Heterogeneous Computing

Add GPUs: Accelerate Applications

**CPUs**: designed to run a few tasks quickly.

**GPUs**: designed to run many tasks efficiently.
Minimum Change, Big Speed-up

Application Code

GPU

Compute-Intensive Functions

Rest of Sequential CPU Code

CPU
Ways to Accelerate Applications

Applications

Libraries

Directives

Programming Languages (CUDA, ..)

High Level Languages (Matlab, ..)

Easiest Approach

Maximum Performance

No Need for Programming Expertise
GPU Accelerated Libraries
“Drop-in” Acceleration for Your Applications

- NVIDIA cuBLAS
- NVIDIA cuRAND
- NVIDIA cuSPARSE
- NVIDIA NPP
- GPU VSIPL
- CULA tools
- MAGMA
- NVIDIA cuFFT
- Rogue Wave Software
- CUSP
- LibJacket
- Thrust

Vector Signal Image Processing
GPU Accelerated Linear Algebra
Matrix Algebra on GPU and Multicore
C++ STL Features for CUDA

GPU Accelerated Linear Algebra on GPU and Multicore
OpenACC Directives

Easy, Open, Powerful

- Simple Compiler hints
- Works on multicore CPUs & many core GPUs
- Compiler Parallelizes code
- Future Integration into OpenMP standard planned

http://www.openacc-standard.org

Your original Fortran or C code

Program myscience
... serial code ...
!$acc region
do k = 1,n1
do i = 1,n2
... parallel code ...
endo
do
endo
!$acc end region
... End Program myscience

CPU

GPU

OpenACC Compiler Hint
OpenACC

- Compiler directives to specify parallel regions in C, C++, Fortran
  - OpenACC compilers offload parallel regions from host to accelerator
  - Portable across OSes, host CPUs and accelerators

- Create high-level heterogeneous programs
  - Without explicit accelerator initialization,
  - Without explicit data or program transfers between host and accelerator

- Programming model allows programmers to start simple
  - Enhance with additional guidance for compiler on loop mappings, data location, and other performance details
Basic Concepts

For efficiency, decouple data movement and compute off-load
Directive Syntax

- **Fortran**
  
  ```fortran
  !$acc directive [clause [,] clause] ...]
  Often paired with a matching end directive surrounding a structured code block
  !$acc end directive
  ```

- **C**
  
  ```c
  #pragma acc directive [clause [,] clause] ...]
  Often followed by a structured code block
  ```
OpenACC Directive Set

- Parallel Constructs
  - `#pragma acc parallel [clause [[,, clause]]...] new-line`
- Data Constructs
  - `#pragma acc data [clause [[,, clause]]...] new-line`
- Loop Constructs
  - `#pragma acc loop [clause [[,, clause]]...]new-line`
- Runtime Library Routines
- Cache Directives

And few others…
Jacobi Relaxation

\[
\text{iter} = 0 \\
\text{do while ( err .gt tol .and. iter .gt. iter\_max )}
\]

\[
\text{iter} = \text{iter} + 1 \\
\text{err} = 0.0
\]

\[
\text{do j=1,m} \\
\quad \text{do i=1,n}
\]

\[
\text{Anew(i,j) = 0.25 * (A(i+1,j) + A(i-1,j) + A(i,j-1) + A(i, j+1))}
\]

\[
\text{err} = \text{max( err, abs(Anew(i,j)-A(i,j)) )}
\]

\[
\text{end do}
\]

\[
\text{end do}
\]

\[
\text{if( mod(iter,100).eq.0 .or. iter.eq.1 ) print*, iter, err}
\]

\[
\text{A = Anew}
\]

\[
\text{end do}
\]
OpenMP CPU Implementation

iter = 0
do while ( err .gt tol .and. iter .gt. iter_max )

    iter = iter + 1
    err = 0.0
!$omp parallel do shared(m,n,Anew,A) reduction(max:err)
    do j=1,m
        do i=1,n
            Anew(i,j) = 0.25 * (A(i+1,j) + A(i-1,j) + A(i,j-1) + A(i, j+1)
            err = max( err, abs(Anew(i,j)-A(i,j)) )
        end do
    end do
!$omp end parallel do
if( mod(iter,100).eq.0 ) print*, iter, err
    A = Anew
end do
OpenACC GPU Implementation

```c
!$acc data copy(A,Anew)
iter = 0
do while ( err .gt tol .and. iter .gt. iter_max )
    iter = iter + 1
    err = 0.0
!$acc parallel reduction( max:err )
do j=1,m
    do i=1,n
        Anew(i,j) = 0.25 * (A(i+1,j) + A(i-1,j) + A(i,j-1) + A(i,j+1)
        err = max( err, abs(Anew(i,j)-A(i,j)) )
    end do
end do
!$acc end parallel
if( mod(iter,100).eq.0 ) print*, iter, err
A = Anew
end do
!$acc end data
```

- Copy arrays into GPU memory within region
- Parallelise code inside region
- Close off parallel region
- Close off data region, copy data back
Improved OpenACC GPU Implementation

```fortran
!$acc data copyin(A), copyout(Anew)
iter = 0
do while ( err .gt tol .and. iter .gt. iter_max )
    iter = iter + 1
    err = 0.0
!$acc parallel reduction( max:err )
    do j=1,m
        do i=1,n
            Anew(i,j) = 0.25 * ( A(i+1,j) + A(i-1,j) &
                                A(i, j-1) + A(i, j+1)
                             err = max( err, abs(Anew(i,j)-A(i,j)) )
        end do
    end do
!$acc end parallel
    if( mod(iter,100).eq.0 ) print*, iter, err
    A = Anew
end do
!$acc end data
```

Reduced data movement
More Parallelism

```fortran
!$acc data copyin(A), create(Anew)
iter = 0
do while ( err .gt. tol .and. iter .gt. iter_max )

   iter = iter + 1
   err = 0.0
!$acc parallel reduction( max:err )
   do j=1,m
      do i=1,n
         Anew(i,j) = 0.25 * ( A(i+1,j ) + A(i-1,j ) &
                           A(i,  j-1) + A(i,  j+1) )
         err = max( err, abs(Anew(i,j)-A(i,j)) )
      end do
   end do
!$acc end parallel
   if( mod(iter,100).eq.0 ) print*, iter, err
!$acc parallel
   A = Anew
!$acc end parallel
end do
!$acc end data
```

- **Anew** now only exists on GPU
- Find maximum over all iterations
- Add second parallel region inside data region
More Performance

```fortran
!$acc data copyin(A), create(Anew)
iter = 0
do while ( err .gt. tol .and. iter .gt. iter_max )

    iter = iter + 1
    err = 0.0
!$acc kernels loop reduction( max:err ), gang(32), worker(8)
    do j=1,m
        do i=1,n
            Anew(i,j) = 0.25 * ( A(i+1,j ) + A(i-1,j ) &
            A(i, j-1) + A(i, j+1) )
            err = max( err, abs(Anew(i,j)-A(i,j)) )
        end do
    end do
!$acc end kernels loop
    if( mod(iter,100).eq.0 ) print*, iter, err
!$acc parallel
    A = Anew
!$acc end parallel
end do
!$acc end data
```

30% faster than default schedule
OpenACC: Small Effort, Real Impact

**Large Oil Company**
- Solving billions of equations iteratively for oil production at world’s largest petroleum reservoirs
- 3x in 7 days

**Univ. of Houston**
- Studying magnetic systems for innovations in magnetic storage media and memory, field sensors, and biomagnetism
- 20x in 2 days
- Prof. M.A. Kayali

**Uni. Of Melbourne**
- Better understand complex reasons by lifecycles of snapper fish in Port Phillip Bay
- 65x in 2 days
- Prof. Kerry Black

**Ufa State Aviation**
- Generating stochastic geological models of oilfield reservoirs with borehole data
- 7x in 4 Weeks
- Prof. Arthur Yuldashev

**GAMESS-UK**
- Used for various fields such as investigating biofuel production and molecular sensors.
- 10x
- Dr. Wilkinson, Prof. Naidoo
CUDA 4.1 Highlights

**Advanced Application Development**
- New LLVM-based compiler
- 3D surfaces & cube maps
- Peer-to-Peer between processes
- GPU reset with nvidia-smi
- New GrabCut sample shows interactive foreground extraction
- New code samples for optical flow, volume filtering and more…

**GPU-Accelerated Libraries**
- 1000+ new imaging functions
- Tri-diagonal solver 10x faster vs. MKL
- MRG32k3a & MTGP11213 RNGs
- New Bessell functions in Math lib
- 2x faster matrix-vector w/ HYB-ELL
- Boost-style placeholders in Thrust
- Batched GEMM for small matrices

**New & Improved Developer Tools**
- Re-Designed Visual Profiler
- Parallel Nsight 2.1
- Multi-context debugging
- assert() in device code
- Enhanced CUDA-MEMCHECK
New LLVM-based CUDA Compiler

- Delivers up to 10% faster application performance
- Faster compilation for increased developer productivity
- Modern compiler with broad support
  - Will bring more languages to the GPU
  - Easier to support CUDA to more platforms
NVIDIA Opens Up CUDA Platform

CUDA Compiler Source for Researchers & Tools Vendors

Enables
New Language Support

New Processor Support

Apply for early access at

© NVIDIA Corporation 2012
GPU Technology Conference 2012
May 14-17 | San Jose, CA

The one event you can’t afford to miss
- Learn about leading-edge advances in GPU computing
- Explore the research as well as the commercial applications
- Discover advances in computational visualization
- Take a deep dive into parallel programming

Ways to participate
- Speak - share your work and gain exposure as a thought leader
- Register - learn from the experts and network with your peers
- Exhibit/Sponsor - promote your company as a key player in the GPU ecosystem

www.gputechconf.com
Heterogeneous/hybrid Computing is the future

OpenACC Directives provide a standardized way to hybrid computing
  - Easy, Open and Powerful

Use highly optimized GPU libraries
Use CUDA for maximum performance

Don’t wait and start today !!
GPU Programming
with CUDA and OpenACC
Axel Koehler - NVIDIA (akoehler@nvidia.com)