ➊ Europe: Swiss National Supercomputing Centre.

OpenACC: Directives

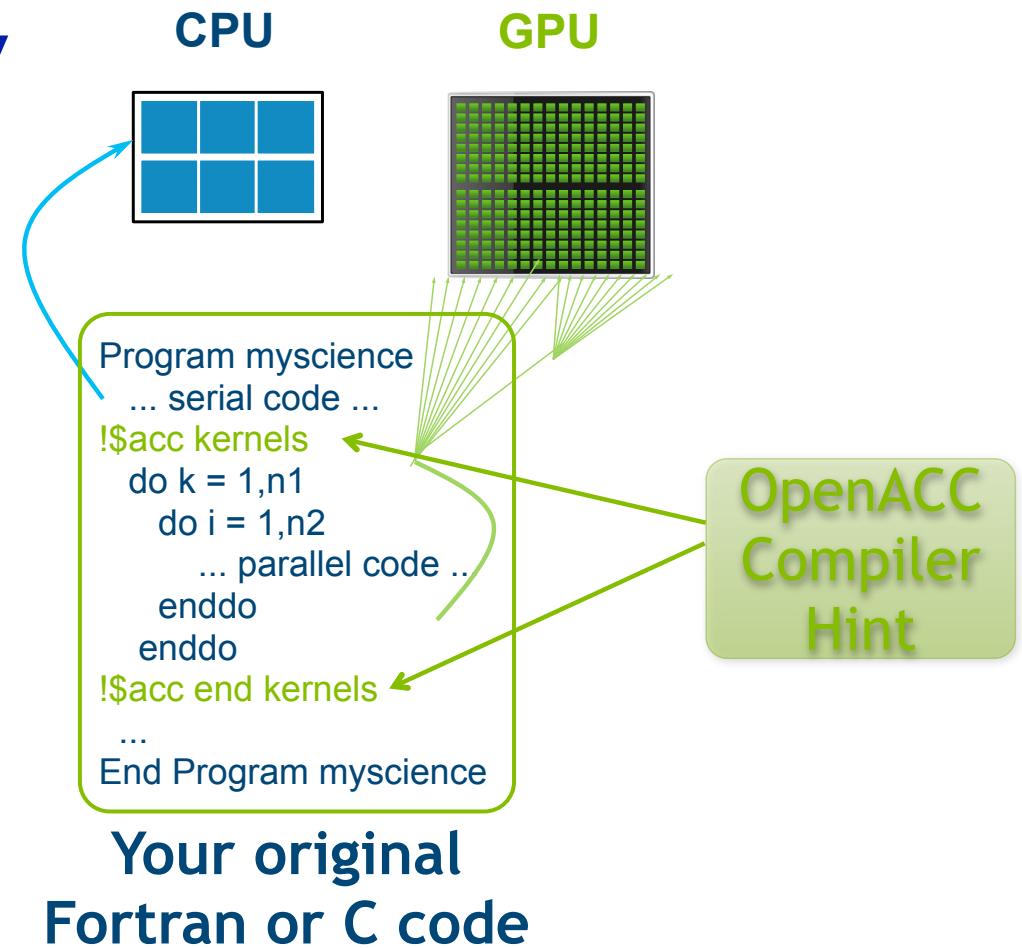
- ➊ Directives provide a **common code base** that is
 - ➊ Multi-platform.
 - ➊ Multi-vendor.
- ➋ This brings an **open** way to preserve investment in legacy applications by enabling an easy **migration** path to accelerated computing.
- ➌ GPU directives allow complete access to the massive parallel **power** of a GPU.
- ➍ Optimizing code with directives is quite **easy**, especially compared to CPU threads or writing CUDA kernels.
- ➎ A big achievement is **avoiding restructuring** of existing code for production applications.

OpenACC: How directives work

- Starting from simple hints, the compiler parallelizes the code.

- It works on:

- Many-core GPUs.
- Multi-core CPUs.



Two basic steps to get started

- ➊ Step 1: Annotate source code with directives.

- ➊ !\$acc data copy(util1,util2,util3) copyin(ip,scp2,scp2i)
 - ➋ !\$acc parallel loop
 - ➌ ... <source code>
 - ➍ !\$acc end parallel
 - ➎ !\$acc end data

- ➋ Step 2: Compile & run.

- ➌ pgf90 -ta=nvidia -Minfo=accel file.f

An example

```

!$acc data copy (A,Anew)
iter=0
do while ( err > tol .and. iter < iter_max )

iter = iter +1
err=0._fp_kind

!$acc kernels
do j=1,m
do i=1,n
Anew(i,j) = .25_fp_kind *( A(i+1,j) + A(i-1,j) &
                           +A(i,j-1) + A(i,j+1))
err = max( err, Anew(i,j)-A(i,j))
end do
end do

!$acc end kernels
IF (mod(iter,100)==0 .or. iter == 1) print *, iter, err
A= Anew

end do
!$acc end data

```

Copy arrays into GPU memory within data region

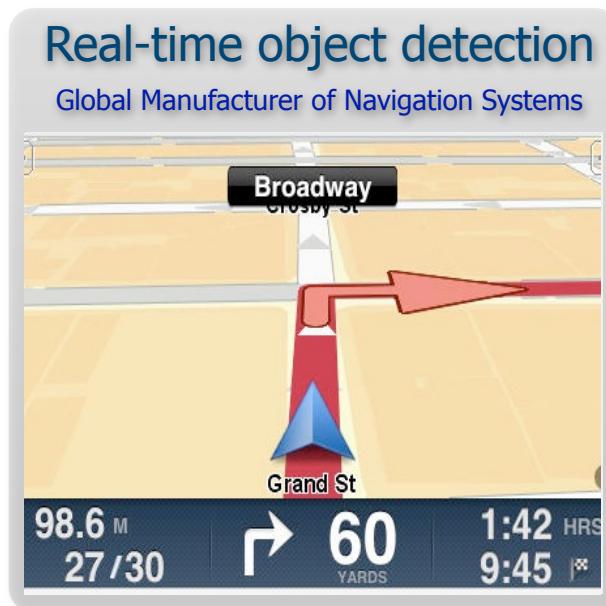
Parallelize code inside region

Close off parallel region

Close off data region,
copy data back

The key question is: How much performance do we lose?

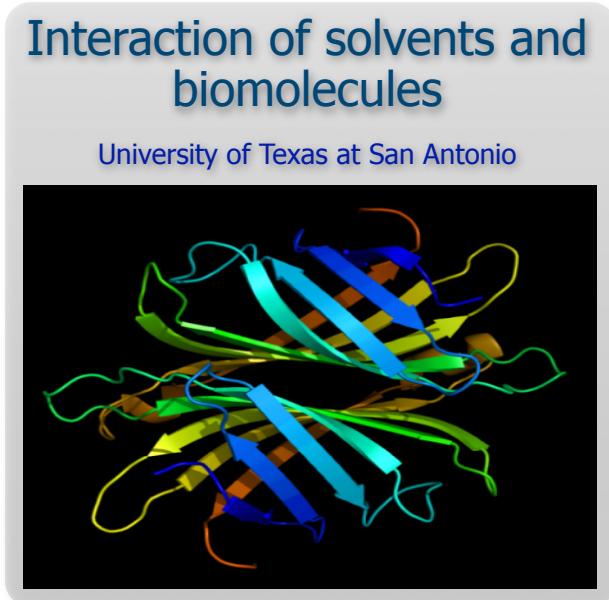
- Some results say only 5-10% vs. CUDA in "some" cases. Other sources say 5x gains investing a week or even a day.
- But this factor is more application-dependent than influenced by programmer skills.



5x in 1 week



2x in 4 hours



5x in 1 day

More recent examples

Lifecycles of fish
in Australia

University of Melbourne



65x in 2 Days

Stars and galaxies
12.5B years ago

University of Groningen



5.6x in 5 Days

Neural networks in
self-learning robot

The University of Plymouth



4.7x in 4 Hours

A witness from a recent OpenACC workshop at Pittsburgh Supercomputing Center

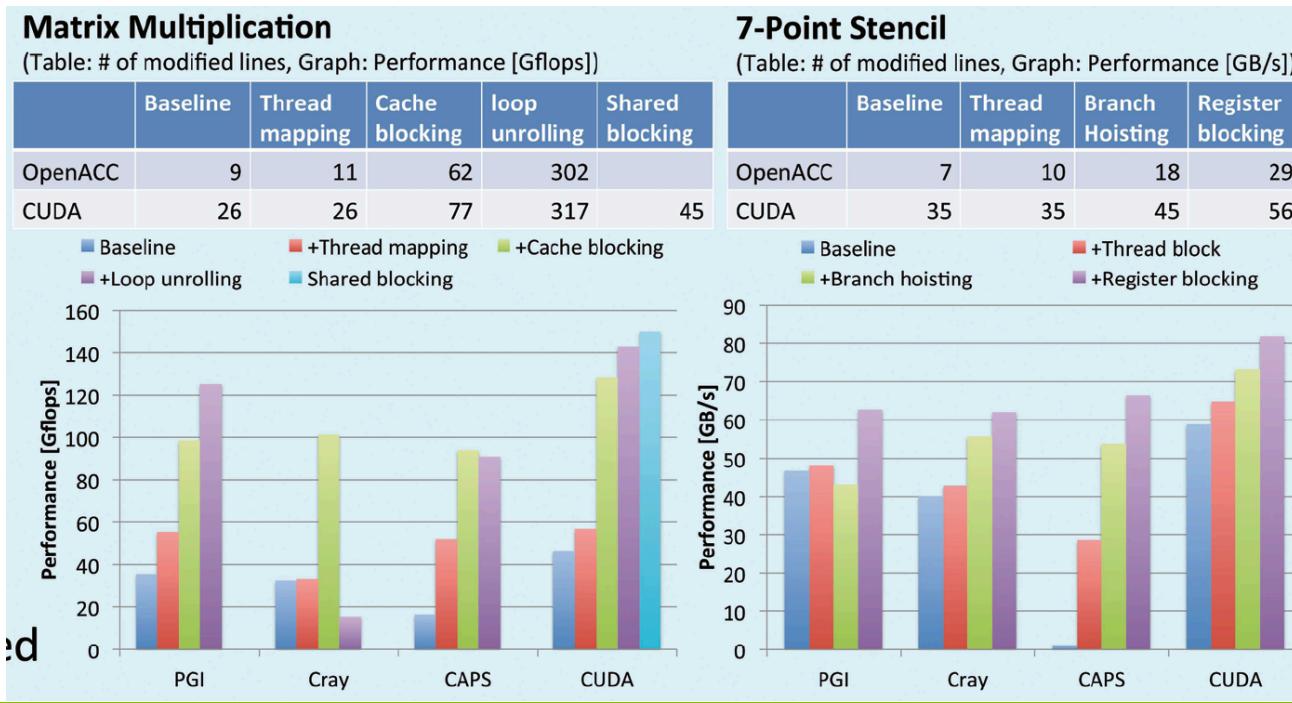
By end of second day
10x on one atmospheric kernel
6 directives

Technology Director
National Center for Atmospheric
Research (NCAR)



More case studies from GTC'13: 3 OpenACC compilers [PGI, Cray and CAPS]

- Performance on M2050 GPU (Fermi, 14x 32 cores), without counting the CPU-GPU transfer overhead.
- Matrix Multiplication size: 2048x2048.
- 7-point Stencil: 3D array size: 256x256x256.



Source: "CUDA vs. OpenACC: Performance Case Studies", by T. Hoshino, N. Maruyama, S. Matsuoka.

Start now with OpenACC directives

- ➊ Sign up for a free trial of the directives compiler (thanks to PGI), and get also tools for quick ramp (see <http://www.nvidia.com/gpudirectives>)
- ➋ A compiler is also available from CAPS for \$199/199€.

 [Search NVIDIA](#) [USA - United States](#)

[DOWNLOAD DRIVERS](#) [COOL STUFF](#) [SHOP](#) [PRODUCTS](#) [TECHNOLOGIES](#) [COMMUNITIES](#) [SUPPORT](#)

TESLA

NVIDIA Home > Products > High Performance Computing > OpenACC GPU Directives

GPU COMPUTING SOLUTIONS

- Main
- What is GPU Computing?
- Why Choose Tesla
- Industry Software Solutions
- Tesla Workstation Solutions
- Tesla Data Center Solutions
- Tesla Bio Workbench
- Where to Buy
- Contact US
- Sign up for Tesla Alerts
- Fermi GPU Computing Architecture

SOFTWARE AND HARDWARE INFO

- Tesla Product Literature
- Tesla Software Features
- Software Development Tools
- CUDA Training and Consulting Services
- GPU Cloud Computing Service Providers
- OpenACC GPU Directives

Accelerate Your Scientific Code with OpenACC
The Open Standard for GPU Accelerator Directives

Thousands of cores working for you.

Based on the [OpenACC](#) standard, GPU directives are the easy, proven way to accelerate your scientific or industrial code. With GPU directives, you can accelerate your code by simply inserting compiler hints into your code and the compiler will automatically map compute-intensive portions of your code to the GPU. Here's an example of how easy a single directive hint can accelerate the calculation of pi. With GPU directives, you can get started and see results in the same afternoon.

```
#include <stdio.h>
#define N 10000
int main(void) {
    double pi = 0.0f; long i;
    #pragma acc region for
    for (i=0; i<N; i++) {
        double t= (double)((i+0.5)/N);
        pi +=4.0/(1.0+t*t);
    }
    printf("pi=%f\n",pi/N);
    return 0;
}
```

By starting with a free, 30-day trial of PGI directives today, you are working on the technology that is the foundation of the OpenACC directives standard. OpenACC is:

"I have written micron (written in Fortran 90) properties of two and dimensional magnetic directives approach export my existing code perform my computation which resulted in a significant speedup (more than 20 times faster)." [Learn more](#)

"The PGI compiler is not just how powerful it is software we are writing times faster on the NVidia GPU are very pleased and excited about future uses. It's like overhauling a supercomputer." [Learn more](#)

Professor M. Amin Kavousi
University of Houston

Dr. Kerry Black
University of Melbourne

II. Programming examples: Six ways to SAXPY on GPUs



What does SAXPY stand for? Single-precision Alpha X Plus Y. It is part of BLAS Library.

```
void saxpy_serial(float ... )  
{  
    for (int i = 0; i < n; ++i)  
        y[i] = a*x[i] + y[i];  
}
```

- Using this basic code, we will illustrate six different ways of programming the GPU:
 - CUDA C.
 - CUBLAS Library.
 - CUDA Fortran.
 - Thrust C++ Template Library.
 - C# with GPU.NET.
 - OpenACC.

1. CUDA C

Standard C code:

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invoke SAXPY kernel (serial on 1M elements)
saxpy_serial(4096*256, 2.0, x, y);
```

CUDA code for a parallel execution on GPU:

```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}
// Invoke SAXPY kernel (parallel on 4096 blocks of 256 threads)
saxpy_parallel<<<4096, 256>>>(4096*256, 2.0, x, y);
```

2. CUBLAS Library

Sequential BLAS code

```
int N = 1 << 20;
// Utiliza la librería BLAS de tu elección

// Invoke SAXPY routine (serial on 1M elements)
blas_saxpy(4096*256, 2.0, x, 1, y, 1);
```

cuBLAS parallel code

```
int N = 1 << 20;
cublasInit();
cublasSetVector (N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector (N, sizeof(y[0]), y, 1, d_y, 1);
// Invoke SAXPY routine (parallel on 1M elements)
cublasSaxpy (N, 2.0, d_x, 1, d_y, 1);
cublasGetVector (N, sizeof(y[0]), d_y, 1, y, 1);
cublasShutdown();
```

3. CUDA Fortran

Standard Fortran

```

module my module contains
    subroutine saxpy (n, a, x, y)
        real :: x(:), y(:), a
        integer :: n, i
        do i=1,n
            y(i) = a*x(i) + y(i);
        enddo
    end subroutine saxpy
end module mymodule

program main
    use mymodule
    real :: x(2**20), y(2**20)
    x = 1.0, y = 2.0

    $ Perform SAXPY on 1M elements
    call saxpy(2**20, 2.0, x, y)

end program main

```

Parallel Fortran

```

module mymodule contains
    attributes(global) subroutine saxpy(n, a, x, y)
        real :: x(:), y(:), a
        integer :: n, i
        attributes(value) :: a, n
        i = threadIdx%x + (blockIdx%x-1) * blockDim%x
        if (i<=n) y(i) = a*x(i) + y(i)
    end subroutine saxpy
end module mymodule

program main
    use cudafor; use mymodule
    real, device :: x_d(2**20), y_d(2**20)
    x_d = 1.0, y_d = 2.0

    $ Perform SAXPY on 1M elements
    call saxpy<<<4096,256>>>(2**20, 2.0, x_d, y_d)
    y = y_d
end program main

```

4.1.CUDA C++: Develop Generic Parallel Code

- CUDA C++ features enable sophisticated and flexible applications and middleware:

- Class hierarchies.
- `__device__` methods.
- Templates.
- Operator overloading.
- Functors (function objects).
- Device-side new/delete.
- ...

```
template <typename T>
struct Functor {
    __device__ Functor(_a) : a(_a) {}
    __device__ T operator(T x) { return a*x; }
    T a;
}

template <typename T, typename Oper>
__global__ void kernel(T *output, int n) {
    Oper op(3.7);
    output = new T[n]; // dynamic allocation
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        output[i] = op(i); // apply functor
}
```

it

4.2. Thrust C++ STL

- Thrust is an open source parallel algorithms library which resembles C++ Standard Template Library (STL).

Major features:

- High-level interface:**

- Enhances developer productivity.
- Enables performance portability between GPUs and CPUs.

- Flexible:**

- CUDA, OpenMP and TBB (Thread Building Blocks) backends.
- Extensible and customizable.
- Integrates with existing software.

- Efficient:**

- GPU code written without directly writing any CUDA kernel calls.

4.2. Thrust C++ STL (cont.)

Serial C++ Code with STL and Boost

```
int N = 1<<20;  
std::vector<float> x(N), y(N);  
...  
  
// Invoke SAXPY on 1M elements  
std::transform(x.begin(), x.end()  
( ),  
               y.begin(), x.end()  
( ),  
               2.0f * _1 +  
               _2);
```

Parallel C++ Code

```
int N = 1<<20;  
thrust::host_vector<float> x(N), y(N);  
...  
...  
thrust::device_vector<float> d_x = x;  
thrust::device_vector<float> d_y = y;  
  
// Invoke SAXPY on 1M elements  
thrust::transform(x.begin(), x.end(),  
                  y.begin(), y.begin(),  
                  2.0f * _1 + _2);
```

<http://www.boost.org/libs/lambda>

<http://developer.nvidia.com/thrust>

5. C# with GPU.NET

Standard C#

```

private static
void saxpy (int n, float a,
           float[] a, float[] y)
{
    for (int i=0; i<n; i++)
        y[i] = a*x[i] + y[i];
}

int N = 1<<20;

// Invoke SAXPY on 1M elements
saxpy(N, 2.0, x, y)

```

Parallel C#

```

[kernel]
private static
void saxpy (int n, float a,
           float[] a, float[] y)
{
    int i = BlockIndex.x * BlockDimension.x +
          ThreadIndex.x;
    if (i < n)
        y[i] = a*x[i] + y[i];
}

int N = 1<<20;

Launcher.SetGridSize(4096);
Launcher.SetBlockSize(256);

// Invoke SAXPY on 1M elements
saxpy(2**20, 2.0, x, y)

```

6. OpenACC Compiler Directives

Parallel C Code

```

void saxpy (int n, float a,
            float[] a, float[] y)
{
#pragma acc kernels
    for (int i=0; i<n; i++)
        y[i] = a*x[i] + y[i];
}

...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y)
...

```

Parallel Fortran Code

```

subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
$!acc kernels
    do i=1. n
        y(i) = a*x(i) + y(i)
    enddo
$!acc end kernels
end subroutine saxpy

...
$ Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...

```

Summary

- There is support for all these 6 approaches on every CUDA GPU (more than 400 million as of 2013). It is very likely that you have one of those within your laptop/desktop.

1. CUDA C/C++

<http://developer.nvidia.com/cuda-toolkit>

2. CUDA Fortran

<http://developer.nvidia.com/cuda-fortran>

3. CUBLAS Library

<http://developer.nvidia.com/cublas>

4. Thrust

<http://developer.nvidia.com/thrust>

5. C# with GPU.NET

<http://tidepowerd.com>

6. OpenACC

<http://developer.nvidia.com/openacc>