The Asynchronous Dynamic Load-Balancing Library

Rusty Lusk, Steve Pieper, Ralph Butler, Anthony Chan

Mathematics and Computer Science Division
Nuclear Physics Division
Argonne National Laboratory
Outline

- The Nuclear Physics problem - GFMC
- The CS problem: scalability, load balancing, simplicity for the application
- Approach: a portable, scalable library, implementing a simple programming model

Status
- Choices made
- Lessons learned
- Promising results

Plans
- Scaling up
- Branching out
- Using threads
The Physics Project: Green’s function Monte Carlo (GFMC)

- The benchmark for nuclei with 12 or fewer nucleons
- Starts with variational wave function containing non-central correlations
- Uses imaginary-time propagation to filter out excited-state contamination
  - Samples removed or multiplied during propagation -- work fluctuates
  - Local energies evaluated every 40 steps
  - For $^{12}$C expect ~10,000 Monte Carlo samples to be propagated for ~1000 steps
- Non ADLB version:
  - Several samples per node; work on one sample not parallelized
- ADLB version:
  - Three-body potential and propagator parallelized
  - Wave functions for kinetic energy and two-body potential done in parallel
  - Can use many processors per sample
- Physics target: Properties of both ground and excited states of $^{12}$C
The Computer Science Project: ADLB

- Specific Problem: to scale up GFMC, a master/slave code
- General Problem: scaling up the master/slave paradigm in general
  - Usually based on a single (or shared) data structure
- Goal for GFMC: scale to 160,000 processes (available on BG/P)
- General goal: provide simple yet scalable programming model for algorithms parallelized via master/slave structure
- General goal in GFMC setting: eliminate (most) MPI calls from GFMC and scale the general approach
- GFMC is not an easy case:
  - Multiple types of work
  - Any process can create work
  - Large work units (multi-megabyte)
  - Priorities and types used together to specify some sequencing without constraining parallelism ("workflow")
Master/Slave Algorithms and Load Balancing

- **Advantages**
  - Automatic load balancing

- **Disadvantages**
  - Scalability - master can become bottleneck

- **Wrinkles**
  - Slaves may create new work
  - Multiple work types and priorities that impose ordering
Load Balancing in GFMC Before ADLB

- Master/Slave algorithm
- Slaves do create work dynamically
- Newly created work stored locally
- Periodic load balancing
  - Done by master
  - Slaves communicate work queue lengths to master
  - Master determines reallocation of work
  - Master tells slaves to reallocate work
  - Slaves communicate work to one another
  - Computation continues with balanced work queues
GFMC Before ADLB
Zoomed in
BlueGene P

- 4 cpus per node
- 4 threads or processes per node
- 160,000 cpus
- Efficient MPI communication
- Original GFMC not expected to continue to scale past 2000 processors
  - More parallelism needed to exploit BG/P for Carbon 12
  - Existing load-balancing mechanism will not scale
- A general-purpose load-balancing library should
  - Enable GFMC to scale
  - Be of use to other codes as well
  - Simplify parallel programming of applications
The Vision

- No explicit master for load balancing; slaves make calls to ADLB library; those subroutines access local and remote data structures (remote ones via MPI).

- Simple Put/Get interface from application code to distributed work queue hides most MPI calls
  - Advantage: multiple applications may benefit
  - Wrinkle: variable-size work units, in Fortran, introduce some complexity in memory management

- Proactive load balancing in background
  - Advantage: application never delayed by search for work from other slaves
  - Wrinkle: scalable work-stealing algorithms not obvious
The API (Application Programming Interface)

- **Basic calls**
  - `ADLB_Init(num_servers, am_server, app_comm)`
  - `ADLB_Server()`
  - `ADLB_Put(type, priority, len, buf, answer_dest)`
  - `ADLB_Reserve(req_types, work_handle, work_len, work_type, work_prio, answer_dest)`
  - `ADLB_Ireserve(…)`
  - `ADLB_GetReserved(…)`
  - `ADLB_Set_done()`
  - `ADLB_Finalize()`

- **A few others, for tuning and debugging**
  - (still at experimental stage)
Asynchronous Dynamic Load Balancing - Thread Approach

- The basic idea:

  - Application Threads
  - ADLB Library Thread
  - Shared Memory
    - Put/get
    - Work queue
  - MPI Communication with other nodes
ADLB - Process Approach

Application Processes
ADLB Servers

put/get
Early Version of ADLB in GFMC on BG/L
History and Status

- Year 1: learned the application; worked out first version of API; did thread version of implementation
- Year 2: Switched to process version, began experiments at scale
  - On BG/L (4096), SiCortex (5800), and BG/P (16K so far)
  - Variational Monte Carlo (part of GFMC)
  - Full GFMC (see following performance graph)
  - Latest: GFMC for neutron drop at large scale
- Some additions to the API for increased scalability, memory management
- Internal changes to manage memory
- Still working on memory management for full GMFC with fine-grain parallelism, needed for $^{12}$C
- Basic API not changing
Comparing Speedup

![Graph comparing speedup with different configurations and number of processors.](image-url)
Most Recent Runs

- 14-neutron drop on 16,384 processors of BG/P
- Speedup of 13,634 (83% efficiency)
- No microparallization since more configurations
- ADLB processes 171 million work packages of size 129KB each, total of 20.5 terabytes of data moved
- Heretofore uncomputed level of accuracy for the computed energy and density
- Also some benchmarking runs for $^9$Be and $^7$Li
Future Plans

- Shortdetour into scalable debugging tools for understanding behavior, particularly memory usage
- Further microparallelization of GFMC using ADLB
- Scaling up on BG/P
- Revisit the thread model for ADLB implementation, particularly in anticipation of Q and experimental compute-node Linux on P
- Help with multithreading the application (locally parallel execution of work units via OpenMP)
- Work with others in the project who use manager/worker patterns in their codes
- Work with others outside the project in other SciDACs
Summary

- We have designed a simple programming model for a class of applications
- We are working on this in the context of a specific UNEDF application, which is a challenging one
- We have done two implementations
- Performance results are promising so far
- Still have not completely conquered the problem
- Needed: tools for understanding behavior, particularly to help with application-level debugging
The End