ADLB Update

Recent and Current Adventures with the Asynchronous Dynamic Load Balancing Library

Rusty Lusk

Mathematics and Computer Science Division
Argonne National Laboratory
Outline

- Review of what ADLB is
- Progress in the context of GFMC
- Recent applications other than GFMC
- Current implementation activities
Load Balancing

- **Definition:** the assignment (scheduling) of tasks (code + data) to processes so as to minimize the total idle times of processes

- **Static load balancing**
  - all tasks are known in advance and pre-assigned to processes
  - works well if all tasks take the same amount of time
  - requires no coordination process

- **Dynamic load balancing**
  - tasks are assigned to processes by coordinating process when processes become available
  - Requires communication between manager and worker processes
  - Tasks may create additional tasks
  - Tasks may be quite different from one another
Generic Master/Slave Algorithm

- Easily implemented in MPI
- Solves some problems
  - implements dynamic load balancing
  - termination detection (either preemptive or by exhaustion)
  - dynamic task creation
  - can implement workflow structure of tasks
- Scalability problems
  - Master can become a communication bottleneck (granularity dependent)
  - Memory can become a bottleneck (depends on task description size)
The ADLB Model (no master)

- Doesn’t really change algorithms in slaves
- Not a new idea (e.g. Linda)
- But need scalable, portable, distributed implementation of shared work queue
  - MPI complexity hidden here
API for a Simple Programming Model

- Basic calls
  - ADLB_Init( num_servers, am_server, app_comm)
  - ADLB_Server()
  - ADLB_Put( type, priority, len, buf, target_rank, answer_dest )
  - ADLB_Reserve( req_types, handle, len, type, prio, answer_dest)
  - ADLB_Ireserve( ... )
  - ADLB_Get_Reserved( handle, buffer )
  - ADLB_Set.Done()
  - ADLB_Finalize()

- A few others, for tuning and debugging
  - ADLB_{Begin,End}_Batch_Put()
  - Getting performance statistics with ADLB_Get_info(key)
API Notes

- Return codes (defined constants)
  - ADLB_SUCCESS
  - ADLB_NO_MORE_WORK
  - ADLB_DONE_BY_EXHAUSTION
  - ADLB_NO_CURRENT_WORK (for ADLB_Ireserve)

- Batch puts are for inserting work units that share a large proportion of their data

- Types, answer_rank, target_rank can be used to implement some common patterns
  - Sending a message
  - Decomposing a task into subtasks
  - Maybe should be built into API
How It Works (Current production version)

- Application Processes
- ADLB Servers

put/get
Progress with GFMC

Efficiency = compute\_time/wall\_time – 25 Feb 2010

Number of nodes (4 OpenMP cores per node)

Efficiency in %

Feb 2009

Oct 2009

Jun 2009

$^{12}\text{C ADLB+GFMC}$
Another Physics Application - Parameter Sweep

- Luminescent solar concentrators
  - Stationary, no moving parts
  - Operate efficiently under diffuse light conditions (northern climates)
- Inexpensive collector, concentrate light on high-performance solar cell
- In this case, the authors never learned any parallel programming approach before ADLB (ADLB as high-level programming model)
Two Other Applications

- **The “Batcher”**
  - Simple but potentially useful
  - Input is a file of Unix command lines
  - ADLB worker processes execute each one with the Unix “system” call

- **Swift substrate**
  - Swift is a high-level workflow description language
  - ADLB is being tested as an execution engine for Swift programs
  - Fine granularity needed
Alternate Implementations of the Same API

- Single server with one-sided communication among clients
  - Motivation:
    - Eliminate multiple views of “shared” queue data structure and the effort required to keep them (almost) coherent
    - Free up more processors for application calculations by eliminating most servers.
    - Use larger client memory to store work packages
  - Relies on “passive target” MPI-2 remote memory operations
  - Single master proved to be a scalability bottleneck at 32,000 processors (8K nodes on BG/P) not because of processing capability but because of network congestion.
  - Have not yet experimented with hybrid version (1-sided, multiple-server)

- Completely symmetric (“no server”) threaded version
  - ADLB code runs in separate thread on each node.
Asynchronous Dynamic Load Balancing

- The basic idea:

  Application Threads

  ADLB Library Thread

  Put/get

  Shared Memory

  Work queue

  MPI Communication with other nodes
Asynchronous Dynamic Load Balancing

- The basic idea:

  - Application Threads
  - ADLB Library Thread
  - Shared Memory
    - Work queue
  - MPI Communication with other nodes
  - Put/get
Preliminary Experiments with the Threaded Version

- Two kinds of experiments
  - 0-size and 0-length work units to test minimal overheads
  - random size and length of work units to test load balancing

![Graphs showing percent work units processed and hwm as percent of max malloc over ranks and number of ranks.](attachment:graphs.png)

![Bar chart showing total work time for 0.1 to 5.0 Secs.](attachment:bar_chart.png)
Getting ADLB

- Web site is  
  http://www.cs.mtsu.edu/~rbutler/adlb

- To download adlb:
  - svn co http://svn.cs.mtsu.edu/svn/adlbm/trunk adlbm

- What you get:
  - source code (multiple versions)
  - configure script and Makefile
  - README, with API documentation
  - Examples
    - Sudoku
    - Batcher
      - Batcher README
    - Traveling Salesman Problem

- To run your application
  - configure, make to build ADLB library
  - Compile your application with mpicc, use Makefile as example
  - Run with mpiexec

- Problems/complaints/kudos to {lusk,rbutler}@mcs.anl.gov
Conclusions

- The Philosophical Accomplishment: Scalability need not come at the expense of complexity

- The Practical Accomplishment: Multiple uses
  - As high-level library to make simple applications scalable
  - As execution engine for
    - complicated applications (like GFMC)
    - higher-level “many-task” programming models
The End