Hybrid OpenMP/MPI
with Cactus and Carpet

Erik Schnetter
CIGR Talk Series
Baton Rouge, LA, 2009-10-19
OpenMP Parallelisation: Quick Facts

- OpenMP can parallelise within one node only (requires shared memory)
- Saves memory (no ghost zones required); reduces cache pollution
- Can improve scaling (since fewer MPI processes for same number of cores)
- OpenMP directives are ignored by default (are safe to add to existing code)
- OpenMP is supported almost everywhere
Background: MPI Parallelisation

- Decompose domain, one subdomain for each process
- Introduces ghost zones, creating memory overhead
- Requires synchronising after modifying grid functions
OpenMP Parallelisation

- Threads share same memory, work on same arrays
- No ghost zones, no memory overhead
- No synchronisation required
- Usually, only loops are parallelised, remainder of programme remains sequential
Sample Calculation: Ghost Zone Memory Overhead

- Assume $20^3$ grid points per process, 3 ghost zones (4th order with advection)
- evolved points: $20^3 = 8,000$
- overall points: $(20+2 \cdot 3)^3 = 17,576$
- ghost zone overhead: 120% (factor 2.2)
- (Lesson: “3” is a large number if it is found in an exponent...)
Current State

- Most of Cactus, PUGH, Carpet parallelised via OpenMP (but not everything fully optimised yet)
- Note: Can parallelise incrementally by looking at timer output, working on slowest routines
- New codes (CTGamma, McLachlan, etc.) fully parallelised
- Hand-written Fortran codes (CCATIE, Whisky) not yet parallel (tedious!)
- “Serial thorns” in Einstein Toolkit (TwoPunctures, AHFinderDirect) partly parallelised
Benchmark (Scaling)

- Setup:
  - Carpet, McLachlan, 9 AMR levels
- $25^3$ per core, 3 ghost zones, weak scaling
- Infrastructure scales well (except regridding)
- Uses OpenMP to improve scalability
Improved Scaling via OpenMP

- Note: these are outdated weak scaling results, demonstrating how scaling breaks down
- Different #OpenMP threads:
  - Franklin: 1
  - Queen Bee: 8
  - Ranger: 4
- Scaling breakdown depends on #MPI processes, not on #cores
- Using N threads improve scaling by a factor of N

[Outdated results, March 2008]
Benchmark (Single Node)

Cactus Benchmark (using 1 node)

- Varying #cores used, #MPI processes, #OpenMP threads
- ideal scaling would be horizontal line
- using more cores reduces per-core performance
- using OpenMP changes performance

Ranger (varying NT)
Benchmark (shared memory vs. interconnect)

Cactus Benchmark (using 16 cores)

- Varying #nodes used, #MPI processes, #OpenMP threads

Ranger (varying NT)

- Ideal scaling would be horizontal line

- Using more nodes does not influence performance much

- Using OpenMP changes performance
Future Benchmark Work

- Previous slides examine only wall time
- Need more low-level information:
  - cycles, instructions, cache misses, memory bandwidth thread/MPI wait times, etc.
  - compare different architectures, compilers, build options (>30% unexplained difference between different systems)
- Given allocation shortages, 30% difference is huge
OpenMP Support in Tools

- **Kranc**: automated code generation
  <http://numrel.aei.mpg.de/Research/Kranc/>
  
  - Kranc generated code is fully parallelised with OpenMP

- **SimFactory**: simulation management
  <http://www.cct.lsu.edu/~eschnett/SimFactory/>
  
  - Cactus configurations built by SimFactory use OpenMP compiler options by default
  
  - Simulations started via SimFactory can use OpenMP easily (--num-threads=N)
LoopControl

- Generic mechanism to loop over grid functions, can replace nested for/do loops
- Automatically tiles loops (can improve cache efficiency)
- Automatically parallelises via OpenMP
- LoopControl keeps performance statistics, and can optimise its tiling/parallelisation parameters at run time
LoopControl Example

Original:

```c
#pragma omp parallel for
for (int k=1; k<cctk_lsh[2]-1; k++) {
    for (int j=1; j<cctk_lsh[1]-1; j++) {
        for (int i=1; i<cctk_lsh[0]-1; i++) {

            #pragma omp parallel
            LC_LOOP3 (wavetoy, i,j,k,
                      1,1,1,
                      cctk_lsh[0]-1,cctk_lsh[1]-1,cctk_lsh[2]-1,
                      cctk_lsh[0],cctk_lsh[1],cctk_lsh[2])

        }
    }
}
```

with LoopControl:

```c
#include <loopcontrol.h>
#pragma omp parallel
LC_LOOP3 (wavetoy, i,j,k, 
          1,1,1,  
          cctk_lsh[0]-1,cctk_lsh[1]-1,cctk_lsh[2]-1,  
          cctk_lsh[0],cctk_lsh[1],cctk_lsh[2])
```

- LC_LOOP3 macro hides complexity
- Perform loop optimisations (tiling, different OpenMP topologies)
- Could introduce other optimisations later, without changing macro calls
• With OpenMP, typically individual loops are parallelised, leaving other code unchanged

• Loops have OpenMP directives added, e.g.
  
  #pragma omp parallel for

• Need to use special compiler flag (e.g. -openmp) to enable directives (otherwise they are ignored)

• See <http://www.openmp.org/>; many tutorials on the web
OpenMP Concepts

To be parallelised, the individual iterations of a loop must be independent:

- the order of execution must not matter
- different iterations must not access the same variables

- Good examples: RHS evaluation, con2prim
- Not parallel: Gauss-Seidel iteration, performing I/O
OpenMP Fortran Example: Whisky, con2prim

```fortran
!$omp parallel do private (epsnegative, det,
   uxx,uxy,uxz,uyy,uyz,uzz, psi4pt, enthalpy)
do k = 1, nz
   do j = 1, ny
      do i = 1, nx
         !$omp critical
         call CCTK_WARN(1,'Con2Prim: stopping the code.')
         !$omp end critical

         good: only need to annotate 3D loops
         bad: need to list all temporary variables used in the loop
         C, C++: can declare variables inside loop (much simpler)
```

can do I/O in parallel loop if OpenMP is told about it
Private Variables, Reduction Operations

• If a loop uses temporary variables, they either need to be declared inside the loop, or need to be declared as private.

• In other words: you need to tell OpenMP about it, then you’re fine.

• Likewise, if a reduction (e.g. sum) is performed, OpenMP needs to be told.

• Some loops just cannot be parallelised; if you do, you may silently sometimes receive wrong results.