Advanced Parallel Programming

Is there life beyond MPI?
Outline

- MPI vs. High Level Languages
- Declarative Languages
- Map Reduce and Hadoop
- Shared Global Address Space Languages
- Charm++
- ChaNGa
- ChaNGa on GPUs
Parallel Programming in MPI

- Good performance
- Highly portable: de facto standard
- Poor match to some architectures
  - Active Messages, Shared Memory
- New machines are hybrid architectures
  - Multicore, Vector, RDMA, Cell
- Parallel Assembly?
Parallel Programming in High Level Languages

- Abstraction allows easy expression of new algorithms
- Low level architecture is hidden (or abstracted)
- Integrated debugging/performance tools
- Sometimes a poor mapping of algorithm onto the language
- Steep learning curve
Parallel Programming Hierarchy

- Decomposition of computation into parallel components
  - Parallelizing compiler, Chapel
- Mapping of components to processors
  - Charm++
- Scheduling of components
  - OpenMP, HPF
- Expressing the above in data movement and thread execution
  - MPI
Language Requirements

• General Purpose
• Expressive for application domain
  • Including matching representations: *(a + i) vs a[i]
• High Level
• Efficiency/obvious cost model
• Modularity and Reusability
  • Context independent libraries
  • Similar to/interoperable with existing languages
Declarative Languages

- SQL example:
  
  ```sql
  SELECT SUM(L_Bol) FROM stars WHERE tform > 12.0
  ```

- Performance through abstraction

- Limited expressivity, otherwise
  - Complicated
  - Slow (UDF)
Map Reduce & Hadoop

- Map: function produces (key, value) pairs
- Reduce: collects Map output
- Pig: SQL-like query language
- Effective data reduction framework
- Not suitable for HPC
Array Languages, e.g., CAF

- Arrays distributed across images
- Each processor can access data on other processors via co-array syntax
  
  ```
  call sync_all(/up, down/)
  new_A(1:ncol) = new_A(1:ncol)
  +A(1:ncol)[up] + A(1:ncol)[down]
  call sync_all(/up, down/)
  ```

- Easy expression of array model
- Cost transparent
**Programmer:** [Over] decomposition into virtual processors

**Runtime:** Assigns VPs to processors

Enables *adaptive runtime strategies*

---

**System implementation**

**Benefits**

- **Software engineering**
  - Number of virtual processors can be independently controlled
  - Separate VPs for different modules

- **Message driven execution**
  - Adaptive overlap of communication

- **Dynamic mapping**
  - Heterogeneous clusters
    - Vacate, adjust to speed, share
  - Automatic checkpointing
  - Change set of processors used
  - Automatic dynamic load balancing
  - Communication optimization

**User View**
Chare A
void entryMethod_1()
    doSomeWork();
    MyMessage msg = new MyMessage();
    B.entryMethod_2(msg); // returns immediately
doMoreWork();
} 
void entryMethod_3(int var1, float var2) {...

Chare B
void entryMethod_2(MyMessage *msg) {
    delete msg;
    int myInt = 4;
    float myFloat = 3.14f;
    A.entryMethod_3(myInt, myFloat);
}

User view
Gravity Implementations

- Standard Tree-code
- “Send”: distribute particles to tree nodes as the walk proceeds.
  - Naturally expressed in Charm++
  - Extremely communication intensive
- “Cache”: request treenodes from off processor as they are needed.
  - More complicated programming
  - “Cache” is now part of the language
ChaNGa Features

- Tree-based gravity solver
- High order multipole expansion
- Periodic boundaries (if needed)
- SPH: (Gasoline compatible)
- Individual multiple timesteps
- Dynamic load balancing with choice of strategies
- Checkpointing (via migration to disk)
- Visualization
Cosmological Comparisons:
Mass Function

Heitmann, et al. 2005
Overall structure
Remote/local latency hiding

Clustered data on 1,024 BlueGene/L processors

Remote data work

Local data work
Load balancing with GreedyLB

Zoom In 5M on 1,024 BlueGene/L processors

5.6s

6.1s

4x messages
Load balancing with OrbRefineLB

Zoom in 5M on 1,024 BlueGene/L processors

Time Profile Graph

5.6s

5.0s
Scaling with load balancing

Number of Processors x Execution Time per Iteration (s)

- lambda
- lambda-LB
- dwarf-50M
- dwarf-50M-LB
Cosmo Loadbalancer

- Use Charm++ measurement based load balancer
- Modification: provide LB database with information about timestepping.
  - “Large timestep”: balance based on previous Large step
  - “Small step” balance based on previous small step
Results on 3 rung example

613s

429s

228s
Multistep Scaling
SPH Scaling

The diagram shows the performance scalability of different gravitational and density calculations using ChaNGa and PKDGRAV across various processor counts. The y-axis represents the number of particles processed per second per processor, while the x-axis represents the number of processors. The lines correspond to different calculations:

- **ChaNGa Gravity**
- **ChaNGa Density**
- **ChaNGa Density New**
- **PKDGRAV Gravity**
- **PKDGRAV Density**
- **PKDGRAV LB Density**
ChaNGa on GPU clusters

- Immense computational power
- Feeding the monster is a problem
- Charm++ GPU Manager
  - User submits work requests with callback
  - System transfers memory, executes, returns via callback
  - GPU operates asynchronously
  - Pipelined execution
Execution of Work Requests

- wr 0: memory transfer to device
- wr 1: kernel execution
- wr 2: memory transfer from device

Time
GPU Scaling

ChaNGa Overhead (lambs)

![Graph showing GPU scaling with iteration time (s) on the y-axis and number of CPUs on the x-axis.]
GPU optimization

Bucket Size vs. Execution Time on GPU

Time (s)

Particles per bucket

Total
Gravity
Traversal

Optimal = 32.51
Summary

- Successfully created highly scalable code in HLL
  - Computation/communication overlap
  - Object migration for LB and Checkpoints
  - Method prioritization
  - GPU Manager framework
- HLL not a silver bullet
  - Communication needs to be considered
  - “Productivity” unclear
    - Real Programmers write Fortran in any language
Availability

- Charm++: http://charm.cs.uiuc.edu
- ChaNGa download: http://software.astro.washington.edu/nchilada/
- Mailing list: changa-users@u.washington.edu