Asynchronous Collective Output With Non-Dedicated Cores

Phil Miller, Shen Li, Chao Mei
University of Illinois
2011-09-30
IASDS
Motivation

- Application developers know how to scale computation, communication
- Parallel storage is still obscure
- Opportunity: let it stay that way!
Approach

- Dynamic runtime and execution model
- Simple API
- Library as smart as it wants to be
- Resource allocation?
Charm++ Environment

- Framework for large-scale object-based parallel programs
- Portable across all major architectures
- Asynchronous remote method invocation to send messages
- Overdecomposition – many objects per core
Design

Parallel File System

Application Object

Parallel I/O Proxies

Processor
Implementation

- Forward data to selected processors for stripe-disjoint access
  - None of the fancy optimizations (yet)
- Modified NAMD to stand in for its parallel writes
Implementation

```c++
void Manager::write(Token token, const char *data,
                     size_t bytes, size_t offset) {
  Options &opts = files[token].opts;
  do {
    size_t stripe = offset / opts.peStripe;
    int pe = opts.basePE + stripe * opts.skipPEs;
    size_t bytesToSend = std::min(bytes, opts.peStripe -
                                   offset % opts.peStripe);
    thisProxy[pe].write_forwardData(token, data,
                                     bytesToSend, offset);
    data += bytesToSend;
    offset += bytesToSend;
    bytes -= bytesToSend;
  } while (bytes > 0);
}
```
void Manager::
write_forwardData(Token token, const char *data,
    size_t bytes, size_t offset) {
    // Error handling wrapper . . .
    {
        pwrite(files[token].fd, data, bytes, offset);
        // . . .
    }
}
Conclusions

- Get output off the critical path
- Shuffling data around needn't be hard, nor synchronous
- Porting to a new runtime, for easy output? Unlikely . . .
- Future: new MPI-IO for AMPI?

- Questions?