Charm++
Interoperability

Nikhil Jain
Charm Workshop - 2013
Motivation

- Charm++ RTS is powerful - message driven, optimized communication layer, load balancing, fault tolerance, power management, partitioning.

- But legacy codes are huge - rewriting them to use Charm++ may be significant work.

- Can one use Charm++ without code changes or partially to
  - Get concrete evidence of performance benefits for an application.
  - Improve performance of a few kernels.
  - Chunk by chunk transition to Charm++.
Proposed Paths

- For OpenMP
  - Charm++ is not a new language - direct use of existing code.

- For MPI applications
  - Use Adaptive MPI.
  - Interoperate Charm++ with MPI.

- Others - we implement front-end APIs as need arise.
Approach 1 - Adaptive MPI

- Charm++’s implementation of MPI
  - with useful additions.
  - Over-decomposition infused by treating each MPI rank as a virtual process (VP) that executes in its own user-level thread.
  - Each core hosts multiple VPs that are treated as chares of a chare array with scheduling controlled by Charm++ RTS.
AMPI: User and System View

MPI ranks shares that run in user-level threads

Real Processors
AMPI: Augmentations

- Additional functions-
  - MPI_Migrate - perform load balancing.
  - MPI_Checkpoint - checkpoint to disk.
  - MPI_MemCheckpoint - checkpoint to memory.
  - Non-blocking collectives - also in MPI-3 standard.
  - Isomalloc - automated tracking of user data for migration/checkpointing.
  - Swapglobals - automated handling if global data exists.
AMPI: Applications

- Our aim is to enable execution of any MPI code as AMPI

- Some Examples:
  - BRAMS - Brazilian Weather code based on RAMS
  - ISAM - Integrated Science Assessment Model for assessment of climate change
  - NAS Parallel Benchmarks
  - Mantevo Benchmarks
  - Lulesh
AMPI: BRAMS

Profile of Usage for Processors 0-63
(Time 2383670.0 ~ 2430251.0 ms)
AMPI: BRAMS

Profile of Usage for Processors 0-63
(Time 2383670.0 ~ 2430251.0 ms)

Profile of Usage for Processors 0-63
(Time 2009374.0 ~ 2041758.0 ms)
AMPI: BRAMS

Profile of Usage for Processors 0-63
(Time 2383670.0 ~ 2430251.0 ms)

Profile of Usage for Processors 0-63
(Time 1866746.0 ~ 1893523.0 ms)
Uniform distribution of non-zero across rows

<table>
<thead>
<tr>
<th>Numer of cores</th>
<th>Time per step</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>0.50</td>
</tr>
<tr>
<td>256</td>
<td>0.50</td>
</tr>
<tr>
<td>512</td>
<td>0.50</td>
</tr>
<tr>
<td>1024</td>
<td>0.50</td>
</tr>
</tbody>
</table>

- Blue bars: MPI
- Red bars: AMPI
Non-uniform distribution of non-zeros across rows

Time per step vs Number of Cores for MPI and AMPI.
AMPI: Work in Progress

- Improved efficiency - newer algorithms.
- Optimized support on IBM Blue Gene/Q
- No support for mmap - no isomalloc.
- Swapping globals.
Approach 2 - Interoperability

- Chunk by chunk transition to Charm++.
- Identify kernels that are better suited to Charm++.
- Implement them in Charm++.
- Make *calls to Charm++ code from MPI* based code.
Interoperability

- Charm++ resides in the same memory space as the MPI based code.
- Performs necessary low level initializations and resource procurement.
- Pass memory locations - no messaging required.
- Control transfer between Charm++ and the MPI based code analogous to the control transfer between the MPI based code and any other external library such as ParMETIS, FFTW etc.
Interoperability: Modes
Interoperability: Modes

(a) Time Sharing

- MPI Control
- Charm++ Control

Time

P(1) P(2) P(N-1) P(N)
Interoperability: Modes

(a) Time Sharing

(b) Space Sharing

- MPI Control
- Charm++ Control

Time

P(1) P(2) P(N-1) P(N)
Interoperability: Modes

(a) Time Sharing
- MPI Control
- Charm++ Control

(b) Space Sharing

(c) Combined Sharing

Time

P(1) P(2) P(N-1) P(N)
Interoperability: Charm++ Code

- Include mpi-interoperate.h.
- Add an interface function callable from the main program.

```c
void HelloStart(int elems)
{
    if(CkMyPe() == 0) {
        CProxy_MainHello mainhello =
            CProxy_MainHello::ckNew(elems);
    }
    StartCharmScheduler();
}
```
Interoperability: Code Flow

- Begin execution at user main.
- Perform MPI initialization and application initialization.
- Create a sub-communicator for Charm++.
- Initialize Charm++ with this sub-communicator.
- for (as many times needed)
  - perform MPI based communication and application work.
  - invoke Charm++ code.
- Exit Charm++.
Interoperability: Example

MPI_Init(argc,argv); //initialize MPI
//Do MPI related work here

//create comm to be used by Charm++
MPI_Comm_split(MPI_COMM_WORLD, myRank % 2, myRank, newComm);
CharmLibInit(newComm,.) //initialize Charm++ over my communicator

if(myRank % 2)
    StartHello(); //invoke Charm++ library on one set
else
    //do MPI work on other set

kNeighbor(); //invoke Charm++ library on both sets
CharmLibExit(); //destroy Charm++
Interoperability: Use cases

- Demonstrated in HPC Challenge submission with FFT benchmark.
- High performance sorting library based on
  - Highly Scalable Parallel Sorting by Edgar Solomonik and Laxmikant Kale (IPDPS, 2009).
- Efficient collision detection library based on
  - A Voxel based Parallel Collision Detection Algorithm by Orion Lawlor and Laxmikant Kale (ICS, 2002).
Interoperability: Work in Progress

- Enable space and combined sharing on non-MPI layers such as PAMI, uGNI.
- Development of interoperable libraries in Charm++
  - Graph algorithms - BFS, Spanning tree, Shortest path etc.
  - Efficient solvers.
- Integrate performance analysis of interoperable code using Projections.
Questions