High-Performance Scientific Computing
Lecture 6: MPI

MATH-GA 2011 / CSCI-GA 2945 · October 10, 2012

Today

Tool of the day: gdb

MPI: Point-to-Point

Bits and pieces

- HW2: ...
- HW4: due today
- HW5: out tomorrow
- On HW5: 5 minute project pitch $\rightarrow$ due next week!
- Project: form teams

Atomic: Compare-and-swap

```
int atomic_cmpxchg (__global int *p, int cmp, int val)
int atomic_cmpxchg (__local int *p, int cmp, int val)
```

Does:

- Read the 32-bit value (referred to as old) stored at location pointed to by p.
- Compute `(old == cmp) ? val : old`.
- Store result at location pointed to by p.
- Returns old.

Atomic: Compare-and-swap

```
int atomic_cmpxchg (__global int *p, int cmp, int val)
int atomic_cmpxchg (__local int *p, int cmp, int val)
```

Does:

- Read the 32-bit value (referred to as old) stored at location pointed to by p.
- Compute `(old == cmp) ? val : old`.
- Store result at location pointed to by p.
- Returns old.

Implement atomic `float` add?

Outline

gdb

Demo time

Outline

MPI

Demo time

Send definition

MPI 3.0, Section 3.4:

> [`MPI_Send`] is blocking: it does not return until the message data and envelope have been safely stored away so that the sender is free to modify the send buffer.

Send definition

MPI 3.0, Section 3.4:

> [`MPI_Send`] is blocking: it does not return until the message data and envelope have been safely stored away so that the sender is free to modify the send buffer.

Send definition

MPI 3.0, Section 3.4:

> [`MPI_Send`] is blocking: it does not return until the message data and envelope have been safely stored away so that the sender is free to modify the send buffer.

Send definition

MPI 3.0, Section 3.4, more:

> [`MPI_Send`] uses the standard communication mode. In this mode, it is up to MPI to decide whether outgoing messages will be buffered.
>
> MPI may buffer outgoing messages. In such a case, the send call may complete before a matching receive is invoked. On the other hand, MPI may choose not to buffer outgoing messages.

Send definition

MPI 3.0, Section 3.4, more:

> [`MPI_Send`] uses the standard communication mode. In this mode, it is up to MPI to decide whether outgoing messages will be buffered.
>
> MPI may buffer outgoing messages. In such a case, the send call may complete before a matching receive is invoked. On the other hand, MPI may choose not to buffer outgoing messages.

Send definition

MPI 3.0, Section 3.4, more:

> [`MPI_Send`] uses the standard communication mode. In this mode, it is up to MPI to decide whether outgoing messages will be buffered.
>
> MPI may buffer outgoing messages. In such a case, the send call may complete before a matching receive is invoked. On the other hand, MPI may choose not to buffer outgoing messages.

Send definition

MPI 3.0, Section 3.4, more:

> [MPI_Send] uses the standard communication mode. In this mode, it is up to MPI to decide whether outgoing messages will be buffered.
>
> MPI may buffer outgoing messages. In such a case, the send call may complete before a matching receive is invoked. On the other hand, MPI may choose not to
> sages.

What is correct behavior?

Must, should, may (RFC 2119)

Send definition

MPI 3.0, Section 3.4, more:

> [MPI_Send] uses the standard communication mode. In this mode, it is up to MPI to decide whether outgoing messages will be buffered.
>
> MPI may buffer outgoing messages. In such a case, the send call may complete before a matching receive is invoked. On the other hand, MPI
> sages.

What is correct behavior?

Must, should, may (RFC 2119)

Alternative communication modes:

- Buffered
- Synchronous
- Ready

Send definition

MPI 3.0, Section 3.4, yet more:

> A send in standard mode can be started whether or not a matching receive has been posted.
>
> It may complete before a matching receive is posted.
>
> The standard mode send is non-local: successful completion of the send operation may depend on the occurrence of a matching receive.

Send definition

MPI 3.0, Section 3.4, yet more:

A send in standard mode can be started whether or not a matching receive has been posted.

It may complete before a matching receive is posted.

The standard mode send is non-local: successful completion of the send operation may depend on the occurrence of a matching receive.

Send definition

MPI 3.0, Section 3.4, yet more:

A send in standard mode can be started whether or not a matching receive has been posted.

It may complete before a matching receive is posted.

The standard mode send is non-local: successful completion of the send operation may depend on the occurrence of a matching receive.

Send definition

MPI 3.0, Section 3.4, yet more:

> A send in standard mode can be started whether or not a matching receive has been posted.
>
> It may complete before a matching receive is posted.
>
> The standard mode send is non-local: successful completion of the send operation may depend on the occurrence of a matching receive.

Send definition

MPI 3.0, Section 3.4, yet more:

> A send in standard mode can be started whether or not a matching receive has been posted.
>
> It may complete before a matching receive is posted.
>
> The standard mode send is non-local : successful completion of the send operation may depend on the occurrence of a matching receive.

Send definition

MPI 3.0, Section 3.4, yet more:

> A send in standard mode can be started whether or not a matching receive has been posted.
>
> It may complete before a matching receive is posted.
>
> The standard mode send is non-local: successful completion of the send operation may depend on the occurrence of a matching receive.

Meta-lesson

Can learn a lot from *how* something is said.

Lessons

- Blocking $\leftrightarrow$ buffers
- Communication modes
- Operation life cycle
- Matching
- Non-locality

Removing the deadlock

Two ways laid out:

Removing the deadlock

Two ways laid out:

- Use buffered send (brittle!)
- Change order (not always easy! Example?)

Removing the deadlock

Two ways laid out:

- Use buffered send (brittle!)
- Change order (not always easy! Example?)

Would like a middle ground:

"Just keep the buffer I've got right here!"

But when is it safe to reuse that buffer?

Non-blocking

MPI 3.0, Section 3.5:

> Nonblocking message-passing operations [...] can be used to avoid the need for buffering outgoing messages.

Additional Advantage: **[Sec. 3.7]**

> One can improve performance on many systems by overlapping communication and computation.

Non-blocking

MPI 3.0, Section 3.5:

> Nonblocking message-passing operations [...] can be used to avoid the need for buffering outgoing messages.

Additional Advantage: **[Sec. 3.7]**

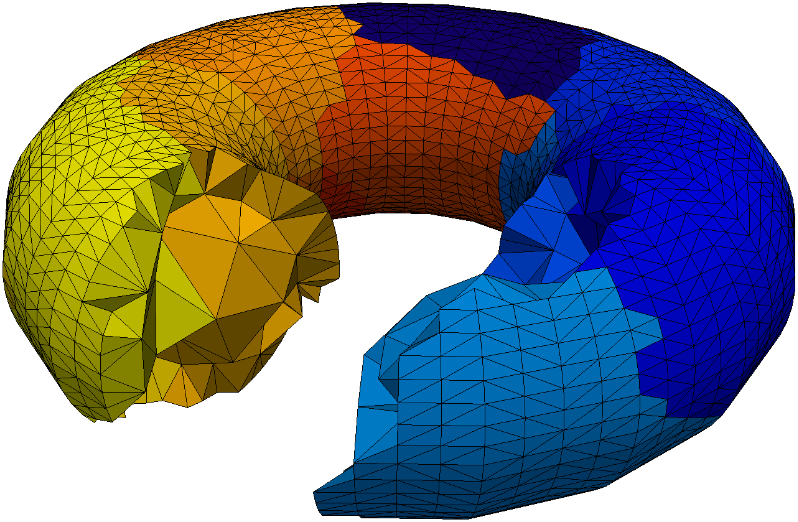
> One can improve pe
> overlapping commu

Nonblocking can be *combined* with buffered/ready/synchronous.
$\rightarrow$ It's not a "mode".

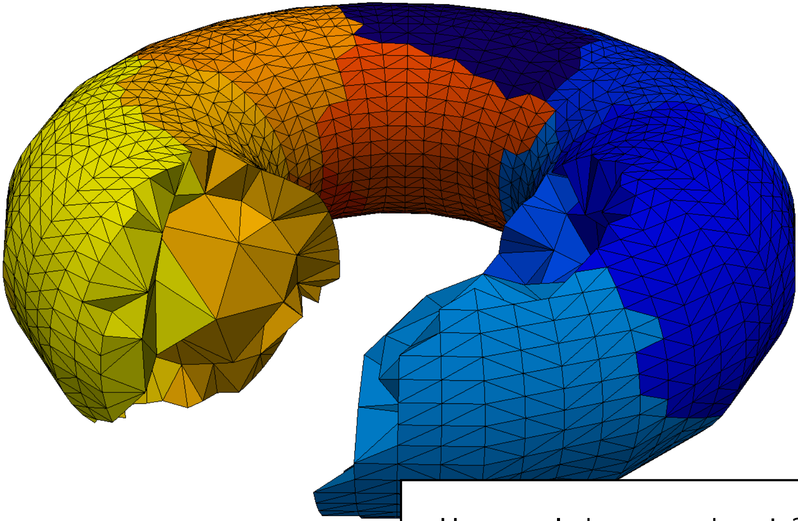
Nonblocking sends can be matched with blocking receives, and vice-versa. **[3.7]**

Nonblocking demo time

Partitioning for neighbor communication



Partitioning for neighbor communication



How can I chop up a domain?

MPI

Neighbor comm demo time

MPI: Ordering

MPI 3.0, Section 3.5:

> **Order** Messages are non-overtaking : If a sender sends two messages in succession to the same destination, and both match the same receive, then this operation cannot receive the second message if the first one is still pending.
>
> If a receiver posts two receives in succession, and both match the same message, then the second receive operation cannot be satisfied by this message, if the first one is still pending.

MPI: Ordering

MPI 3.0, Section 3.5:

Order Messages are non-overtaking: If a sender sends two messages in succession to the same destination, and both match the same receive, then this operation cannot receive the second message if the first one is still pending.

If a receiver posts two receives in succession, and both match the same message, then the second receive operation cannot be satisfied by this message, if the first one is still pending.

MPI: More on Ordering

Possible problem?

```
if (rank == 0)
{
  MPI_Bsend(buf1, count, MPI_DOUBLE, 1, tag1, comm)
  MPI_Ssend(buf2, count, MPI_DOUBLE, 1, tag2, comm)
}
else if (rank == 1) then
{
  MPI_Recv(buf1, count, MPI_DOUBLE, 0, tag2, comm, status)
  MPI_Recv(buf2, count, MPI_DOUBLE, 0, tag1, comm, status)
}
```

Questions?

?