Profiling MPI code to identify bottlenecks

Ed Bennett
@QuantumofEd

Swansea University
Prifysgol Abertawe
@SwanseaUni

SUPERCOMPUTING WALES
UWCHGYFRIFIADURA CYMRU
@SuperCompWales

SA2C
@sa2c_swansea

Supercomputing Wales Swansea Symposium, 2018-09-13

Slides: git.io/fAV4t

MPI, briefly

- MPI: **M**essage **P**assing **I**nterface

MPI, briefly

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1

MPI, briefly

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model

MPI, briefly

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)

MPI, briefly

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - *Distributed memory*

MPI, briefly

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - *Distributed memory*
  - Communication is explicit

MPI, briefly

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - *Distributed memory*
  - Communication is explicit
- Point-to-point send and receive

MPI, briefly

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - *Distributed memory*
  - Communication is explicit
- Point-to-point send and receive
- **I**mmediate (non-blocking) versions

MPI, briefly

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - *Distributed memory*
  - Communication is explicit
- Point-to-point send and receive
- **I**mmediate (non-blocking) versions
- Point-to-all (broadcast) operations

MPI, briefly

- MPI: **M**essage **P**assing **I**nterface
- Born in 1991, now at version 3.1
- Single Program Multiple Data model
  - Same program runs multiple times (on one or more nodes)
  - *Distributed memory*
  - Communication is explicit
- Point-to-point send and receive
- **I**mmediate (non-blocking) versions
- Point-to-all (broadcast) operations
- Collectives, e.g. reductions (sums, maxima, minima, etc.)

Profiling MPI

- Intel MPI provides profiling tools

```
source /apps/compilers/intel/2018.3/itac_2018/bin/itacvars.sh
mpirun -trace [my_application]
```

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

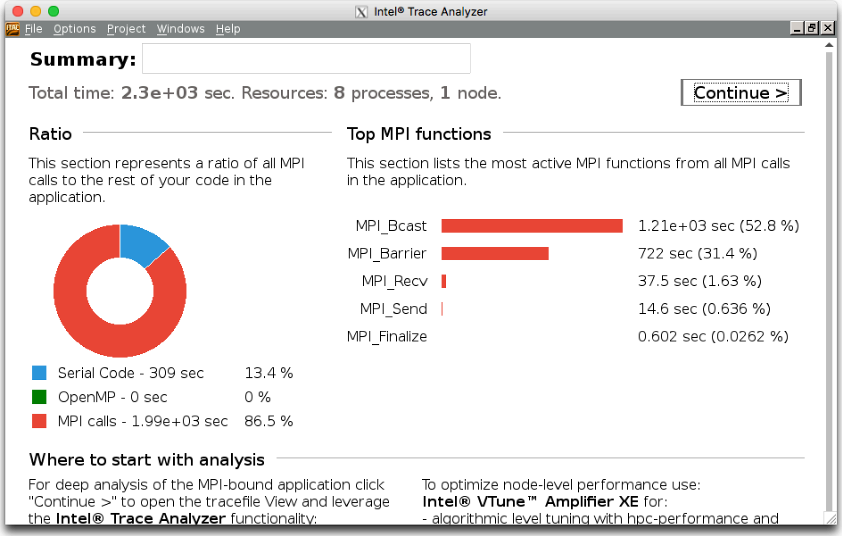
Profiling MPI

- 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



Summary:

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

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 %
🟥	MPI calls - 1.99e+03 sec	86.5 %

Top MPI functions

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

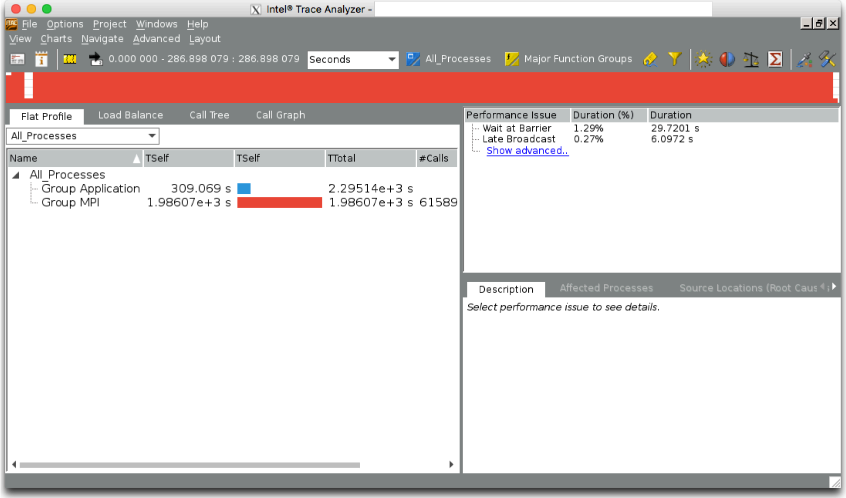
MPI_Bcast	1.21e+03 sec (52.8 %)
MPI_Barrier	722 sec (31.4 %)
MPI_Recv	37.5 sec (1.63 %)
MPI_Send	14.6 sec (0.636 %)
MPI_Finalize	0.602 sec (0.0262 %)

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



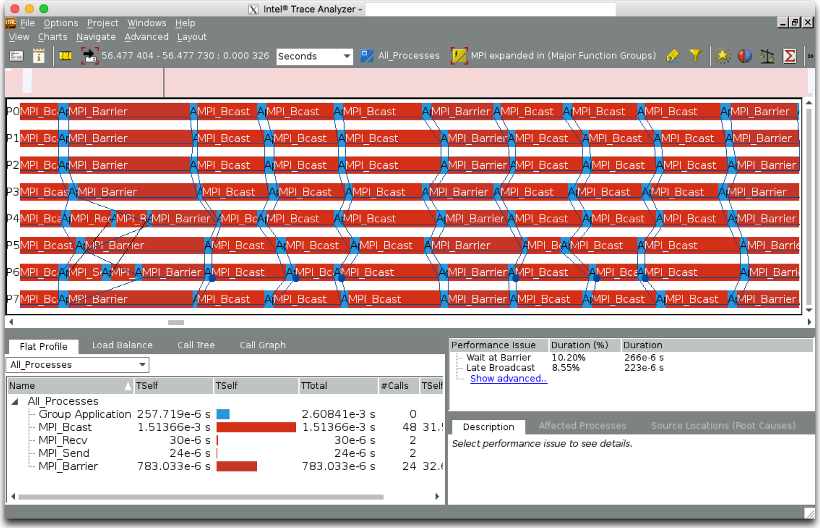
Name	TSelf	TSelf	TTotal	#Calls
All_Processes				
Group Application	309.069 s	🟦	2.29514e+3 s	
Group MPI	1.98607e+3 s	🟥	1.98607e+3 s	61589

Performance Issue	Duration (%)	Duration
Wait at Barrier	1.29%	29.7201 s
Late Broadcast	0.27%	6.0972 s
Show advanced..		

Select performance issue to see details.

Slides: git.io/fAV4t

Detailed timeline



56.477 404 - 56.477 730 : 0.000 326 Seconds All_Processes MPI expanded in (Major Function Groups)

All_Processes

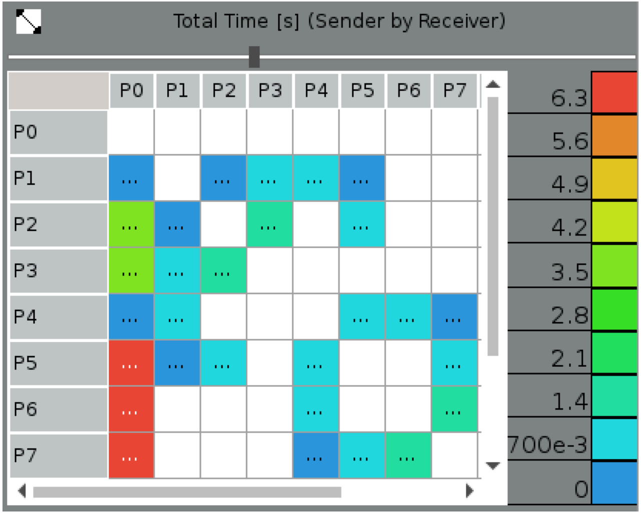
Name	TSelf	TSelf	TTotal	#Calls	TSelf
All_Processes					
Group Application	257.719e-6 s		2.60841e-3 s	0	
MPI_Bcast	1.51366e-3 s		1.51366e-3 s	48	31.
MPI_Recv	30e-6 s		30e-6 s	2	
MPI_Send	24e-6 s		24e-6 s	2	
MPI_Barrier	783.033e-6 s		783.033e-6 s	24	32.

Performance Issue	Duration (%)	Duration
Wait at Barrier	10.20%	266e-6 s
Late Broadcast	8.55%	223e-6 s
Show advanced..		

Select performance issue to see details.

Slides: git.io/fAV4t

Message pattern

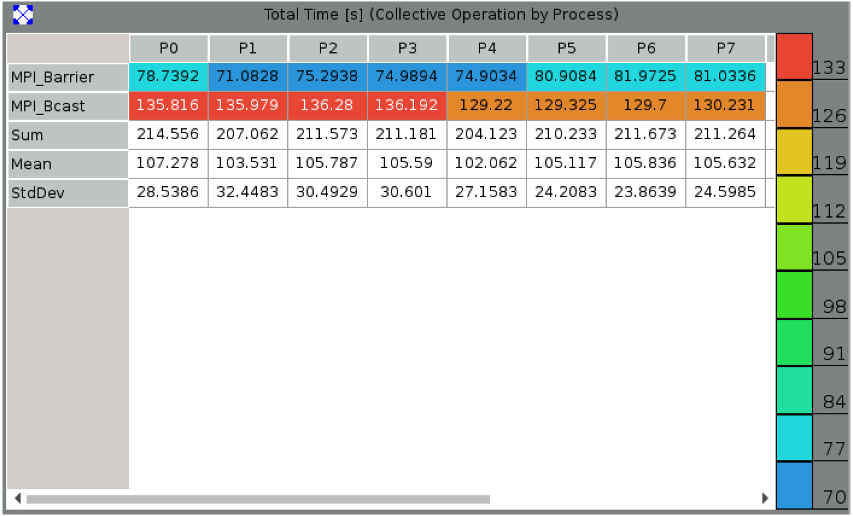


Total Time [s] (Sender by Receiver)

	P0	P1	P2	P3	P4	P5	P6	P7
P0								
P1	...		...	...	...	...		
P2	...	...		...		...		
P3	...	...	...					
P4	...	...				...	...	...
P5	...	...	...		...			...
P6	...				...			...
P7	...				...	...	...	

Slides: git.io/fAV4t

Reductions summary



Total Time [s] (Collective Operation by Process)

	P0	P1	P2	P3	P4	P5	P6	P7
MPI_Barrier	78.7392	71.0828	75.2938	74.9894	74.9034	80.9084	81.9725	81.0336
MPI_Bcast	135.816	135.979	136.28	136.192	129.22	129.325	129.7	130.231
Sum	214.556	207.062	211.573	211.181	204.123	210.233	211.673	211.264
Mean	107.278	103.531	105.787	105.59	102.062	105.117	105.836	105.632
StdDev	28.5386	32.4483	30.4929	30.601	27.1583	24.2083	23.8639	24.5985

Slides: git.io/fAV4t

Ways forward

- We found that:

Ways forward

- We found that:
  - MPI_Barrier and MPI_Bcast taking up a lot of time

Ways forward

- We found that:
  - `MPI_Barrier` and `MPI_Bcast` taking up a lot of time
  - Communications are scattered throughout the program

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

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

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

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

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?

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?

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?

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

TIL #1: MPI-IO

- Avoid channelling all I/O through a single rank/node
- Higher performance for reading and writing data

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

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

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_l * ksizey_l * ksizet_l, &
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

Persistent MPI communications

- Each `MPI_Send`/`MPI_Recv` pair has an overhead
- For a tight loop, this wastes time

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

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

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`

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

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!

Slides: git.io/fAV4t