OpenMP/OpenACC compiler directives

GPU/MIC key architectural features

- Optimized for high degree of regular parallelism
- Classically optimized for low precision
  - Fermi supports double precision at ½ single precision bandwidth
- High bandwidth memory (Fermi supports ECC)
- Highly multithreaded (slack parallelism)
- Hardware thread scheduling
- Non-coherent software-managed data caches
  - Fermi has two-level hardware data cache
- No multiprocessor memory model guarantees
  - some guarantees with fence operations

Data-parallel Programming

- Think of the CPU or the Intel Phi as a massively-threaded co-processor
- Write "kernel" functions that execute on the device -- processing multiple data elements in parallel

- Keep it busy! ⇨ massive threading
- Keep your data close! ⇨ local memory
- Minimize data transfers and increase data reuse

- Avoid "branchy" code

Running on the Intel Phi

- Connect to stampede
- Submit an interactive job requesting one node with all cores
  - srun -A TG-EAR140026 -p development -t 0:30:00 -n 32 --pty /bin/bash –l
- Compile for the mic
  - icc –mmic –O3 hello.c –o hello.exe.mic
  - ifort –mmic –O3 hello.f90 – hello.exe.mic
- Running
  - ssh mic0 # then run on the MIC
  - micrun ./hello.exe.mic
  - ./hello.exe.mic

Environment variables

- If ssh to MIC then
  - OMP_NUM_THREADS
- If launching
  - MIC_OMP_NUM_THREADS

Performance Considerations

- Vectorization and Parallelization necessary for performance
  - Otherwise you get 1 GHz Pentium performance
- Use
  - -vec-report3

Offloading

- Have a block of code to be executed on the MIC.
- Directive based approach
  - No rewrite
  - Simple
  - Data movement to MIC
  - Optimization may require changes
- #pragma   offload target(mic)
- !dir$ offload target(mic)

Examples

- See notebook

OpenACC Directives

- Preprocessed from *C/C*++ and Fortran sources
- Developed by PGI, Cray and NVIDIA with support from CAPS
- Based on the PGI Accelerator programming model
- Directives/Pragmas are added to existing code to identify accelerator "codelets”
- Supported by Cray and PGI

PGI tools

- PGI compiler module
  - module load pgi/pgi-13.6
- Identify your GPU
  - pgaccelinfo
- Build code for the GPU and the CPU
  - pgcc -ta=nvidia,host,time -Minfo –fast test.c
  - Creates a PGI unified binary

Accelerator programming

- Allocate data on the GPU
- Move data from host or initialize on GPU
- Launch kernels
- Gather results from GPU
- Deallocate data

Writing Many-Core Programs

- Appropriate algorithm: lots of parallelism
  - Lots of MIMD parallelism to fill the multiprocessors
  - Lots of SIMD parallelism to fill cores on a multiprocessor
  - Lots more MIMD parallelism to fill multithreading parallelism
  - High compute intensity
  - Insert directives, read feedback for parallelism hindrances
    - Dependence relations, pointers
    - Scalars live out from the loop
    - Arrays that need to be private
    - Iterate

Writing Accelerator Programs

- Tune data movement between Host and Accelerator
  - Read compiler feedback about data moves
  - Minimize amount of data moved
  - Minimize frequency of data moves
  - Minimize noncontiguous data moves
  - Optimize data allocation in device memory
  - Optimize allocation in host memory (esp. For C)
  - Insert local, copyin, copyout clauses
  - Use data regions, update clauses

Tuning Accelerator Programs

- Tune kernel code
  - **Profile the code (cudaprof, pgcollect, -ta=nvidia,time)**
  - Experiment with kernel schedule using loop directives
  - Unroll clauses
  - **Enable experimental optimizations (-ta=nvidia,o3)**
  - Single precision vs. Double precision
  - **24-bit multiply for indexing (-ta=nvidia,mul24)**
  - **Low precision transcendentals (-ta=nvidia,fastmath)**

Basic OpenACC concepts

- Fortran accelerator directive syntax
  - **!$acc Directive [clause]...**
  - &continuation
- C accelerator directive syntax
  - **#pragma acc *directive*[*clause*]...**
  - continue to next line with backslash

Region

- **region is single-entry/single-exit region**
  - in Fortran, delimited by begin/end directives
  - in C, a single statement, or {...} region
  - no jumps into/out of region, no return
- Compute region contains loops to send to GPU
  - loop iterations translated to GPU threads
- Data region encloses compute regions
  - data moved at region boundaries

Best candidates for acceleration

- Nested parallel loops
  - iterations map to threads
  - parallelism means threads are independent
  - nested loops means lots of parallelism
- Regular array indexing
  - **allows for stride-1 array fetches**

Behind the Scenes

- Compiler determines parallelism
- Compiler generates thread code
  - Split up the iterations into threads, thread groups
  - Inserts code to use software data cache
  - Accumulate partial sum
  - Second kernel to combine final sum
- Compiler also inserts data movement
  - Compiler or user determines what data to move
  - Data moved at boundaries of data/compute region

Program Execution Model

- Host
  - Executes most of the program
  - Allocates accelerator memory
  - Initiates data copy from host memory to accelerator
  - Sends kernel code to accelerator
  - Queues kernels for execution on accelerator
  - Waits for kernel completion
  - Initiates data copy from accelerator to host memory
  - Deallocates accelerator memory
- Accelerator
  - Executes kernels, one after another
  - Concurrently, may transfer data between host and accelerator

Performance Tips

- Data movement between Host and Accelerator
  - Minimize amount of data
  - Minimize number of data moves
  - Minimize frequency of data moves
  - Optimize data allocation in device memory
- Parallelism on accelerator
  - Lots of MIMD parallelism to fill the multiprocessors
  - Lots of SIMD parallelism to fill cores on a multiprocessor
  - Lots more MIMD parallelism to fill multithreading parallelism

Compute Region Clauses

- Conditional
  - if(condition)
- Data allocation clauses
  - copy(list)
  - **copyin(*list*)**
  - **copyout(*list*)**
  - **local(*list*)**
  - data in the lists must be distinct (data in only one list)
- Data update clauses
  - updatein(list) or update device(list)
  - updateout(list) or update host (list)
  - data must be in a data allocate clause for an enclosing data region

Possible Loop Schedules

- **PARALLEL**– parallelize across multi-processors
- **VECTOR** – SIMD vectorize within a multi-processor
- **VECTOR(n)** – SIMD vectorize within a multi-processor in strips of width ‘n’
- **PARALLEL, VECTOR(n)** – parallelize and SIMD vectorize the same loop
- **SEQ**–execute the loop sequentially on each thread processor
- **HOST**–execute the loop on the host

Compute Region Clauses

- copyin and updatein (host to gpu) at region entry
- copyout and updateout (gpu to host) at region exit

Data Region

- C

```
#pragma acc data region
{
....
}
```

- Fortran

```
!$acc data region
....
!$acc end data region
```

- May be nested and may contain compute regions
- May not be nested within a compute region

Data Region Clauses

- Data allocation clauses
  - copy(list)
  - **copyin(*list*)**
  - **copyout(*list*)**
  - **local(*list*)**
  - data in the lists must be distinct (data in only one list)
  - may not be in a data allocate clause for an enclosing data region
- Data update clauses
  - updatein(list)orupdate device(list)
  - updateout(list)orupdate host (list)
  - data must be in a data allocate clause for an enclosing data region

Data Region Update Directives

- update host(list)
- update device(list)
  - data must be in a data allocate clause for an enclosing data region
  - both may be on a single line
    - **update host(*list*) device(*list*)**

Statements in a Compute Region

- Arithmetic
  - C: int, float, double, struct
  - F: integer, real, double precision, complex, derived types
  - Loops, Ifs
  - Kernel loops must be rectangular: trip count is invariant
- Obstacles with C
  - **unbound pointers –use restrict key word, or –Msafeptr, or –Mipa=fast**
  - **default is double –use float constants (0.0f), or –Mfcon, and float intrinsics**
- Obstacles with Fortran
  - Fortran pointer attribute is not supported

Intrinsics

- C
  - #include <accelmath.h>
- Fortran

Other Functions

- Libm routines
  - Use libm
  - #include <accelmath.h>
- Device builtin routines
  - Use cudadevice
  - #include <cudadevice.h>

Runtime Routines Summary

- **acc_get_num_devices(acc_device_nvidia)**
- **acc_set_device_num(n, acc_device_nvidia)**
- **acc_set_device(acc_device_nvidia| acc_device_host)**
- **acc_get_device()**
- **acc_init(acc_device_nvidia| acc_device_host)**

Porting

Get the code in the right “shape”:

- parallel nested loops
- long rectangular loop limits
- may have to inline calls
- loop unrolling
- can do this incrementally
- test on the host
- techniques: loop fusion, routine inlining, loop reordering

Initial GPU Execution

- Add region directive
  - compile and look at messages
  - look at dependences
- Add loop directives
  - arrays that need to be privatized

Summary

- Directives are an easy way to utilize the power of many-core processing
- Compiler and tools still rapidly evolving