Durham University
Department of Computer Science

2021 Code Performance Series: From analysis to insight

Third session

T. Weinzierl

March 2021

Agenda

Durham University
Department of Computer Science

9:00-9:30	Experience reports ▶ Flash
9:30-12:00	Score-P instrumentation & measurements
10:00-11:00	MPI Tracing
11:00-12:00	Case studies of parallel performance analyses with CUBE & Vampir
13:00-15:00	Workshop
13:00-14:00	MPI programming refresher (optional)

Experience report: Flash

Durham University
Department of Computer Science

Before we start: non-verbal feedback

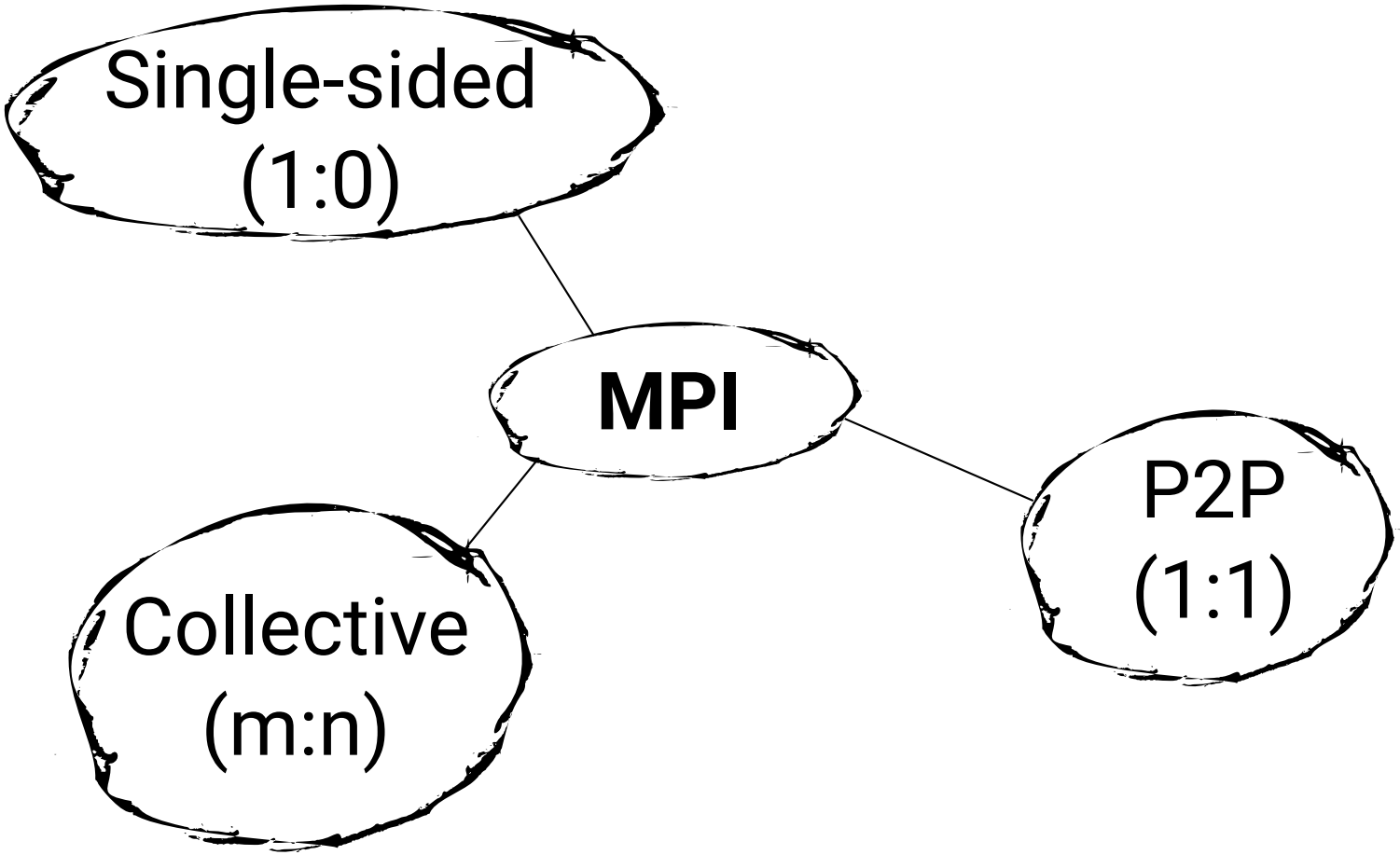
Durham University
Department of Computer Science

- I can't see (most of) you, but I'd appreciate non-verbal feedback
- Open `http://dev.atlas-tool.eu/room?username=username&creator=atlas&room=Performance+Analysis+Workshop03763003231`
- Press button (mouse, keyboard) in Windows *if something is not clear*
  (you can still scratch your head)
- Feel free to continue to press until things become clearer again
- We will distribute a questionnaire after the session to gather some feedback (pilot)

More info: `http://dev.atlas-tool.eu/tool`

Three types of message exchange

Durham University
Department of Computer Science



We ignore single-sided today.

Point-to-point: send plus receive

Durham University
Department of Computer Science

```
int MPI_Send(
  const void *buffer, int count, MPI_Datatype datatype,
  int dest, int tag, MPI_Comm comm
)
```

```
int MPI_Recv(
  void *buffer, int count, MPI_Datatype datatype,
  int source, int tag, MPI_Comm comm,
  MPI_Status *status
)
```

- `buffer` is a pointer to the piece of data you want to send away or into which you want to receive.
- Every send has to be matched by receive.
- Messages are transferred in-order.
- `tag` allows you to distinguish (and receive out-of-order).

Collectives for n:m

Durham University
Department of Computer Science

Collective operation: A collective (MPI) operation is an operation involving many/all nodes/ranks.

Type of collective	One-to-all	All-to-one	All-to-all
Synchronisation			
Communication			
Computation			

Insert the following MPI operations into the table (MPI prefix neglected):

- Barrier
- Broadcast
- Reduce
- Allgather
- Scatter
- Gather
- Allreduce

The notion of blocking

Durham University
Department of Computer Science

> Blocking: MPI_Send is called `blocking` as it terminates as soon as you can reuse the buffer, i.e. assign a new value to it, without an impact on MPI.

> Blocking: MPI_Recv is called `blocking` as it terminates as soon as you can read the buffer, i.e. MPI has written the whole message into this variable.

- If a blocking send returns, it does **not** mean that the corresponding message has been received on the other side.
- Blocking and asynchronous or synchronous execution have **nothing** to do with each other. However blocking receives do make receiver side lag behind sender.
- With blocking sends, you never have a guarantee that the data has been received, i.e. blocking sends are not *synchronised*.
- The term blocking just refers to the safety of the local variable. If a blocking send returns, the data might have been copied to the local network chip, e.g.

$\Rightarrow$ Both P2P ops and collectives are by default blocking.

Where messages roam

Durham University
Department of Computer Science

Where is the message buffered?

> Eager protocol: Destination rank has some memory buffer available or receive has been posted already.

> Rendezvous protocol: No spare buffer available. Sender thus waits until destination rank signals ok.

- Protocol choice depends on MPI configuration, number of messages in the system, message sizes, ...
- Eager reduces algorithmic latency, but
  - requires additional memory copies
  - searching through unexpected message queue is expensive
- Rendezvous reduces memory movement overhead, but
  - synchronises ranks

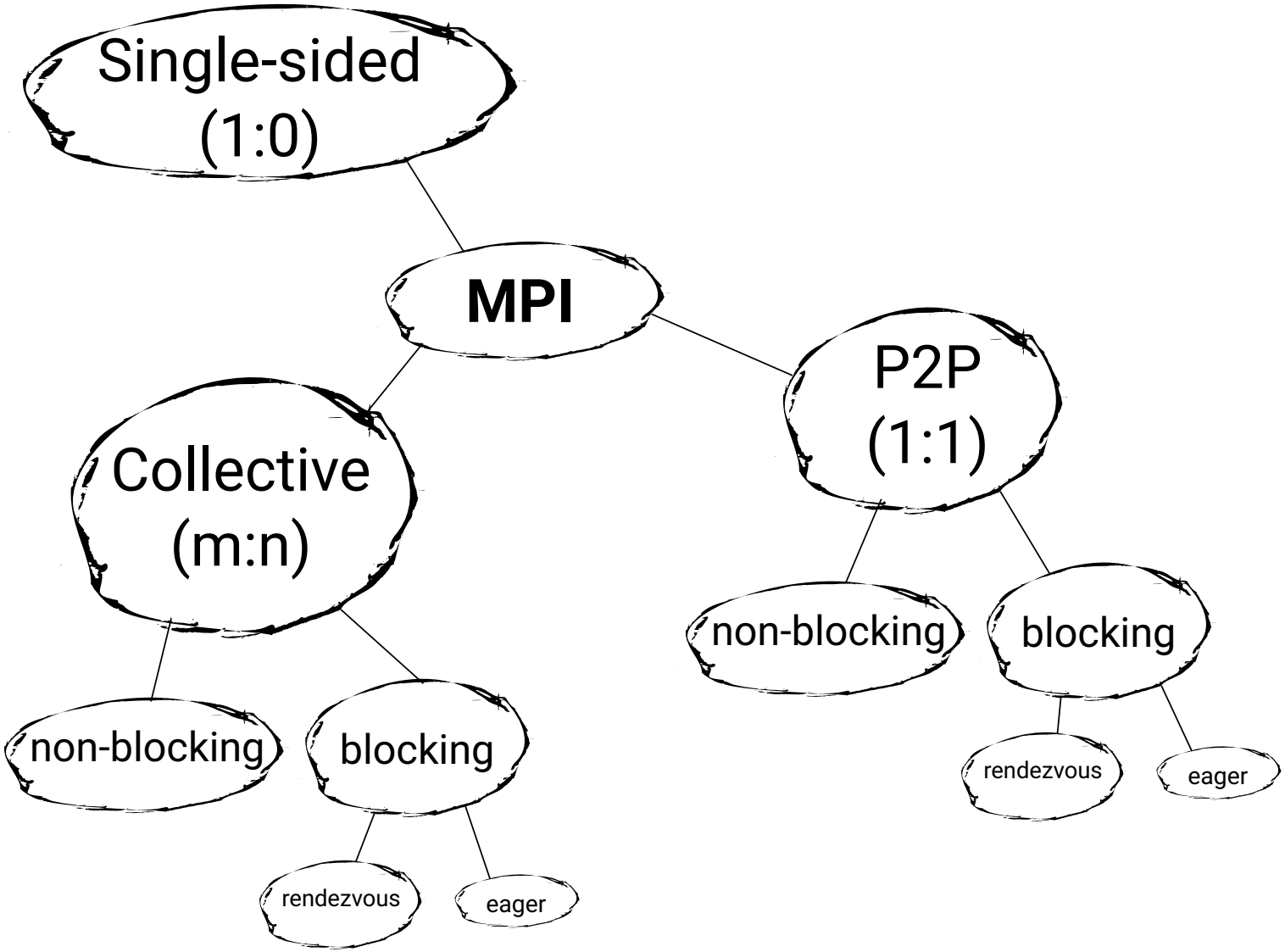
$\Rightarrow$ Deadlock #1: MPI falls back to rendezvous protocol but no buffer becomes available as receiver side in return waits for a buffer on another rank

$\Rightarrow$ Deadlock #2: MPI can't even get the envelope of rendezvous through

- You can tweak your MPI protocol thresholds (rarely do this; rather understand algorithm's behaviour)

Flavours of message exchange

Durham University
Department of Computer Science



Nonblocking P2P communication

Durham University
Department of Computer Science

- Non-blocking commands start with I (immediate return)
- Non-blocking means that operation returns immediately though MPI might not have transferred data yet
- Buffer (variable) thus is still in use and we may not overwrite it
- We explicitly have to validate whether message transfer has completed before we reuse or delete the buffer

```
Create helper variable (handle)
int a = 1;
trigger the send
do some work
check whether communication has completed
a = 2;
...
```

- We now can overlap communication and computation, as we eliminate synchronisation.
- Works for P2P and collectives.
- Wannabe MPI experts will sell you this one as solution to all problems.

Isend and Irecv


```
int MPI_Send(
  const void *buffer, int count, MPI_Datatype datatype,
  int dest, int tag, MPI_Comm comm
)
```

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

int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
int MPI_Wait(MPI_Request *request, MPI_Status *status)
```

- Pass additional pointer to object of type `MPI_Request`.
- Non-blocking, i.e. operation returns immediately.
- Check for send completition with `MPI_Wait` or `MPI_Test`.
- `MPI_Irecv` analogous.
- The status object is not required for the receive process, as we have to hand it over to wait or test later.

Progression

Durham University
Department of Computer Science

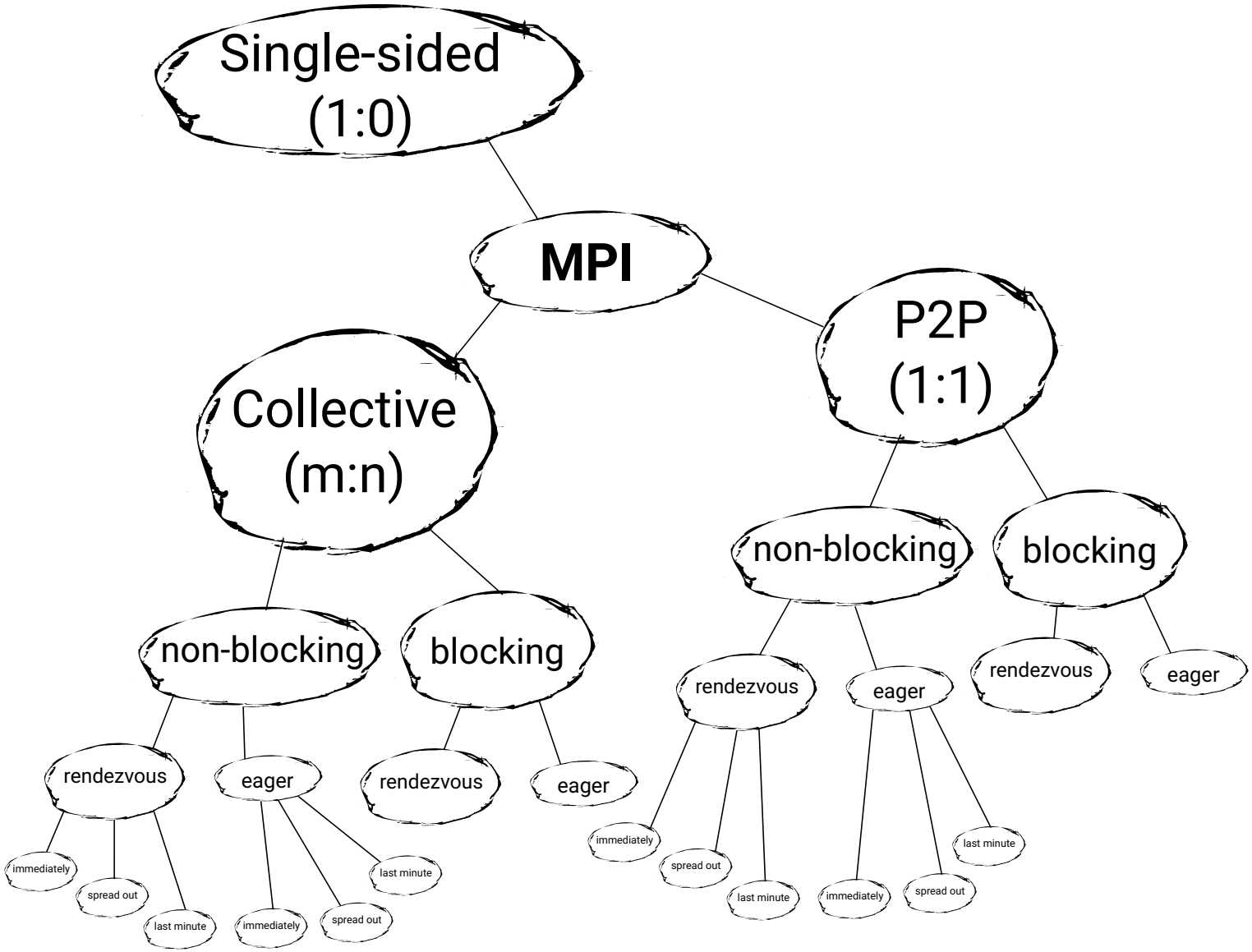
MPI Progression: Machine still needs CPU cycles to transfer message

- Rely on separate (progression) thread (Intel MPI)
- Rely on hardware
- Rely on MPI to plug into other MPI calls

If MPI does the job (predominant pattern):

- Immediately: Send immediately returns, i.e. logically no difference to blocking exchange.
- Spread out: Some `MPI_xxx` call in your code makes MPI interrupt your execution and transfer data.
- Last minute: The final `MPI_Wait` or `MPI_Test` makes MPI realise message transfer.

Summary


Single-sided
(1:0)
MPI
Collective
(m:n)
P2P
(1:1)
non-blocking
blocking
non-blocking
blocking
rendezvous
eager
rendezvous
eager
rendezvous
eager
rendezvous
eager
immediately
spread out
last minute
immediately
spread out
last minute
immediately
spread out
last minute
immediately
spread out
last minute