Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

XJCO3221 Parallel Computation

Peter Jimack

University of Leeds

Lecture 12: Non-blocking communication

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Previous lectures
Today's lecture

Previous lectures

So far we have only considered **blocking communication** in distributed memory systems:

- Do not return until it is safe to use the resources, *i.e.* the memory allocated for the data.
- This *may* happen after the data is copied to a **buffer**.
- If the buffer is too small, will wait until it is being **received**.
- **Point-to-point communication**: MPI_Send(), MPI_Recv().
- **Collective communication**: MPI_Bcast(), MPI_Gather(), MPI_Scatter(), MPI_Reduce().

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Previous lectures
Today's lecture

Today's lecture

Today we will look at **non-blocking communication**:

- Non-blocking calls return **'immediately'**.

- Require **extra coding** to determine when it is safe to re-use the resources.

- Can overlap communication with computation to improve performance: **Latency hiding**.

- Useful in situations that require **domain partitioning**.

- Briefly look at **stencils**, a graphical representation of calculation locality.

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Blocking and synchronous communication
Non-blocking communication in MPI
Performance benefits and latency hiding

Blocking communication

Definition

A communication is **blocking** if return of control to the calling process **only** occurs after **all resources** are safe to re-use.

In MPI, **resources** primarily refers to the memory allocated for the message, such as the pointer `data` in this `MPI_Send()` example:

```
1 MPI_Send( data, size, MPI_INT, ... );
```

Note this only refers to the viewpoint of the **calling** process; the **receiving** process is not mentioned.

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Blocking and synchronous communication
Non-blocking communication in MPI
Performance benefits and latency hiding

Synchronous communication

Definition
Communication is **synchronous** if the operation does not complete
before **both** processes have started their communication operation.

On return all resources can be re-used *and* we know the
destination process has started receiving[1].

For instance, a blocking call may return once the data has been
copied to the buffer, **before it has even been sent to the
network** (and is therefore **not** synchronised with the receiver).

---

[1]MPI supports synchronised communication with MPI_Ssend(). A common use
is **debugging**: If replacing MPI_Send() with MPI_Ssend() results in **deadlock**,
the original code would have deadlocked **when the data exceeded the buffer**.

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Blocking and synchronous communication
Non-blocking communication in MPI
Performance benefits and latency hiding

Non-blocking and asynchronous communication

Definition

A **non-blocking operation** may return before it is safe to re-use the resources. In particular, changing `data` after returning **may change the data being sent**.

Essentially, such calls only **start** the communication.

Definition

Asynchronous communication does not require any co-operation between the sender(s) and the receiver(s).

e.g. a send that **doesn't expect a corresponding receive**.

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Blocking and synchronous communication
Non-blocking communication in MPI
Performance benefits and latency hiding

Blocking $\neq$ synchronous

Sometimes the terms *blocking* and *synchronous*, and *non-blocking* and *asynchronous*, are used interchangeably.

- Blocking *can* act as a form of synchronisation.
- e.g. `MPI_Recv()` will not return until the data has been received.

However, the distinction is more subtle:

- **Blocking** and **non-blocking** refer to a single process's view, *i.e.* 'what the programmer needs to know.'

- **Synchronous** and **asynchronous** refer to a more global view involving at least two processes.

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Blocking and synchronous communication
Non-blocking communication in MPI
Performance benefits and latency hiding

Non-blocking communication in MPI

The key routines are:

```
MPI_Isend() : Start a non-blocking send.
MPI_Irecv() : Start a non-blocking receive.
MPI_Wait()  : Will not return until the communication is complete.
MPI_Test()  : Test to see if the communication is complete (but
              return immediately).
```

The 'I' in MPI_Isend() and MPI_Irecv() stands for **immediate**, because they return (almost) immediately.

There are other routines, including non-blocking collective communication in MPI v3 [`MPI_Ibcast()`, ...], but these will not be covered here.

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Blocking and synchronous communication
Non-blocking communication in MPI
Performance benefits and latency hiding

MPI_Request

To link each `MPI_Isend()` or `MPI_Irecv()` with its corresponding `MPI_Wait()` or `MPI_Test()`, MPI uses **requests**.

```
1  MPI_Request request;
2  MPI_Status status;
3
4  // Start the communication.
5  MPI_Isend( data, size, ..., &request );
6
7  // Do other things not involving 'data'.
8  ...
9
10 // Wait until the communication is complete.
11 // (Can replace &status with MPI_STATUS_IGNORE.)
12 MPI_Wait( &request, &status );
13
14 // Can now safely re-use 'data'.
```

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Blocking and synchronous communication
Non-blocking communication in MPI
Performance benefits and latency hiding

Why use non-blocking communication?

Since a non-blocking communication call returns immediately, we can perform other useful calculations **while the communication is going on** – as long as they do not involve the resources.

So rather than performing calculations and communications sequentially, some may be performed **concurrently**.

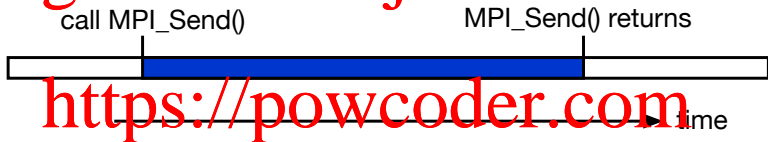- Reduces total runtime, improving performance.

Latency hiding

The primary reason to use non-blocking communication is to **overlap** communication with computation or other communications. This is known as **latency hiding**.

Overview
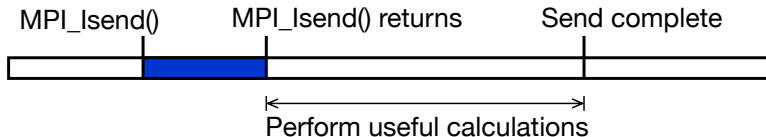Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Blocking and synchronous communication
Non-blocking communication in MPI
Performance benefits and latency hiding

Schematic (sending)

Blocking MPI_Send()

call MPI_Send()   MPI_Send() returns

time

Non-blocking MPI_Isend()

MPI_Isend()   MPI_Isend() returns   Send complete

Perform useful calculations

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Blocking and synchronous communication
Non-blocking communication in MPI
Performance benefits and latency hiding

Testing for completion or lock availability

In Lecture 7 we saw how locks can be used to synchronise threads in a shared memory system:

```
1  regionLock.lock();
2  // Does not return until lock acquired
```

The lock() method is blocking: it does not return until the lock is available.

Using test() could allow useful calculations to be performed while waiting for the lock to become available:

```
1  while ( !regionLock.test() )
2    { ... /* Do as many calculations as possible */ }
```

The MPI function MPI_Test() performs a similar role for non-blocking communication.

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Domain partitioning
Stencils
Ghost cells
Implementations with computation and communication

Potential applications of non-blocking communucation

Many applications require large data sets to be modified according to some rules. Examples include[1]:

Signal processing: (1D data sets)

- Frequency analysis, noise filtering, ...

Image processing: (2D data sets)

- Colour filtering, blurring, edge detection, ...

Scientific and engineering modelling: (1D, 2D or 3D)

- Fluid dynamics, elasticity/mechanics, weather forecasting, ...

---

[1]Wilkinson and Allen, *Parallel programming* (Pearson, 2005).

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Domain partitioning
Stencils
Ghost cells
Implementations with computation and communication

Domain partitioning

The standard way to parallelise such problems with **distributed** memory is to **partition the domain** between the processes, *i.e.*
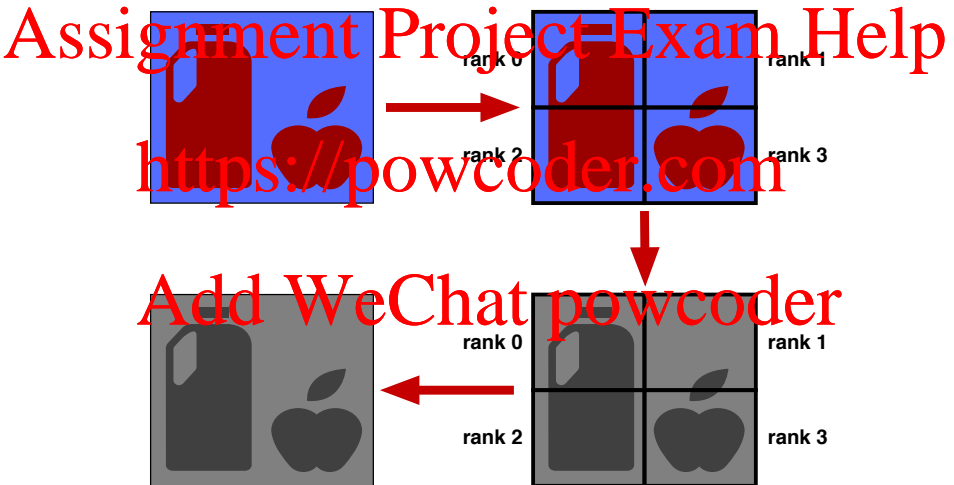
- Segments of a time series.
- Regions of an *image* [*next slide*].
- . . .

> **Domain partitioning**
> Each processing unit is responsible for transforming **one partition**.

If the transformation only depends on each data point **in isolation**, this is a **map**; also an **embarrassingly parallel problem**.

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Domain partitioning
Stencils
Ghost cells
Implementations with computation and communication

Map example: Colour transformation

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Domain partitioning
Stencils
Ghost cells
Implementations with computation and communication

Local transformations

More commonly, however, the transformation depends on **nearby information**.

- **Blurring** or **edge detection** in image processing.

- Most scientific and engineering applications solve equations with **gradient** terms (i.e. changes in quantities).
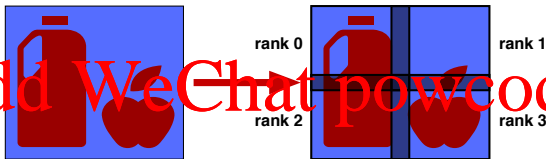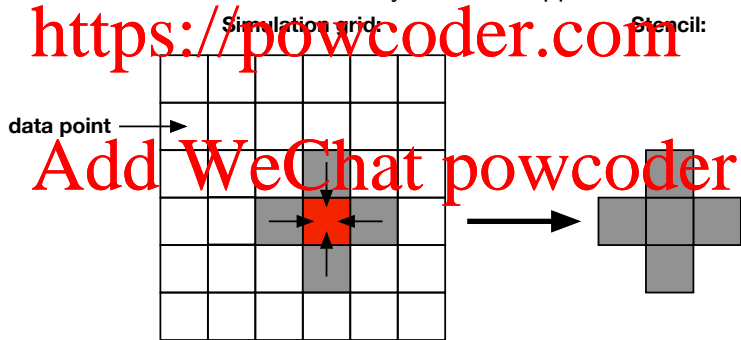

Need to **communicate** information lying at the **edges** of domains to perform the calculations correctly.

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Domain partitioning
Stencils
Ghost cells
Implementations with computation and communication

Stencils

A **stencil** is a graphical representation of where the required data exists relative to point being calculated.

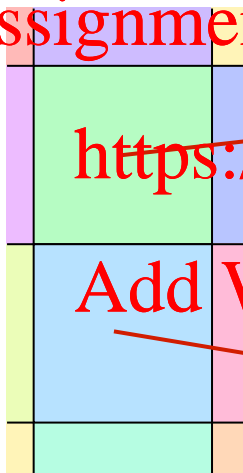This common stencil arises in many scientific applications:

Simulation grid: **Stencil:**

data point ⟶

red cell calculation requires values of grey cells

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Domain partitioning
Stencils
Ghost cells
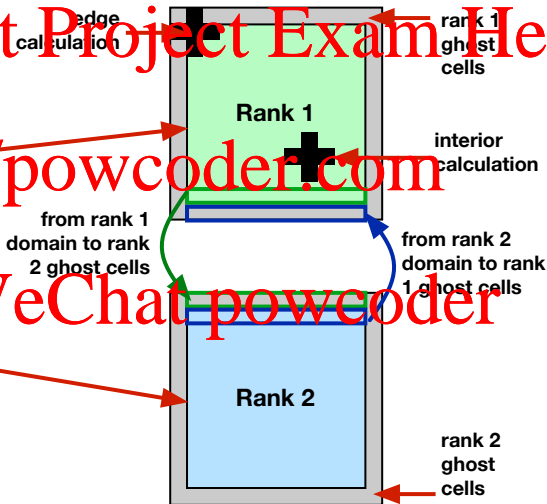Implementations with computation and communication

Ghost cells

The standard way to communicate across boundaries is to use **ghost cells**, sometimes known as a **halo**.

- Layers of data points around each process's domain.
- Contains **read-only copy** of corresponding points from neighbouring processes' domain.
- **Updated** after each iteration to match the values calculated by the neighbouring processes.
- Updating performed using **point-to-point communication**.

Overview · Non-blocking and asynchronous communication · **Application: Domain partitioning** · Summary and next lecture

Domain partitioning · Stencils · **Ghost cells** · Implementations with computation and communication

Conceptual simulation domains:

Implemented simulation domains:

edge calculation

rank 1 ghost cells

Rank 1

interior calculation

from rank 1 domain to rank 2 ghost cells

from rank 2 domain to rank 1 ghost cells

Rank 2

rank 2 ghost cells

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Domain partitioning
Stencils
Ghost cells
Implementations with computation and communication

Implementation v1
Code on Minerva: `heatEqn.c`

In pseudocode the obvious implementation would be:

```
1  // Iterate multiple times.
2  for(iter=0;iter<NUM_ITERATIONS;iter++)
3  {
4    // Send data at edge of domain to other processes.
5    communicateBetweenDomains();   // BLOCKING.
6
7    // Update values within this rank's domain.
8    solveWithinDomain();
9  }
```

However, this ignores the fact that **only** the data points **near** to the edge of the domain require other processes' data.

- **Interior sites** can be calculated **prior to communication**.
- This is normally the **bulk** of the calculation.

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Domain partitioning
Stencils
Ghost cells
Implementations with computation and communication

Implementation v2

Since the ghost cell need only be updated before the edge cell calculations, can in principle solve the interior cells first:
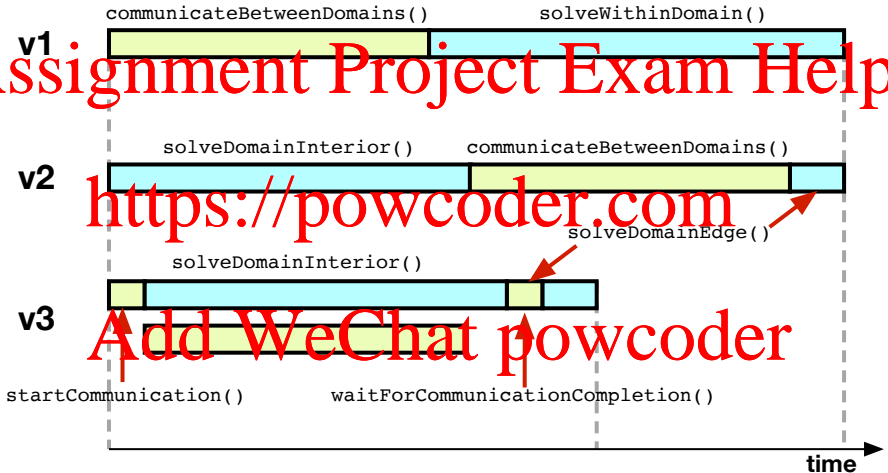
```
1  for ( iter =0; iter < NUM_ITERATIONS ; iter ++)
2  {
3    // Calculate data points within the domain.
4    solveDomainInterior ();
5
6    // Send data at edge of domain to other processes.
7    communicateBetweenDomains ();    // BLOCKING.
8
9    // Now solve the remaining few points at the edge.
10   solveDomainEdge ();
11 }
```

But this is still using **blocking** communication.

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Domain partitioning
Stencils
Ghost cells
Implementations with computation and communication

Implementation v3

Use non-blocking communication to overlap the **calculation** of interior points with the **communication** of ghost cells.

```
 1  for(iter=0;iter<NUM_ITERATIONS;iter++)
 2  {
 3    // Start communication with other processes.
 4    startCommunication();  // NON-BLOCKING.
 5
 6    // Calculate data points within the domain
 7    // WHILE THE COMMUNICATION IS GOING ON!
 8    solveDomainInterior();
 9
10    // Wait until the communication has finished.
11    waitForCommunicationCompletion();
12
13    // Now solve the remaining few points at the edge.
14    solveDomainEdge();
15  }
```

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Domain partitioning
Stencils
Ghost cells
Implementations with computation and communication

v1

communicateBetweenDomains()   solveWithinDomain()

v2

solveDomainInterior()   communicateBetweenDomains()

solveDomainEdge()

v3

solveDomainInterior()

startCommunication()   waitForCommunicationCompletion()

time

Overview
Non-blocking and asynchronous communication
Application: Domain partitioning
Summary and next lecture

Summary and next lecture

Summary and next lecture

Today we have looked at the difference between **blocking** and **non-blocking** communication, and **synchronous** and **asynchronous** communication:

- Non-blocking can overlap communication with computation: **Latency hiding**.
- Example of **domain partitioning** using **ghost cells**.
- **Stencils** describe the locality of the calculation.

Next time we will look in detail at a *very* important concept for **all** parallel systems - **load balancing**.