CS4961 Parallel Programming

Lecture 17: Message Passing, cont. Introduction to CUDA

Mary Hall
November 3, 2009

Administrative

- Homework assignment 3
- Due, Thursday, November 5 before class
  - Use the "handin" program on the CADE machines
  - Use the following command:

    "handin cs4961 hw3 <gzipped tar file>"

OMIT VTUNE PORTION but do Problem 3 as homework. CADE Lab working on VTUNE installation. You can turn it in later for EXTRA CREDIT!

- Mailing list set up: cs4961@list.eng.utah.edu

A Few Words About Final Project

- Purpose:
  - A chance to dig in deeper into a parallel programming model and explore concepts.
  - Present results to work on communication of technical ideas
- Write a non-trivial parallel program that combines two parallel programming languages/models. In some cases, just do two separate implementations.
  - OpenMP + SSE-3
  - TBB + SSE-3
  - MPI + OpenMP
  - MPI + SSE-3
  - MPI + CUDA
  - Open CL??? (keep it simple! need backup plan)
- Present results in a poster session on the last day of class

Example Projects

- Look in the textbook or on-line
  - Recall Red/Blue from Ch. 4
    - Implement in MPI (+ SSE-3)
    - Implement main computation in CUDA
  - Algorithms from Ch. 5
  - SOR from Ch. 7
    - CUDA implementation?
  - FFT from Ch. 10
  - Jacobi from Ch. 10
  - Graph algorithms
  - Image and signal processing algorithms
  - Other domains…

Next Thursday, November 12

- Turn in <1 page project proposal
  - Algorithm to be implemented
  - Programming model(s)
  - Validation and measurement plan

Today's Lecture

- More Message Passing, largely for distributed memory
- Message Passing Interface (MPI): a Local View language
- Sources for this lecture
  - Larry Snyder, http://www.cs.washington.edu/education/courses/524/08wi/
  - Online MPI tutorial http://www-unix.mcs.anl.gov/mpi/tutorial/gropp/talk.html
  - Vivek Sarkar, Rice University, COMP 422, F08 http://www.cs.rice.edu/~vs3/comp422/lecture-notes/index.html

MPI Critique from Last Time (Snyder)

- Message passing is a very simple model
- Extremely low level; heavy weight
  - Expense comes from λ and lots of local code
  - Communication code is often more than half
  - Tough to make adaptable and flexible
  - Tough to get right and know it
  - Tough to make perform in some (Snyder says most) cases
- Programming model of choice for scalability
- Widespread adoption due to portability, although not completely true in practice

Today's MPI Focus

- Blocking communication
  - Overhead
  - Deadlock?
- Non-blocking
- One-sided communication

MPI-1

- MPI is a message-passing library interface standard.
  - Specification, not implementation
  - Library, not a language
  - Classical message-passing programming model
- MPI was defined (1994) by a broadly-based group of parallel computer vendors, computer scientists, and applications developers.
  - 2-year intensive process
- Implementations appeared quickly and now MPI is taken for granted as vendor-supported software on any parallel machine.
- Free, portable implementations exist for clusters and other environments (MPICH2, Open MPI)

MPI-2

- Same process of definition by MPI Forum
- MPI-2 is an extension of MPI – Extends the message-passing *model.*
- *Parallel I/O*
- *Remote memory operations (one-sided)*
- *Dynamic process management*
  - Adds other functionality
  - C++ and Fortran 90 bindings
  - similar to original C and Fortran-77 bindings
  - External interfaces
  - Language interoperability
  - MPI interaction with threads

Non-Buffered vs. Buffered Sends

- A simple method for forcing send/receive semantics is for the send operation to return only when it is safe to do so.
- In the non-buffered blocking send, the operation does not return until the matching receive has been encountered at the receiving process.
- Idling and deadlocks are major issues with non-buffered blocking sends.
- In buffered blocking sends, the sender simply copies the data into the designated buffer and returns after the copy operation has been completed. The data is copied at a buffer at the receiving end as well.
- Buffering alleviates idling at the expense of copying overheads.

Non-Blocking Communication

- The programmer must ensure semantics of the send and receive.
- This class of non-blocking protocols returns from the send or receive operation before it is semantically safe to do so.
- Non-blocking operations are generally accompanied by a check-status operation.
- When used correctly, these primitives are capable of overlapping communication overheads with useful computations.
- Message passing libraries typically provide both blocking and non-blocking primitives.

Deadlock?

```
int a[10], b[10], myrank;
MPI_Status status; ...
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

if (myrank == 0) {
    MPI_Send(a, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
   MPI_Send(b, 10, MPI_INT, 1, 2, MPI_COMM_WORLD); }

else if (myrank == 1) {
    MPI_Recv(b, 10, MPI_INT, 0, 2, MPI_COMM_WORLD);
    MPI_Recv(a, 10, MPI_INT, 0, 1, MPI_COMM_WORLD);
}
...
```


Deadlock?

Consider the following piece of code, in which process i sends a message to process *i + 1 (modulo the number of processes) and receives a message from process i - 1 (module the number of processes).*

```
int a[10], b[10], npes, myrank;
MPI_Status status; ...
MPI_Comm_size(MPI_COMM_WORLD, &npes);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Send(a, 10, MPI_INT, (myrank+1)%npes, 1,
  MPI_COMM_WORLD);
MPI_Recv(b, 10, MPI_INT, (myrank-1+npes)%npes, 1,
  MPI_COMM_WORLD); ...
```


Non-Blocking Communication

- To overlap communication with computation, MPI provides a pair of functions for performing non-blocking send and receive operations ("I" stands for "Immediate"):

  int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)

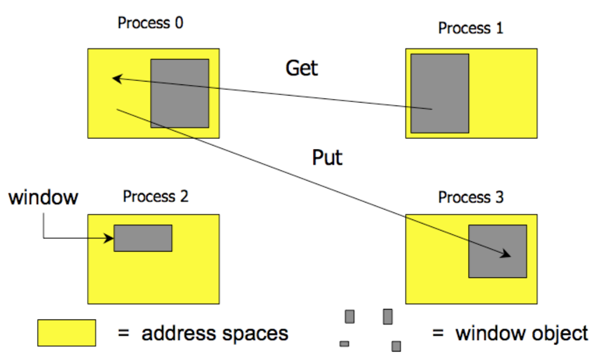
  int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

  These operations return before the operations have been completed.
- Function MPI_Test tests whether or not the non- blocking send or receive operation identified by its request has finished.

  int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
- MPI_Wait waits for the operation to complete.

  int MPI_Wait(MPI_Request *request, MPI_Status *status)

One-Sided Communication



MPI One-Sided Communication or Remote Memory Access (RMA)

- Goals of MPI-2 RMA Design
  - Balancing efficiency and portability across a wide class of architectures
  - shared-memory multiprocessors
  - NUMA architectures
  - distributed-memory MPP's, clusters
  - Workstation networks
- Retaining "look and feel" of MPI-1
- Dealing with subtle memory behavior issues: cache coherence, sequential consistency

MPI Constructs supporting One-Sided Communication (RMA)

- **MPI_Win_create exposes local memory to RMA operation by other processes in a communicator**
  - **Collective operation**
  - **Creates window object**
- **MPI_Win_free deallocates window object**
- **MPI_Put moves data from local memory to remote memory**
- **MPI_Get retrieves data from remote memory into local memory**
- **MPI_Accumulate updates remote memory using local values**