Tool Chain For Co-Design

Created by Tom Scogland / tom.scogland@vt.edu
Part of the SyNeRG lab: synergy.cs.vt.edu
Why worry about the tool chain?
Lets look at a simple dot product.
Dot Product on CPU: C

double * dotP(double *a, double *b, size_t length){
    double result = 0.0;
    for(size_t i=0; i < length; i++){
        result += a[i] * b[i];
    }
    return result;
}
Dot Product on CPU: OpenMP

double * dotP(double *a, double *b, size_t length){
    double result = 0.0;
#pragma omp parallel for reduction(+:result)
    for(size_t i=0; i < length; i++){
        result += a[i] * b[i];
    }
    return result;
}
Dot Product on GPU: CUDA

```c
__global__ void dotP(double *g_a, double *g_b, int *g_odata) {
    extern __shared__ int sdata[];
    // each thread loads one element from global to shared mem
    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x * blockDim.x + threadIdx.x;
    sdata[tid] = g_a[i] * g_b[i];
    __syncthreads();
    // do reduction in shared memory
    for(unsigned int s=1; s < blockDim.x; s *= 2) {
        if (tid % (2*s) == 0) {
            sdata[tid] += sdata[tid + s];
        }
        __syncthreads();
    }
    // write result for this block to global mem
    if (tid == 0) g_odata[blockIdx.x] = sdata[0];
} // Must be run 2-3 times to produce a final result

(Based on an example from "Optimizing Parallel Reduction in CUDA" by Mark Harris)```
double * dotP(double *a, double *b, size_t length) {
    double result = 0.0;
    accel_dotProd_reduce(a, b, 
        {length, 0, 0}, {0, 0, 0}, {length, 0, 0}, 
        &result);
    return result;
}
Dot Product on GPU: OpenACC

double * dotP(double *a, double *b, size_t length){
    double result = 0.0;
    #pragma acc kernels for copyin(a[0:length],b[0:length]) reduction(+:result)
    for(size_t i=0; i < length; i++){
        result += a[i] * b[i];
    }
    return result;
}
double * dotP(double *a, double *b, size_t length) {
    double result = 0.0;
    #pragma omp parallel for reduction(+:result)
    #pragma acc kernels for copyin(a[0:length],b[0:length]) reduction(+:result)
    for (size_t i = 0; i < length; i++) {
        result += a[i] * b[i];
    }
    return result;
}
Advantages to OpenACC or OpenMP 4.0

- Little to no alteration of core code is required
- The CPU and GPU code are often the same
- Directives are portable, supporting CPU, GPU and potentially even FPGA devices given an appropriate compiler
What’s the Catch?
OpenACC is new and evolving rapidly
Limitations to OpenACC

1. No support for atomic operations
2. Minimal support for routine calls
3. No device debugging support
4. Lack of support for deep memory copies
5. No automatic work-sharing across devices
The Upside of Rapid Evolution

PGI 2014 Update Improvements

1. Atomics are accessible from Fortran OpenACC
2. Preliminary support for routine calls
3. CUDA Fortran and OpenACC device and host debugging
4. C-style 2d array copy support, a first step toward deep copies
Device Debugging Support

- Before PGI 2014
  - OpenACC debugging is host-only
  - Device debugging can only check outputs
- With PGI 2014 and Allinea DDT
  - Debugging on host and device with:
    - Breakpoints
    - Individual thread stepping
    - Memory watchpoints on all CUDA memory spaces
  - Device memory debugging
    - Inspect values and arrays in global and even shared memory
    - Catch and debug out-of-range accesses
Deep Copies

Several projects, notably Sensei, have encountered the lack of support for arrays where each element contains, or is, a dynamic array.
Deep Copies

C Example

double ** a2 = (double**)calloc(y_length, sizeof(double*));
double ** b2 = (double**)calloc(y_length, sizeof(double*));

for (int i = 0; i < y_length; i++) {
    a2[i] = calloc(x_length, sizeof(double));
    b2[i] = calloc(x_length, sizeof(double));
}

#pragma acc kernels for copyin(a2[0:y_length][0:x_length],b2[0:y_length][0:x_length]
for (int i = 0; i < y_length; i++) {
    for (int j = 0; j < x_length; j++) {
        sum += a2[i][j] * b2[i][j];
    }
}
}
Deep Copies
Performance Consequences

![Bar Chart]

- **Time (μs)**
  - Contiguous:
  - Deep copy:

The chart illustrates the performance consequences of deep copies, showing a significant increase in time compared to contiguous copies.
Deep Copies
Performance Consequences

Note that the Y axis was log10, Contiguous is 483 times faster
Co-Design in the Tool Chain

Collaborations with PGI and the OpenMP Accelerator Working Group
OpenACC Optimization with PGI
Sensei Lite
Unexpected Data Movement

- Data movement is one of the largest causes of overhead in heterogeneous applications
  - OpenACC reductions can be an unexpected source
  - Every region containing a reduction imposes a synchronous data copy of the reduction variable back to the host!
- Sensei Lite uses a reduction for almost all kernels
  - Error residuals detect algorithm convergence
  - Are they all necessary?

By testing for convergence less often, a full run can enqueue far more work between barriers.
Performance with Infrequent Reductions

![Graph showing performance with infrequent reductions. The x-axis represents the length of each matrix dimension (2d matrices) ranging from 256 to 4096, and the y-axis represents GigaFLOPs/s. There are different markers for Device (c2070, k20c) and Version (GPU 16x8, GPU Asynchronous).]
Targeting Multiple Architectures with OpenACC

- OpenACC can, in principle, support:
  - CPUs
  - GPUs: NVIDIA and AMD
  - Co-processors: Xeon Phi, Tile64
  - Etc.

In practice, a single binary normally supports just one, and that one is usually NVIDIA GPUs with the possible addition of the host CPU.
PGI OpenACC -> AMD Radeon OpenCL

In private beta until January 24th, 2013
PGI 14.1 marks the release of official support
AMD Radeon evaluation for CFD

Required:

• Re-targeting Sensei Lite to support multiple devices
• Adding support for selecting multiple device types
• Backing out non-portable optimizations:
  ▫ Custom gang and vector sizes
  ▫ Synchronization between iterations returns, to exchange boundary values
Multi-GPU and AMD

OpenMP 4.0

OpenACC is a target, and standard, of convenience.
OpenMP 4.0

OpenMP Accelerator directives are more likely to be long-lasting, but do not offer support for certain critical optimization tools.
OpenMP 4.0→4.1

We have been working directly with the OpenMP Accelerator working group on improved support for features critical to the performance of our designs including:

- Support for unstructured data lifetimes
- Asynchronous invocation of "target", or accelerator, regions
- Task-dependency resolution across both host and device tasks
- Work-sharing across devices
Summary

- The expressibility, stability and usability of the tool chain can have a significant effect on rate of progress
- Significant performance gains can be realized with directive-based programming models
- Neither OpenACC nor OpenMP 4.0 are perfect, but both can provide real advantages, and are improving quickly

Questions?