

# Profiling MPI code to identify bottlenecks

Ed Bennett

@QuantumofEd



Swansea University  
Prifysgol Abertawe

@SwanseaUni



SUPERCOMPUTING WALES  
UWCHGYFRIFIADURA CYMRU

@SuperCompWales



@sa2c\_swansea

Supercomputing Wales Swansea Symposium, 2018-09-13

- MPI: **Message Passing Interface**
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - Communication is explicit
- Initialise program with `MPI_Init`, finish with `MPI_Finalize`
- Point-to-point send and receive with `MPI_Send`, `MPI_Recv`
- Immediate (non-blocking) versions: `MPI_Isend`, `MPI_Irecv`
- Point-to-all operations with `MPI_Bcast`
- Collectives, e.g. `MPI_Reduce`

# Profiling MPI

- Intel MPI provides profiling tools

```
source /apps/compilers/intel/2018.3/itac_2018/bin/itacvars.sh  
mpirun -trace [my_application]
```
- Output can be *large*, so choose a small problem size
- Visualise the results with traceanalyzer

# Summary view

Intel® Trace Analyzer

File Options Project Windows Help

**Summary:** [ ]

Total time: 2.3e+03 sec. Resources: 8 processes, 1 node.

**Continue >**

---

**Ratio**

This section represents a ratio of all MPI calls to the rest of your code in the application.



Serial Code - 309 sec	13.4 %
OpenMP - 0 sec	0 %
<b>MPI calls - 1.99e+03 sec</b>	<b>86.5 %</b>

---

**Top MPI functions**

This section lists the most active MPI functions from all MPI calls in the application.

MPI_Bcast	1.21e+03 sec (52.8 %)
MPI_Barrier	722 sec (31.4 %)
MPI_Recv	37.5 sec (1.63 %)
MPI_Send	14.6 sec (0.636 %)
MPI_Finalize	0.602 sec (0.0262 %)

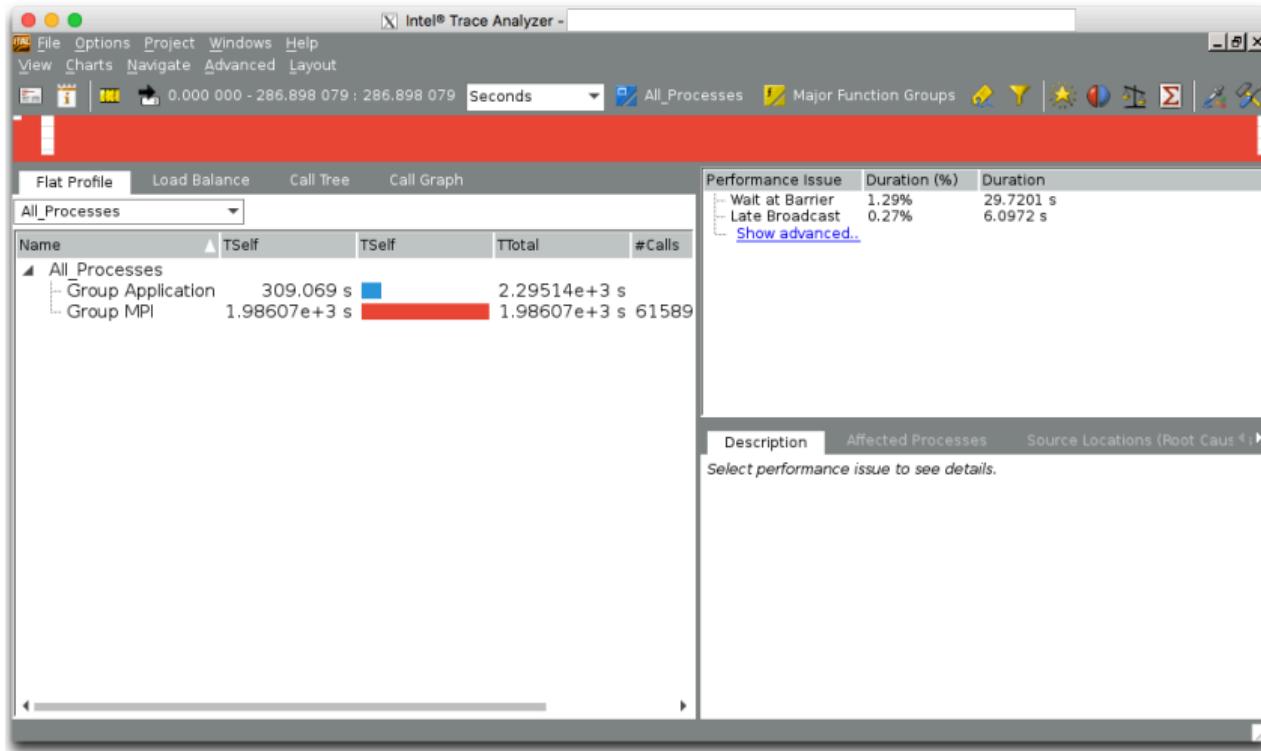
---

**Where to start with analysis**

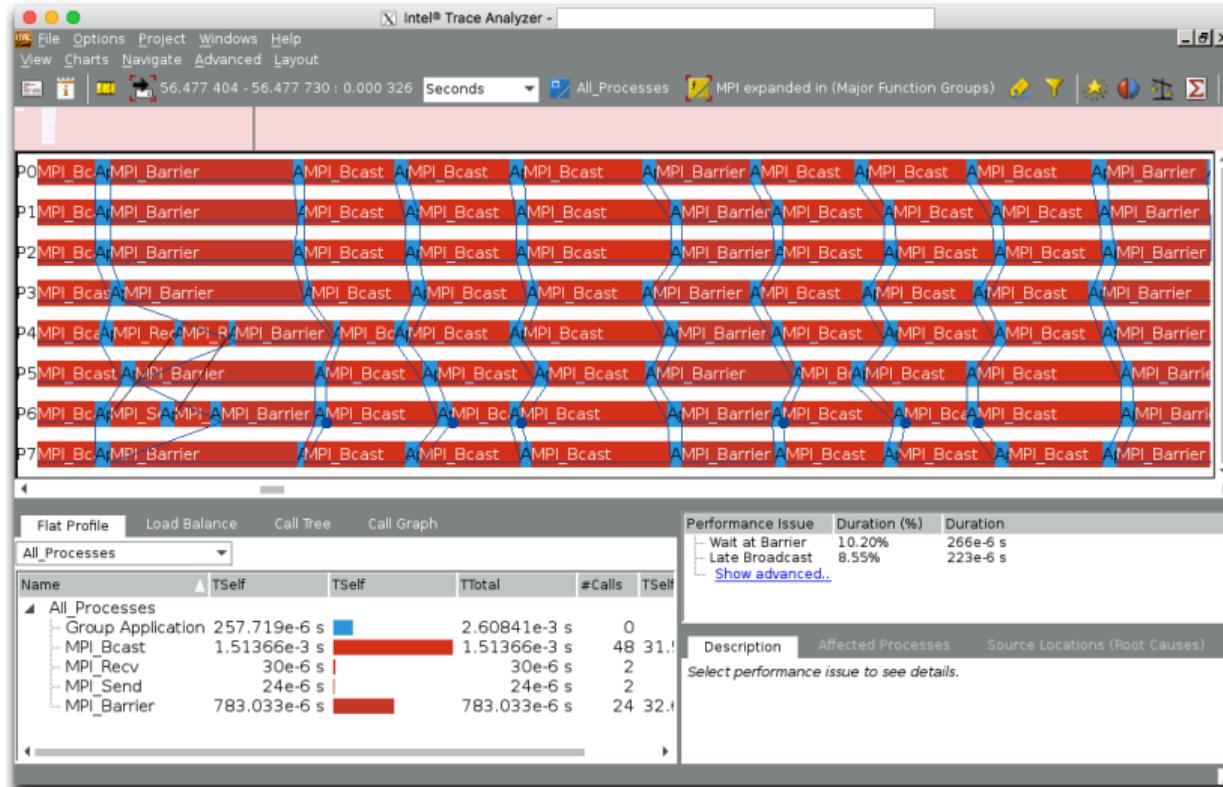
For deep analysis of the MPI-bound application click "Continue >" to open the tracefile View and leverage the **Intel® Trace Analyzer** functionality.

To optimize node-level performance use:  
**Intel® VTune™ Amplifier XE** for:  
- algorithmic level tuning with `hpc-performance` and

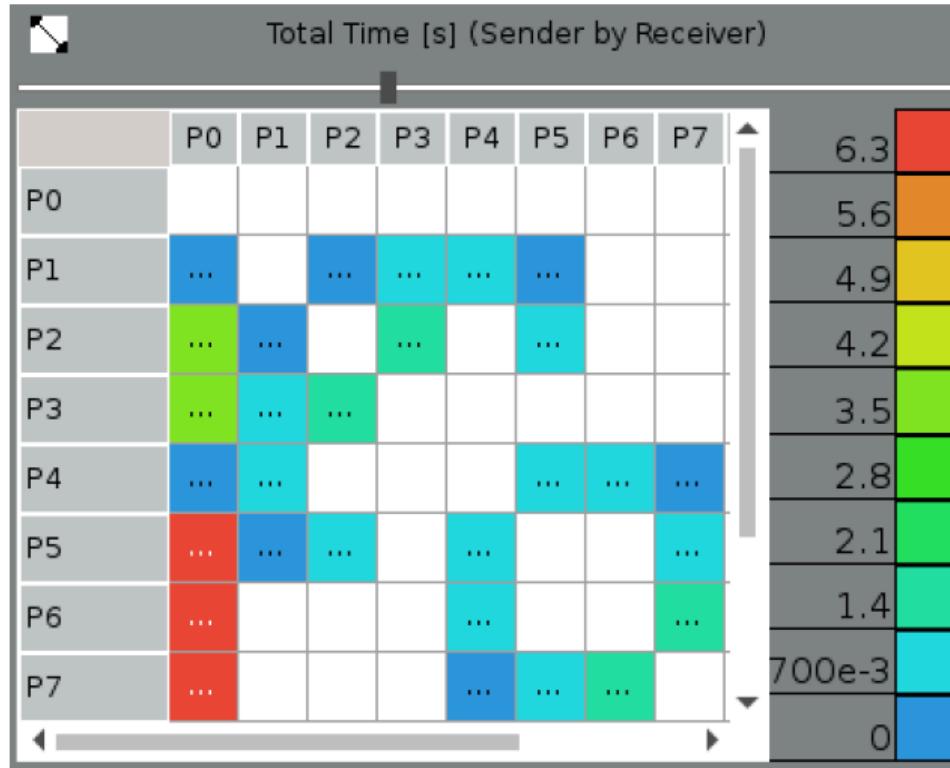
# Whole application timeline



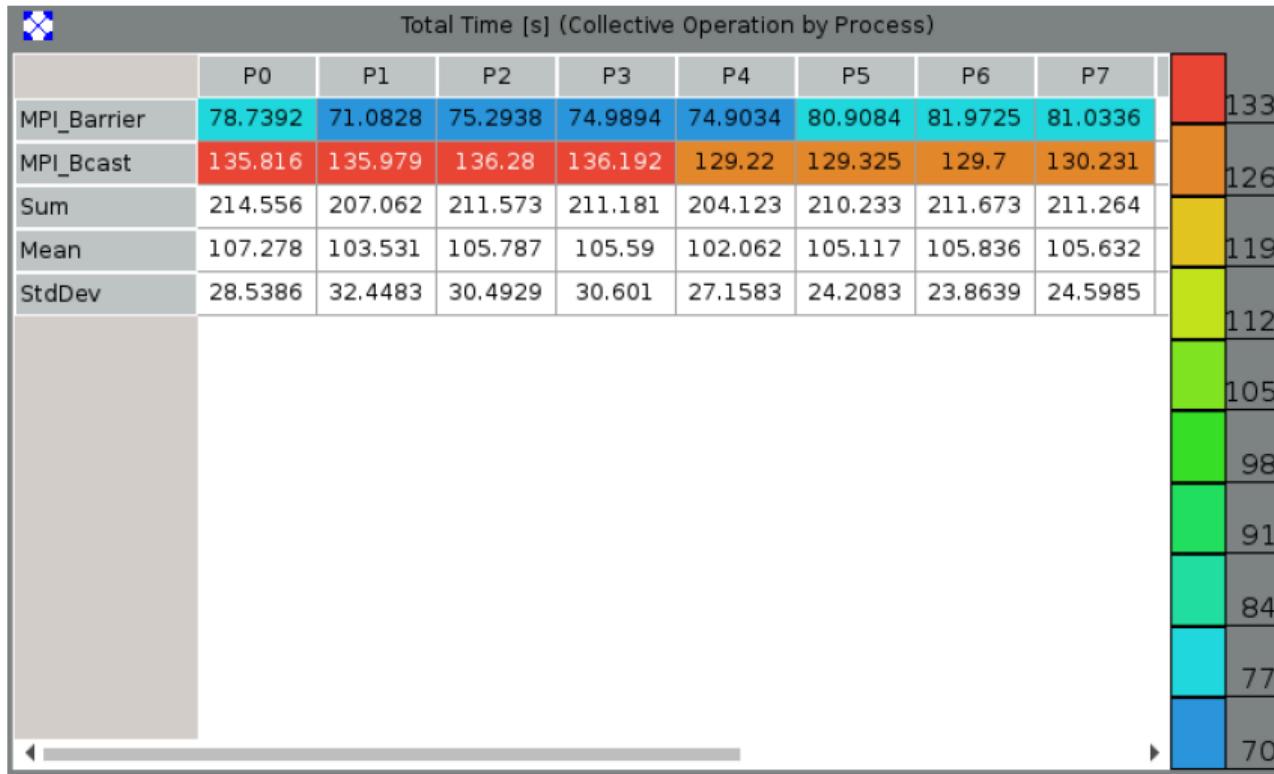
# Detailed timeline



# Message pattern



# Reductions summary



# Ways forward

- We found that:
  - MPI\_Barrier and MPI\_Bcast taking up a lot of time
  - Communications are scattered throughout the program
  - Very small amounts of compute between each communication
  - Coordinator-worker pattern
  - Some minor load balancing concerns
- Things to think about
  - Can we use a peer-to-peer pattern instead?
  - Can we increase the amount of work between communications?
  - Is the direction we're parallelising appropriate?
- If that fails, consider more advanced MPI features

# TIL #1: MPI-IO

- Avoid channelling all I/O through a single rank/node
- Higher performance for reading and writing data
- Works really well with subarray types

```
call MPI_File_Open(comm, 'con', MPI_Mode_Rdonly, &
MPI_Info_Null, mpi_fh)
call MPI_File_Set_View(mpi_fh, 0_8, MPI_Real, mpiio_type, &
"native", MPI_Info_Null)
call MPI_File_Read_All(mpi_fh, theta, &
3 * ksizex_1 * ksizey_1 * ksizet_1, &
MPI_Real, status)
call MPI_File_Close(mpi_fh)
```

# Subarray types

```
subroutine init_single_halo_type_4(direction, position, size4, datatype, typet)
    integer, intent(in) :: direction, position, size4
    type(MPI_Datatype), intent(in) :: datatype
    type(MPI_Datatype), intent(out) :: typet
    integer, dimension(4) :: sizes, subsizes, starts
    sizes = (/ ksizex_l + 2, ksizey_l + 2, ksizet_l + 2, size4 /)
    subsizes = (/ ksizex_l, ksizey_l, ksizet_l, size4 /)
    subsizes(direction+1) = 1
    starts = (/ 1, 1, 1, 0 /)
    starts(direction+1) = position
    call MPI_Type_Create_Subarray(4, sizes, subsizes, starts, &
                                MPI_Order_Fortran, datatype, typet)
    call MPI_Type_Commit(typet)
    return
end subroutine init_single_halo_type_4
```

# Persistent MPI communications

- Each MPI\_Send/MPI\_Recv pair has an overhead
- For a tight loop, this wastes time
- Instead, use MPI\_Send\_Init/MPI\_Recv\_Init outside the loop
- MPI\_Start/MPI\_StartAll inside
- Also need MPI\_Wait/MPI\_WaitAll
- Speedup achieved will depend on architecture/communications fabric combination
- Collectives planned for MPI 3.2, e.g. MPI\_AllReduce\_Init

# Thanks for listening!

