Supporting OpenMP and other Higher Languages in Dyninst

Nick Rutar
University of Maryland

Parallel Language Support for Dyninst

- OpenMP and other parallel languages are becoming more popular
- Advantageous to parse and instrument
- New languages on horizon
  - Want API to be extensible for adding languages
- Start with OpenMP
  - Unless otherise specified, talk will be OpenMP
- UPC, Titanium, Fortress, X10, Chapel planned for future

OpenMP Parallel & Work-Sharing Constructs

- Parallel
  - Main construct
- Do/for
  - Loop parallelism
- Sections
  - Non-iterative work sharing
- Single
  - Executed by only one thread in the team
- Combined Parallel & Work-Sharing
  - Parallel Do
  - Parallel Sections

OpenMP Synchronization Constructs

- Master
  - Only master thread operates on it
- Critical
  - Area of code executed by one thread at a time
- Barrier
  - All threads must reach point before execution continues
- Atomic
  - Specific memory location updated atomically
- Flush
  - Sync point that must have consistent view of memory
- Ordered
  - Iterations in loop will be executed in same order as serial
  - Has to be associated with a for directive

Parallel/Work Sharing Traits (Power)

- Sets up parallelism with
  - Call to _xlsmpParSelf
  - Register bookkeeping
    - Set up parameters for parallel behavior
  - Call to _xlsmp*_TPO
    - This call then calls parallel regions discussed below
- Actual parallel regions stored in function
  - Format
    - <CallingFunction>@OL@<Var++>
  - Parallel Functions(Regions) can call out
    - Nested Constructs, e.g. Parallel, for

Associated Setup Functions(Power)

- Parallel
  - _xlsmpParRegionSetup_TPO
- Do/for
  - _xlsmpWSDoSetup_TPO
- Sections
  - _xlsmpWSSectSetup_TPO
- Single
  - _xlsmpSingleSetup_TPO
- Parallel Do
  - _xlsmpParallelDoSetup_TPO
- Parallel Sections -
  - _xlsmpWSSectSetup_TPO

Synchronization Traits (Power)

- Master
  - Makes call to _xlsmpMaster_TPO
  - Checks to see if master thread
    - If so, explicitly calls a @OL function
- Critical
  - Calls _xlsmpFlush
  - Calls _xlsmpGetDefaultSLock
  - Performs operation (no @OL call)
  - Calls _xlsmpRelDefaultSLock
  - Calls _xlsmpFlush

Synchronization Traits (Power)

- Barrier
  - Calls _xlsmpBarrier_TPO
- Atomic
  - Calls _xlsmpGetAtomicLock
  - Performs operation(not an @OL call)
  - Calls _xlsmpRelAtomicLock
- Flush
  - Calls _xlsmpFlush
- Ordered
  - Calls _xlsmpBeginOrdered_TPO
  - Explicitly Calls @OL function to do operation
  - Calls _xlsmpEndOrdered_TPO

Instrumentable Regions

- Instrument entire function of @OL call
  - Entire region contained neatly within outlined function
  - Parallel, Do, Section, Single, Ordered, Master
- Instrument region
  - Make inst point immediately after given call
  - Store info about end of region
  - Critical, Ordered, Master, Atomic
- One instruction “region”
  - Flush & Barrier calls can be instrumented
  - Insert call to Flush or Barrier in an existing parallel region
- Loop Region
  - Region consists of the instructions in parallel loop body

Bpatch_parRegion

- New class to deal with parallel languages
- Standard region functions
  - getStartAddress()
  - getEndAddress()
  - size()
  - getInstructions()
- Generic Parallel Functions
  - getClause(const char * key)
- Language Specific Functions
  - replaceOMPParameter(const char * key, int value)

getClause

- Accesses information about parallel region
- Every region has at least Region_Type key
  - Enum for designating what region it is
    - enum{OMP_NONE, OMP_PARALLEL, OMP_DO_FOR, …}
    - Other language regions easily added
- Region Specific Keys
  - OMP_DO_FOR
    - CHUNK_SIZE
    - IF
    - NUM_ITERATIONS
    - ORDERED
    - SCHEDULE
- Documentation, API calls contain valid clauses

replaceOMPParameter

- OpenMP passes in parameters to setup functions that dictate behavior
  - Work Sharing Constructs
    - If
    - Nowait
    - Loops
      - Schedule Type
        - Static, dynamic, guided, runtime
      - Chunk Size
  - We can dynamically modify these values
  - Significantly change behavior without recompilation

Sample Code

```
/* Instrument first instruction in each OpenMP Section Construct */
BPatch_thread* appThread= bPatch.createProcess()
BPatch_image* appImage = appThread->getImage();
BPatch_Vector< BPatch_parRegion * > *appParRegions =
                                appImage->getParRegions();
for(int i = 0; i < appParRegions->size(); i++)
    {
      int regionType = (*appParRegions)[i]->getClause("REGION_TYPE");
      if (regionType != OMP_SECTIONS)
          continue;
     BPatch_Vector< BPatch_instruction *> *regionInstructions =
                            (*appParRegions)[i]->getInstructions();
       BPatch_instruction *bpInst = (*regionInstructions)[0];
       long unsigned int firstAdd = (long unsigned int)bpInst->getAddress();
      BPatch_point*point=appImage->createInstPointAtAddr ((caddr_t)firstAdd);
       appThread->insertSnippet( , *point, , ,);
      }
```

Current Status & Future Work

- Everything in talk implemented on
  - Power
  - Solaris
- Future Work
  - Additional platforms for OpenMP support
  - Additional Language support
    - UPC is next on list
  - Support for shared/private variables
    - Variables still handled as BPatch_[Local]Var
    - No distinction between shared or private

Demo

- OpenMP implementation of Life
  - Trivial nearest neighbor computation
- Ran on AIX, Power4 with 8 processors
- Implementation has chunk size of 1
- Dynamically change chunk size to 64
  - Approximately double speed-up for mutatee

Questions?