Identifying Performance Bottlenecks in Hybrid MPI + OpenMP Software

Jonathan Boyle (NAG) & Judit Giménez (BSC)

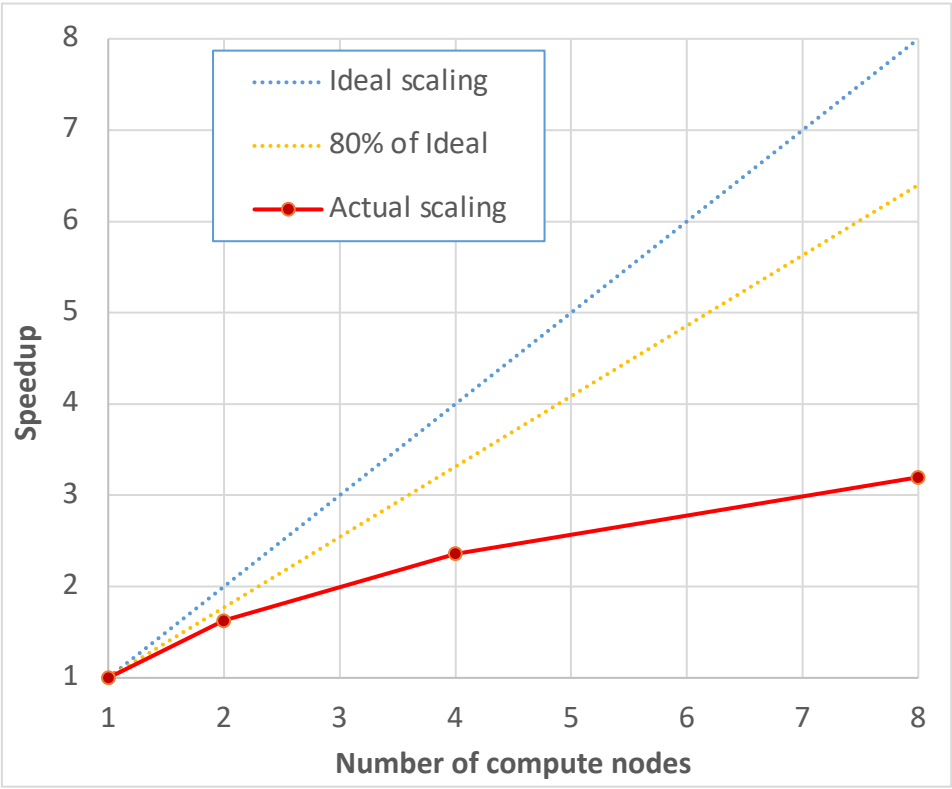
EU H2020 Centre of Excellence (CoE)

1 December 2018 – 30 November 2021

Grant Agreement No 824080

The problem we're addressing

- Getting good parallel performance is hard!!!
- Even with the simplest parallelism we need to think about
  - Serialisation
  - Load balance
  - Memory bottlenecks
  - Processor frequency
  - Instruction count
  - Overheads of parallelization
- MPI + OpenMP adds complexity
  - Is the problem in the OpenMP?
  - Or the MPI?
- And parallel scaling tells us nothing about what is wrong!!



Performance analysis tools

- There are powerful tools to us help analyse performance
  - BSC toolset – Extrae, Paraver, Dimemas, etc
  - Scalasca toolset – Score-P, Scalasca, Cube
  - Intel's tools – VTune, ITAC, etc
  - Plus many more
- But the data they generate is often overwhelming
  - Extremely detailed and complex event timelines per thread
  - Vast amounts of difficult to understand metrics
- **We need to make sense of the trace data before we know what we're looking for in the trace data**
  - But how???

The solution – the POP metrics

- **An easy to understand set of metrics**
- **Metrics which can be easily calculated from typical trace data**
- **Metrics which point at specific causes of performance bottlenecks**

- This idea has been successfully demonstrated by POP's MPI metrics
- We now extend to hybrid MPI + OpenMP (& beyond?)
  - We have **two** approaches for hybrid, both based on the POP MPI metrics
- One extends the idea of defining an individual metric per source of inefficiency
- The other extends the multiplicative approach of the MPI metrics

POP's high level metrics

- The POP methodology defines hierarchies of metrics
  - See POP online training "*Understanding Application Performance with the POP Metrics*" for details
- The top level metrics for both MPI + OpenMP schemes are identical
  - And are the same as those in POP's MPI metrics

1. **Parallel Efficiency**
   - How efficient is the parallelisation?
2. **Computation Scaling**
   - Is the total useful computation increasing or decreasing?
3. **Global Efficiency**
   - Combines inefficiency from parallelisation and computation scaling

POP's high level metrics

- We measure ***comp*** = time in useful computation on each CPU core
  - i.e. the time spent executing useful user code
- And ***comp_ref*** = useful computation on a reference case
  - Usually the smallest number of cores used e.g. 1 compute node
- Assuming strong computation scaling:

Parallel Efficiency = average(*comp*) / *runtime*

Computation Scaling = sum(*comp_ref*) / sum(*comp*)

Global Efficiency = Computation Scaling x Parallel Efficiency

Hybrid metrics method 1

The POP ‘additive’ scheme for MPI + OpenMP

The idea of ‘additive’ metrics

- Lets revisit POP’s **Parallel Efficiency**
  - This measures the efficiency of the parallelisation
- We measure ***comp*** and ***runtime*** on ***n*** cores
  - ***comp*** is useful computation per CPU core

Parallel Efficiency = average(*comp*) / *runtime*

- We can think of **average(*comp*)** as an **ideal runtime**

Parallel Efficiency ideal runtime = average(*comp*) = sum(*comp*) / *n*

- It is the runtime we would get if all useful work (***comp***) is split evenly over the cores with no overheads from the parallelism

The idea……

- Now look again at POP's **Global Efficiency**
  - This measures efficiency of the parallelisation combined with inefficiency due to increases in useful computation
- ***comp_ref*** is useful computation on our reference case
- ***n*** is number of cores for other cases under consideration

Global Efficiency = [sum(*comp_ref*)/*n*] / *runtime*

- We can think of an 'ideal runtime' for Global Efficiency

Global Efficiency ideal runtime = sum(*comp_ref*)/*n*

- This is the runtime we would get if the work from the reference case was to be split evenly over ***n*** cores with no overheads

POP's 'additive' metrics

- Define performance bottlenecks that can be mapped to **known issues**
- For each class of performance bottleneck define an 'ideal runtime'
- Then for each class of bottleneck:

efficiency = ideal runtime / runtime

- We can also define:

inefficiency = 1 - efficiency

  - For optimal performance: efficiency = 1, and inefficiency = 0

- This defines a hierarchy where we can **add** 'child' inefficiency values to get the 'parent' inefficiency value
  - Since **inefficiency = 'time cost of bottleneck(s)' / runtime**
  - Splitting the cost of the bottleneck into the individual contributions is the same as splitting the inefficiency value

Additive metrics for MPI + OpenMP

- We start with Parallel Efficiency and want to split this into child metrics
- With additive metrics we have complete freedom about how we define child metrics

- One obvious option is to split Parallel Efficiency into
  1. Process Efficiency (ignores all thread inefficiencies)
  2. Thread Efficiency (ignores process inefficiencies)

Parallel Inefficiency = Process Inefficiency + Thread Inefficiency

Illustration: Process Efficiency

- To assess Process Efficiency we want to know
  1. How evenly ‘useful work’ is distributed over the processes
  2. How much time the processes spend in MPI

- If we ignore the threading there are only three states, i.e.
  1. A process is in serial computation on the master thread: ***serial_comp***
  2. A process is inside an OpenMP parallel region: ***omp***
  3. A process is inside MPI and outside OpenMP

- When ignoring the threading we assume only time outside OpenMP and inside MPI is non-useful

Illustration: Process Efficiency

- We define ***serial_comp*** and ***omp*** as useful
  - Time in MPI and outside OpenMP is considered the bottleneck

useful* = *serial_comp* + *omp

- And we can define an ‘ideal runtime’ as **average(*useful*)**

Process Efficiency = average(*useful*) / *runtime*

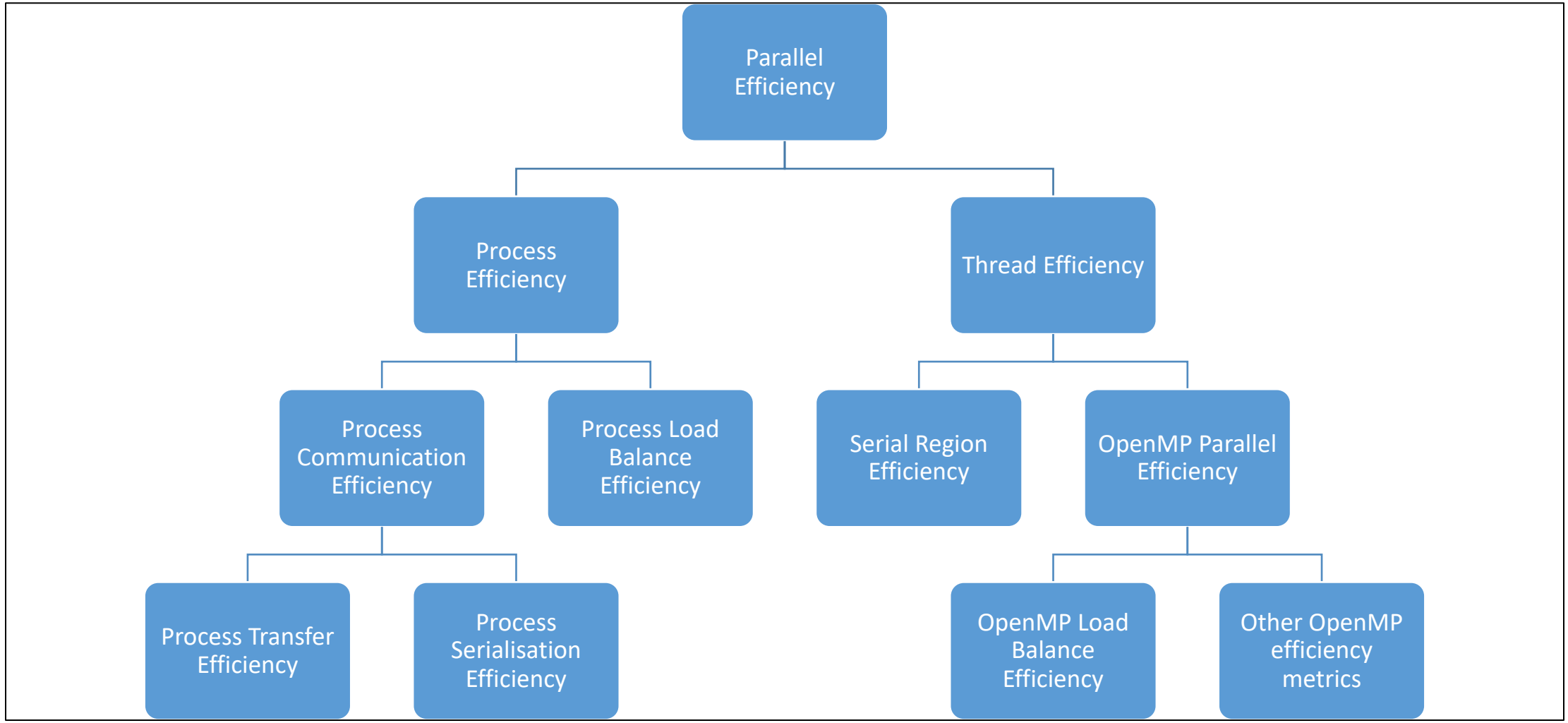
- We can use the additive methodology to define ideal run times that split Process Efficiency into
  - **Load Balance Efficiency** – cost of imbalance of ***useful*** over processes
  - **Transfer Efficiency** – cost of MPI time due to data transfer over network
  - **Serialisation Efficiency** – remaining cost of time in MPI

Thread Efficiencies

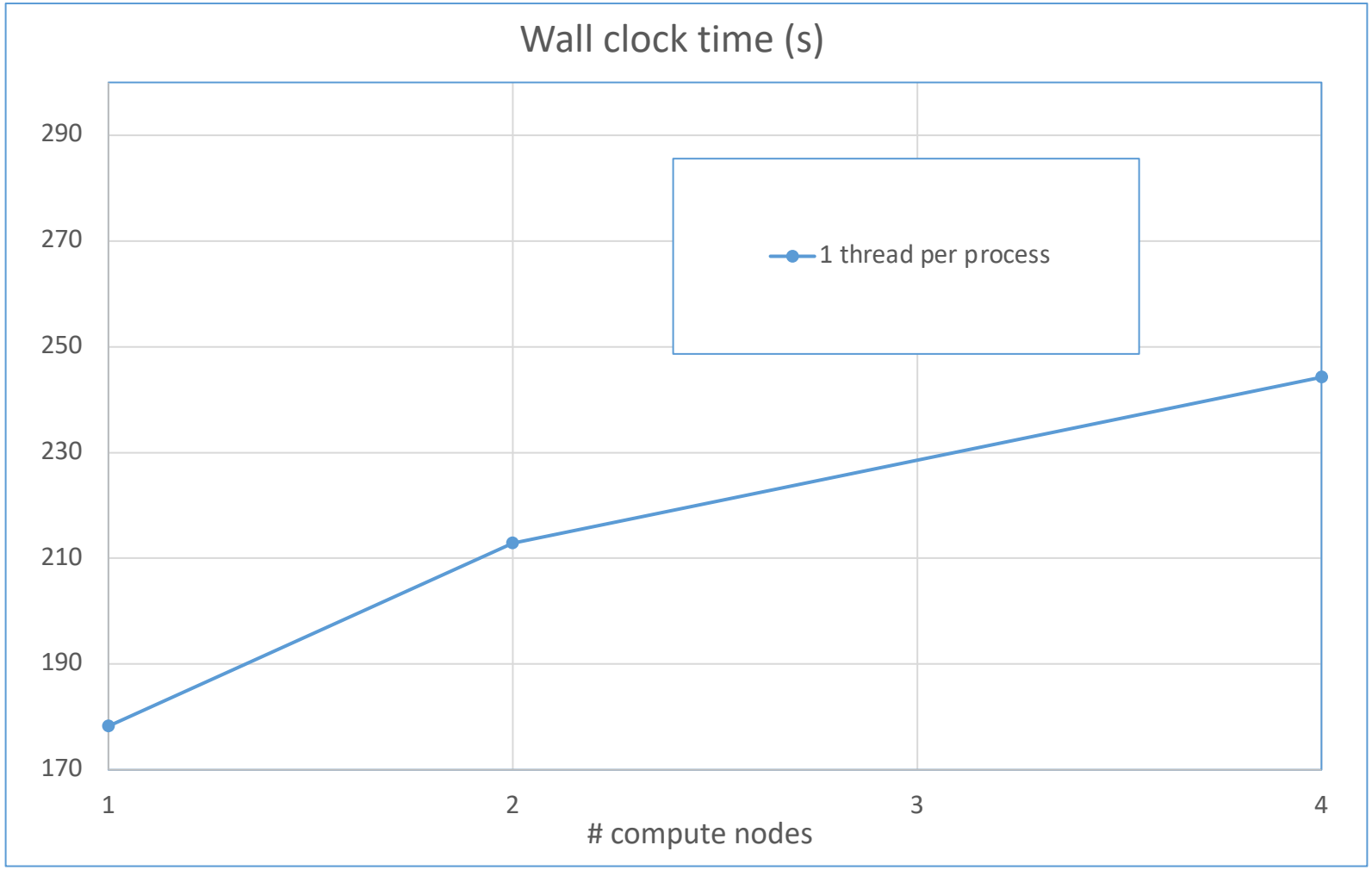
- We can define ideal runtimes which define
  - **Thread Efficiency** – cost of remaining non-useful time
  - Thread Efficiency sub metrics
    - **Serial Region Efficiency** – cost of computation outside OpenMP
    - **OpenMP Parallel Efficiency** – cost of time outside useful computation in OpenMP regions

- And we can also split OpenMP Parallel Efficiency e.g.
  - A contribution per OpenMP bottleneck
    - e.g. cost of thread load imbalance within OpenMP
  - Inefficiency contributions per OpenMP parallel region

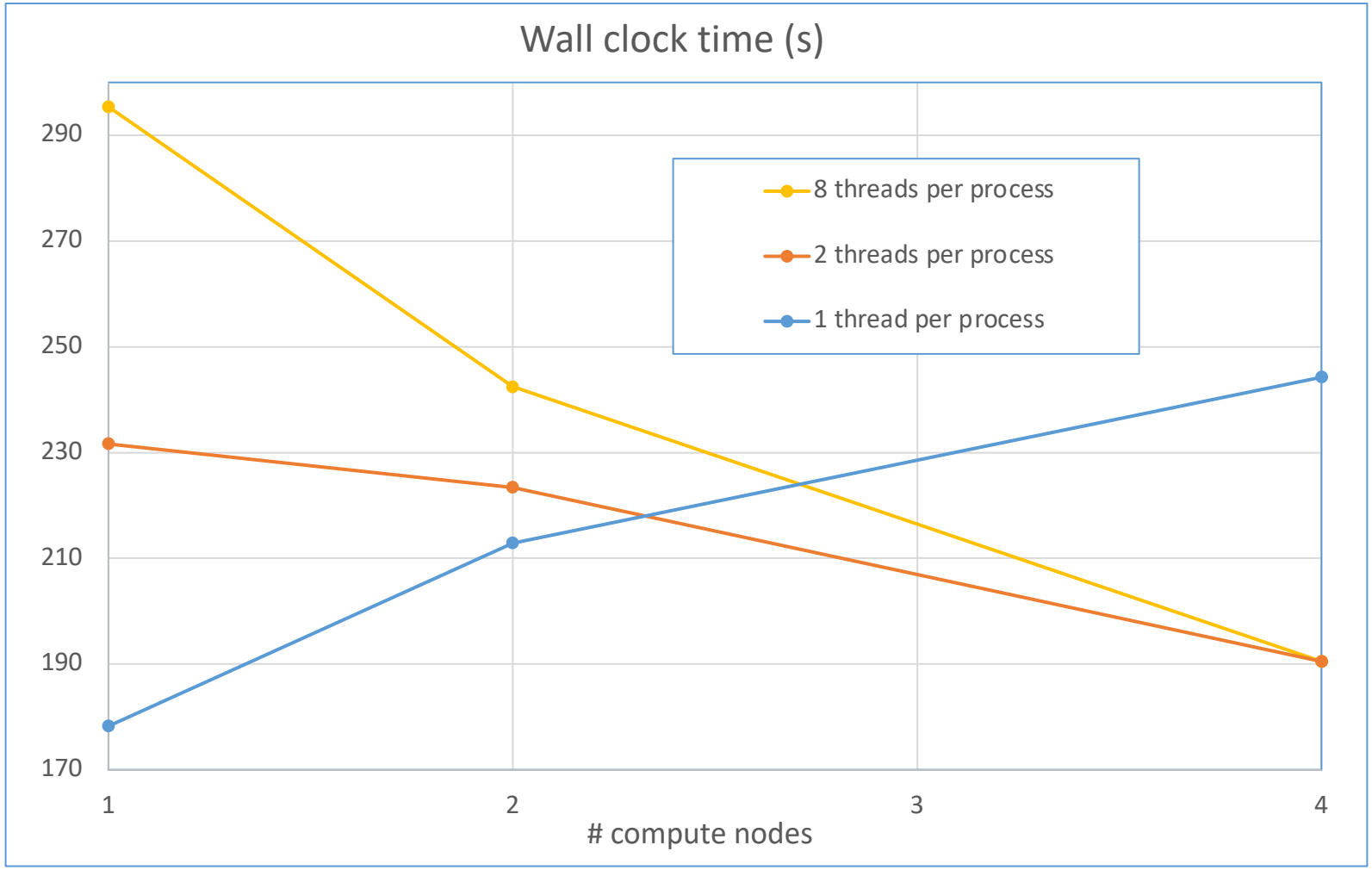
Additive metrics hierarchy



Example (strong scaling)



What is going on?



POP metrics to the rescue!

Number of compute nodes	1			2			4		
Number of Processes	48	24	6	96	48	12	192	96	24
Threads per Process	1	2	8	1	2	8	1	2	8
Total Threads	48	48	48	96	96	96	192	192	192
Speedup	1.00	0.77	0.60	0.84	0.80	0.74	0.73	0.94	0.94
Global Efficiency	0.50	0.39	0.30	0.21	0.20	0.19	0.09	0.12	0.12
↳ Parallel Efficiency	0.50	0.32	0.22	0.24	0.18	0.13	0.13	0.12	0.09
↳ Process Efficiency	0.51	0.42	0.57	0.24	0.24	0.33	0.13	0.17	0.22
↳ Process Load balance	0.97	0.98	0.99	0.98	0.99	0.99	0.99	0.99	0.99
↳ Process Communication Eff.	0.54	0.44	0.58	0.26	0.26	0.35	0.13	0.18	0.24
↳ Process Transfer Efficiency	0.55	0.45	0.59	0.29	0.27	0.36	0.16	0.20	0.25
↳ Process Serialisation Eff.	0.98	0.99	0.99	0.97	0.98	0.99	0.97	0.98	0.99
↳ Thread Efficiency	1.00	0.90	0.65	1.00	0.93	0.80	1.00	0.95	0.86
↳ OpenMP Parallel Efficiency	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
↳ Serial Region Efficiency	1.00	0.90	0.65	1.00	0.94	0.80	1.00	0.95	0.86
↳ Computational Scaling	1.00	1.22	1.37	0.88	1.12	1.39	0.73	0.98	1.36
↳ Instruction Scaling	1.00	1.04	1.05	0.91	1.00	1.04	0.75	0.91	1.02
↳ IPC Scaling	1.00	1.16	1.29	0.98	1.11	1.32	0.99	1.07	1.33
↳ Frequency Scaling	1.00	1.02	1.01	0.99	1.02	1.01	0.98	1.00	1.01

POP metrics to the rescue!

Number of compute nodes	1			2			4		
Number of Processes	48	24	6	96	48	12	192	96	24
Threads per Process	1	2	8	1	2	8	1	2	8
Total Threads	48	48	48	96	96	96	192	192	192
Speedup	1.00	0.77	0.60	0.84	0.80	0.74	0.73	0.94	0.94
Global Efficiency	0.50	0.39	0.30	0.21	0.20	0.19	0.09	0.12	0.12
↳ Parallel Efficiency	0.50	0.32	0.22	0.24	0.18	0.13	0.13	0.12	0.09
↳ Process Efficiency	0.51	0.42	0.57	0.24	0.24	0.33	0.13	0.17	0.22
↳ Process Load balance	0.97	0.98	0.99	0.98	0.99	0.99	0.99	0.99	0.99
↳ Process Communication Eff.	0.54	0.44	0.58	0.26	0.26	0.35	0.13	0.18	0.24
↳ Process Transfer Efficiency	0.55	0.45	0.59	0.29	0.27	0.36	0.16	0.20	0.25
↳ Process Serialisation Eff.	0.98	0.99	0.99	0.97	0.98	0.99	0.97	0.98	0.99
↳ Thread Efficiency	1.00	0.90	0.65	1.00	0.93	0.80	1.00	0.95	0.86
↳ OpenMP Parallel Efficiency	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
↳ Serial Region Efficiency	1.00	0.90	0.65	1.00	0.94	0.80	1.00	0.95	0.86
↳ Computational Scaling	1.00	1.22	1.37	0.88	1.12	1.39	0.73	0.98	1.36
↳ Instruction Scaling	1.00	1.04	1.05	0.91	1.00	1.04	0.75	0.91	1.02
↳ IPC Scaling	1.00	1.16	1.29	0.98	1.11	1.32	0.99	1.07	1.33
↳ Frequency Scaling	1.00	1.02	1.01	0.99	1.02	1.01	0.98	1.00	1.01

POP metrics to the rescue!

Number of compute nodes	1			2			4		
Number of Processes	48	24	6	96	48	12	192	96	24
Threads per Process	1	2	8	1	2	8	1	2	8
Total Threads	48	48	48	96	96	96	192	192	192
Speedup	1.00	0.77	0.60	0.84	0.80	0.74	0.73	0.94	0.94
Global Efficiency	0.50	0.39	0.30	0.21	0.20	0.19	0.09	0.12	0.12
↳ Parallel Efficiency	0.50	0.32	0.22	0.24	0.18	0.13	0.13	0.12	0.09
↳ Process Efficiency	0.51	0.42	0.57	0.24	0.24	0.33	0.13	0.17	0.22
↳ Process Load balance	0.97	0.98	0.99	0.98	0.99	0.99	0.99	0.99	0.99
↳ Process Communication Eff.	0.54	0.44	0.58	0.26	0.26	0.35	0.13	0.18	0.24
↳ Process Transfer Efficiency	0.55	0.45	0.59	0.29	0.27	0.36	0.16	0.20	0.25
↳ Process Serialisation Eff.	0.98	0.99	0.99	0.97	0.98	0.99	0.97	0.98	0.99
↳ Thread Efficiency	1.00	0.90	0.65	1.00	0.93	0.80	1.00	0.95	0.86
↳ OpenMP Parallel Efficiency	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
↳ Serial Region Efficiency	1.00	0.90	0.65	1.00	0.94	0.80	1.00	0.95	0.86
↳ Computational Scaling	1.00	1.22	1.37	0.88	1.12	1.39	0.73	0.98	1.36
↳ Instruction Scaling	1.00	1.04	1.05	0.91	1.00	1.04	0.75	0.91	1.02
↳ IPC Scaling	1.00	1.16	1.29	0.98	1.11	1.32	0.99	1.07	1.33
↳ Frequency Scaling	1.00	1.02	1.01	0.99	1.02	1.01	0.98	1.00	1.01

What needs investigating further?

Number of compute nodes	1			2			4		
Number of Processes	48	24	6	96	48	12	192	96	24
Threads per Process	1	2	8	1	2	8	1	2	8
Total Threads	48	48	48	96	96	96	192	192	192
Speedup	1.00	0.77	0.60	0.84	0.80	0.74	0.73	0.94	0.94
Global Efficiency	0.50	0.39	0.30	0.21	0.20	0.19	0.09	0.12	0.12
↳ Parallel Efficiency	0.50	0.32	0.22	0.24	0.18	0.13	0.13	0.12	0.09
↳ Process Efficiency	0.51	0.42	0.57	0.24	0.24	0.33	0.13	0.17	0.22
↳ Process Load balance	0.97	0.98	0.99	0.98	0.99	0.99	0.99	0.99	0.99
↳ Process Communication Eff.	0.54	0.44	0.58	0.26	0.26	0.35	0.13	0.18	0.24
↳ Process Transfer Efficiency	0.55	0.45	0.59	0.29	0.27	0.36	0.16	0.20	0.25
↳ Process Serialisation Eff.	0.98	0.99	0.99	0.97	0.98	0.99	0.97	0.98	0.99
↳ Thread Efficiency	1.00	0.90	0.65	1.00	0.93	0.80	1.00	0.95	0.86
↳ OpenMP Parallel Efficiency	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
↳ Serial Region Efficiency	1.00	0.90	0.65	1.00	0.94	0.80	1.00	0.95	0.86
↳ Computational Scaling	1.00	1.22	1.37	0.88	1.12	1.39	0.73	0.98	1.36
↳ Instruction Scaling	1.00	1.04	1.05	0.91	1.00	1.04	0.75	0.91	1.02
↳ IPC Scaling	1.00	1.16	1.29	0.98	1.11	1.32	0.99	1.07	1.33
↳ Frequency Scaling	1.00	1.02	1.01	0.99	1.02	1.01	0.98	1.00	1.01

Single thread issues

Number of compute nodes	1			2			4		
Number of Processes	48	24	6	96	48	12	192	96	24
Threads per Process	1	2	8	1	2	8	1	2	8
Total Threads	48	48	48	96	96	96	192	192	192
Speedup	1.00	0.77	0.60	0.84	0.80	0.74	0.73	0.94	0.94
Global Efficiency	0.50	0.39	0.30	0.21	0.20	0.19	0.09	0.12	0.12
↳ Parallel Efficiency	0.50	0.32	0.22	0.24	0.18	0.13	0.13	0.12	0.09
↳ Process Efficiency	0.51	0.42	0.57	0.24	0.24	0.33	0.13	0.17	0.22
↳ Process Load balance	0.97	0.98	0.99	0.98	0.99	0.99	0.99	0.99	0.99
↳ Process Communication Eff.	0.54	0.44	0.58	0.26	0.26	0.35	0.13	0.18	0.24
↳ Process Transfer Efficiency	0.55	0.45	0.59	0.29	0.27	0.36	0.16	0.20	0.25
↳ Process Serialisation Eff.	0.98	0.99	0.99	0.97	0.98	0.99	0.97	0.98	0.99
↳ Thread Efficiency	1.00	0.90	0.65	1.00	0.93	0.80	1.00	0.95	0.86
↳ OpenMP Parallel Efficiency	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
↳ Serial Region Efficiency	1.00	0.90	0.65	1.00	0.94	0.80	1.00	0.95	0.86
↳ Computational Scaling	1.00	1.22	1.37	0.88	1.12	1.39	0.73	0.98	1.36
↳ Instruction Scaling	1.00	1.04	1.05	0.91	1.00	1.04	0.75	0.91	1.02
↳ IPC Scaling	1.00	1.16	1.29	0.98	1.11	1.32	0.99	1.07	1.33
↳ Frequency Scaling	1.00	1.02	1.01	0.99	1.02	1.01	0.98	1.00	1.01

Single node hybrid performance

Number of compute nodes	1			2			4		
Number of Processes	48	24	6	96	48	12	192	96	24
Threads per Process	1	2	8	1	2	8	1	2	8
Total Threads	48	48	48	96	96	96	192	192	192
Speedup	1.00	0.77	0.60	0.84	0.80	0.74	0.73	0.94	0.94
Global Efficiency	0.50	0.39	0.30	0.21	0.20	0.19	0.09	0.12	0.12
↳ Parallel Efficiency	0.50	0.32	0.22	0.24	0.18	0.13	0.13	0.12	0.09
↳ Process Efficiency	0.51	0.42	0.57	0.24	0.24	0.33	0.13	0.17	0.22
↳ Process Load balance	0.97	0.98	0.99	0.98	0.99	0.99	0.99	0.99	0.99
↳ Process Communication Eff.	0.54	0.44	0.58	0.26	0.26	0.35	0.13	0.18	0.24
↳ Process Transfer Efficiency	0.55	0.45	0.59	0.29	0.27	0.36	0.16	0.20	0.25
↳ Process Serialisation Eff.	0.98	0.99	0.99	0.97	0.98	0.99	0.97	0.98	0.99
↳ Thread Efficiency	1.00	0.90	0.65	1.00	0.93	0.80	1.00	0.95	0.86
↳ OpenMP Parallel Efficiency	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
↳ Serial Region Efficiency	1.00	0.90	0.65	1.00	0.94	0.80	1.00	0.95	0.86
↳ Computational Scaling	1.00	1.22	1.37	0.88	1.12	1.39	0.73	0.98	1.36
↳ Instruction Scaling	1.00	1.04	1.05	0.91	1.00	1.04	0.75	0.91	1.02
↳ IPC Scaling	1.00	1.16	1.29	0.98	1.11	1.32	0.99	1.07	1.33
↳ Frequency Scaling	1.00	1.02	1.01	0.99	1.02	1.01	0.98	1.00	1.01

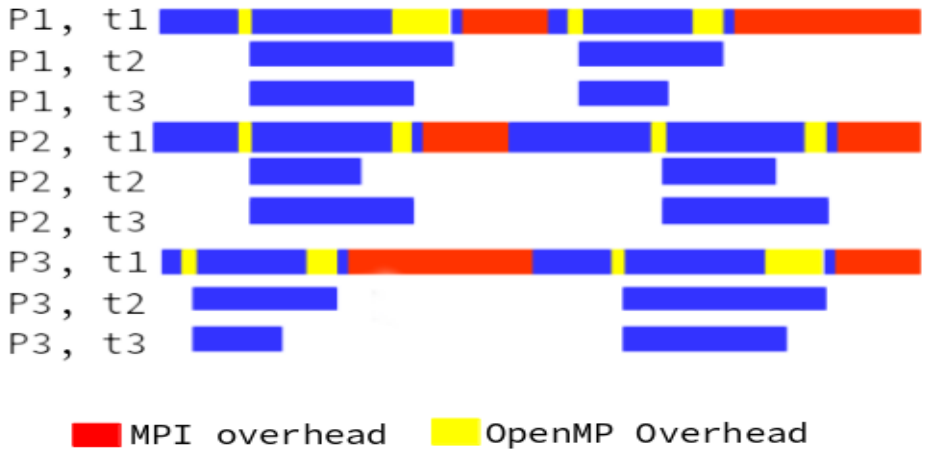
Single versus multiple node hybrid

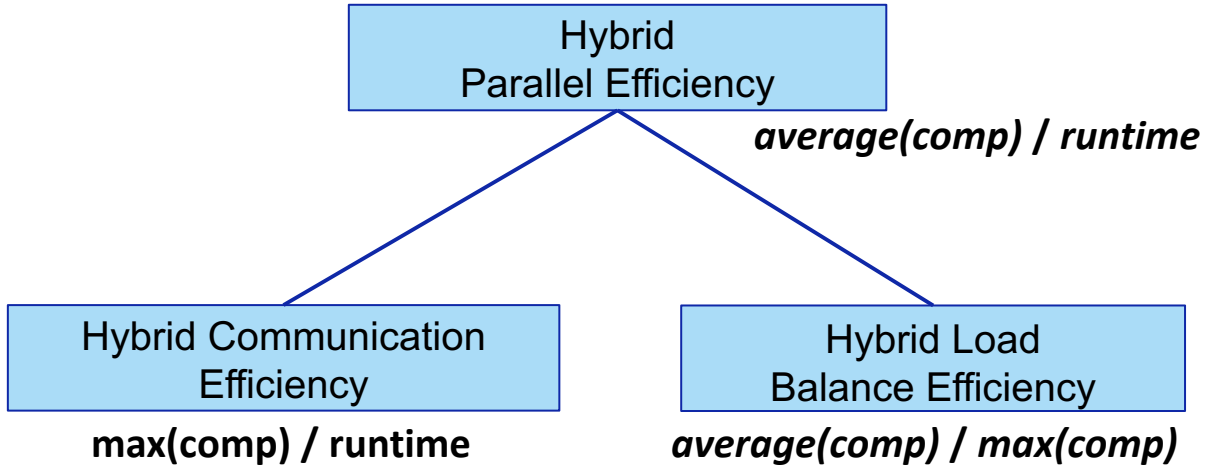
Number of compute nodes	1			2			4		
Number of Processes	48	24	6	96	48	12	192	96	24
Threads per Process	1	2	8	1	2	8	1	2	8
Total Threads	48	48	48	96	96	96	192	192	192
Speedup	1.00	0.77	0.60	0.84	0.80	0.74	0.73	0.94	0.94
Global Efficiency	0.50	0.39	0.30	0.21	0.20	0.19	0.09	0.12	0.12
↳ Parallel Efficiency	0.50	0.32	0.22	0.24	0.18	0.13	0.13	0.12	0.09
↳ Process Efficiency	0.51	0.42	0.57	0.24	0.24	0.33	0.13	0.17	0.22
↳ Process Load balance	0.97	0.98	0.99	0.98	0.99	0.99	0.99	0.99	0.99
↳ Process Communication Eff.	0.54	0.44	0.58	0.26	0.26	0.35	0.13	0.18	0.24
↳ Process Transfer Efficiency	0.55	0.45	0.59	0.29	0.27	0.36	0.16	0.20	0.25
↳ Process Serialisation Eff.	0.98	0.99	0.99	0.97	0.98	0.99	0.97	0.98	0.99
↳ Thread Efficiency	1.00	0.90	0.65	1.00	0.93	0.80	1.00	0.95	0.86
↳ OpenMP Parallel Efficiency	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
↳ Serial Region Efficiency	1.00	0.90	0.65	1.00	0.94	0.80	1.00	0.95	0.86
↳ Computational Scaling	1.00	1.22	1.37	0.88	1.12	1.39	0.73	0.98	1.36
↳ Instruction Scaling	1.00	1.04	1.05	0.91	1.00	1.04	0.75	0.91	1.02
↳ IPC Scaling	1.00	1.16	1.29	0.98	1.11	1.32	0.99	1.07	1.33
↳ Frequency Scaling	1.00	1.02	1.01	0.99	1.02	1.01	0.98	1.00	1.01

Hybrid metrics method 2

The POP ‘multiplicative’ scheme for MPI + OpenMP

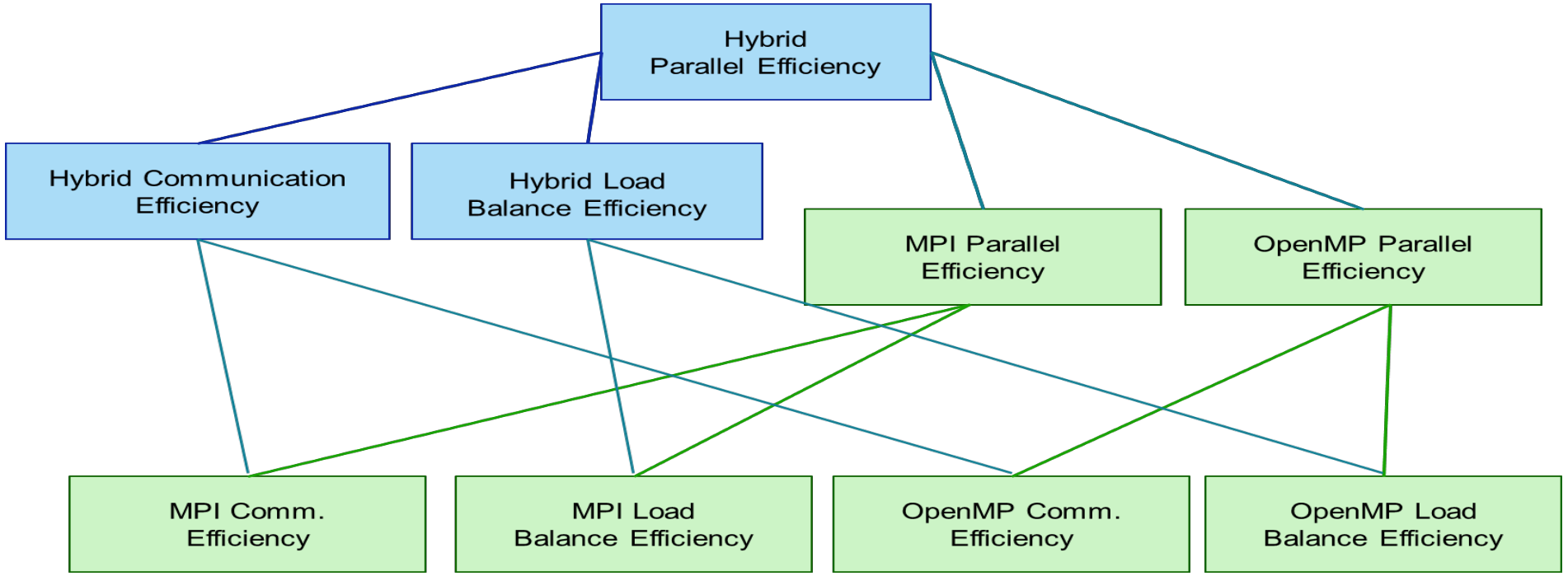
Applying the original model to hybrid codes





- Efficiency computed as percentage of time outside the two parallel runtimes
- Useful as a first step to distinguish between Load Balance and Communication. But how to dig down?
  - Efficiencies are mixing inefficiencies from MPI and OpenMP → Need to distribute the blame between the two programming models

What do we want?



- Global load balance and communication concepts can be mapped to any parallel programming paradigm
- The efficiencies at hybrid level collapse the contributions from the two programming models

Isolating MPI

- First target: hierarchical codes where MPI is the upper level (approach for 90% of the hybrid codes). From the MPI point of view, OpenMP runtime is as useful as computation:

MPI Parallel Efficiency = average*(out_MPI)* / *runtime*

Duration of the computing regions

Hybrid efficiencies [%]
- Parallel efficiency 47.07
- Load Balance 52.09
- Communication 90.37

MPI calls

MPI efficiencies [%]
- Parallel efficiency 95.97
- Load Balance 98.87
- Communication 97.06

OpenMP efficiencies

- Whatever cannot be blamed on MPI is caused by OpenMP. For example:

OpenMP Parallel Eff. = Hybrid Parallel Eff. / MPI Parallel Eff.

Duration of the computing regions

OpenMP efficiencies [%]

Parallel efficiency 49.05

Load Balance 52.69

Communication 93.11

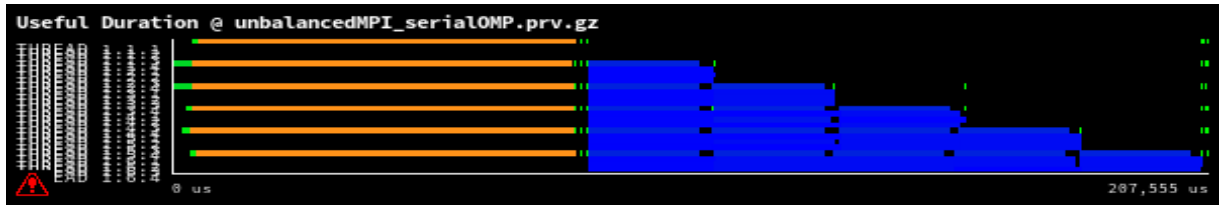
OpenMP parallel functions

→ The model is properly reporting that the main problem is imbalance in the OpenMP parallelization – large computation on the left only executed by the master thread.

Validating the source of imbalance

- Synthetic code with two very different phases

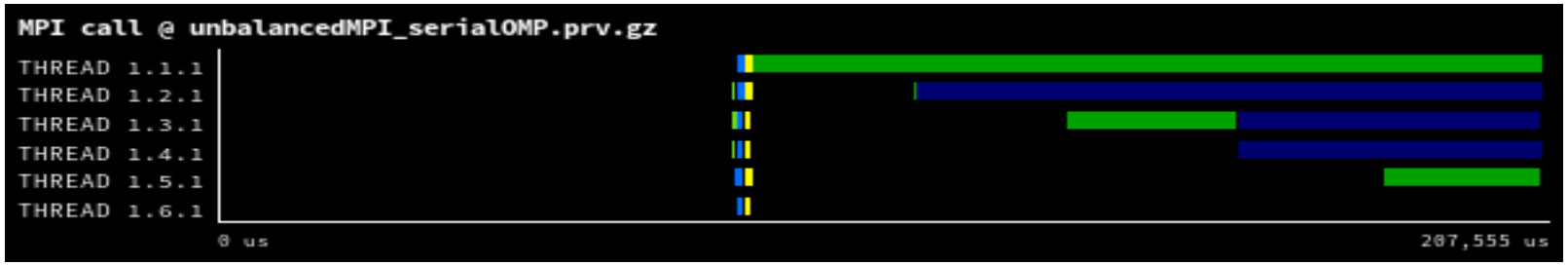
Duration of the computing regions



phase 1 *phase 2*

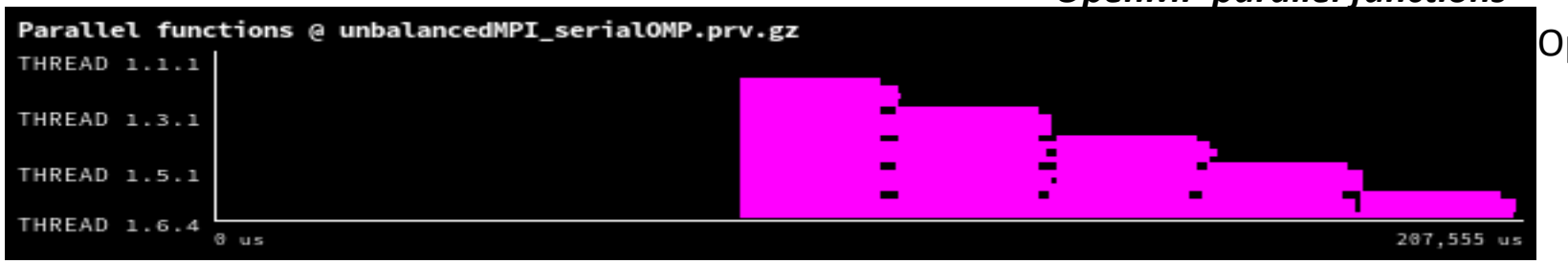
Hybrid efficiencies [%]
- Parallel efficiency 43.68
- Load Balance 48.46
- Communication 90.14

MPI calls



MPI efficiencies [%]
- Parallel efficiency 69.12
- Load Balance 70.01
- Communication 98.6

OpenMP parallel functions



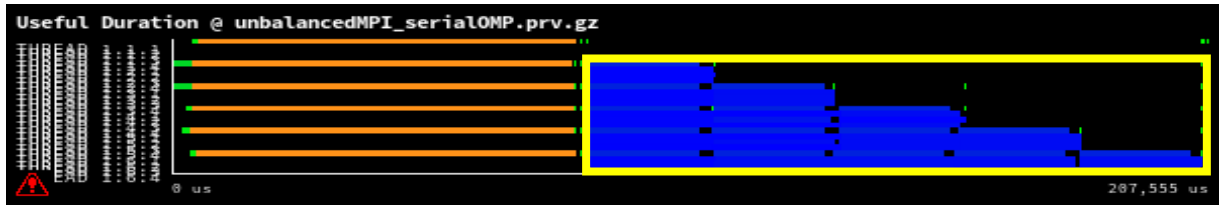
OpenMP efficiencies [%]
- Parallel efficiency 63.19
- Load Balance 69.22
- Communication 91.42

Phase 1 → OpenMP imbalance (previous slide), phase 2 → MPI imbalance?

Validating the source of imbalance

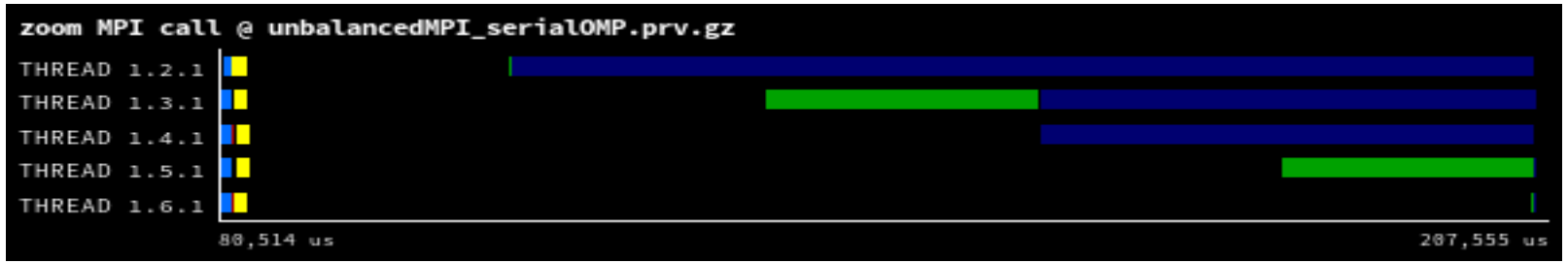
- Zooming in the second phase we can validate our guess

Duration of the computing regions



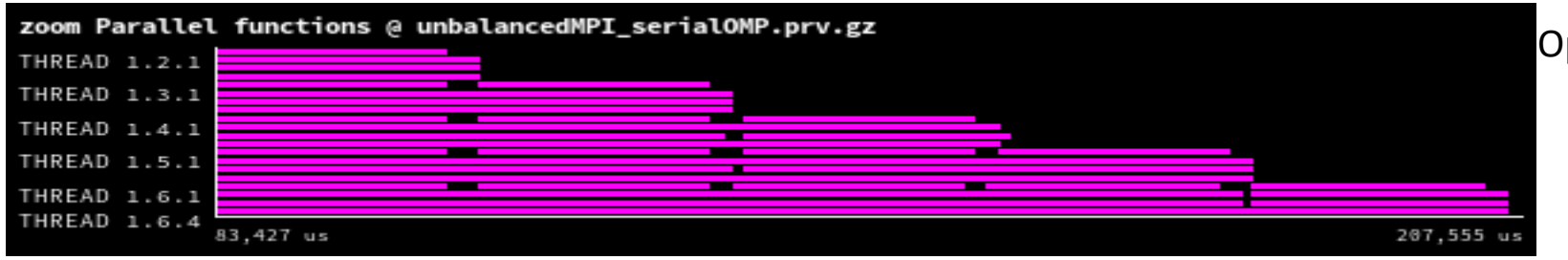
Hybrid efficiencies [%]
- Parallel efficiency 55.95
- Load Balance 58.78
- Communication 95.19

MPI calls



MPI efficiencies [%]
- Parallel efficiency 59.83
- Load Balance 61.20
- Communication 97.74

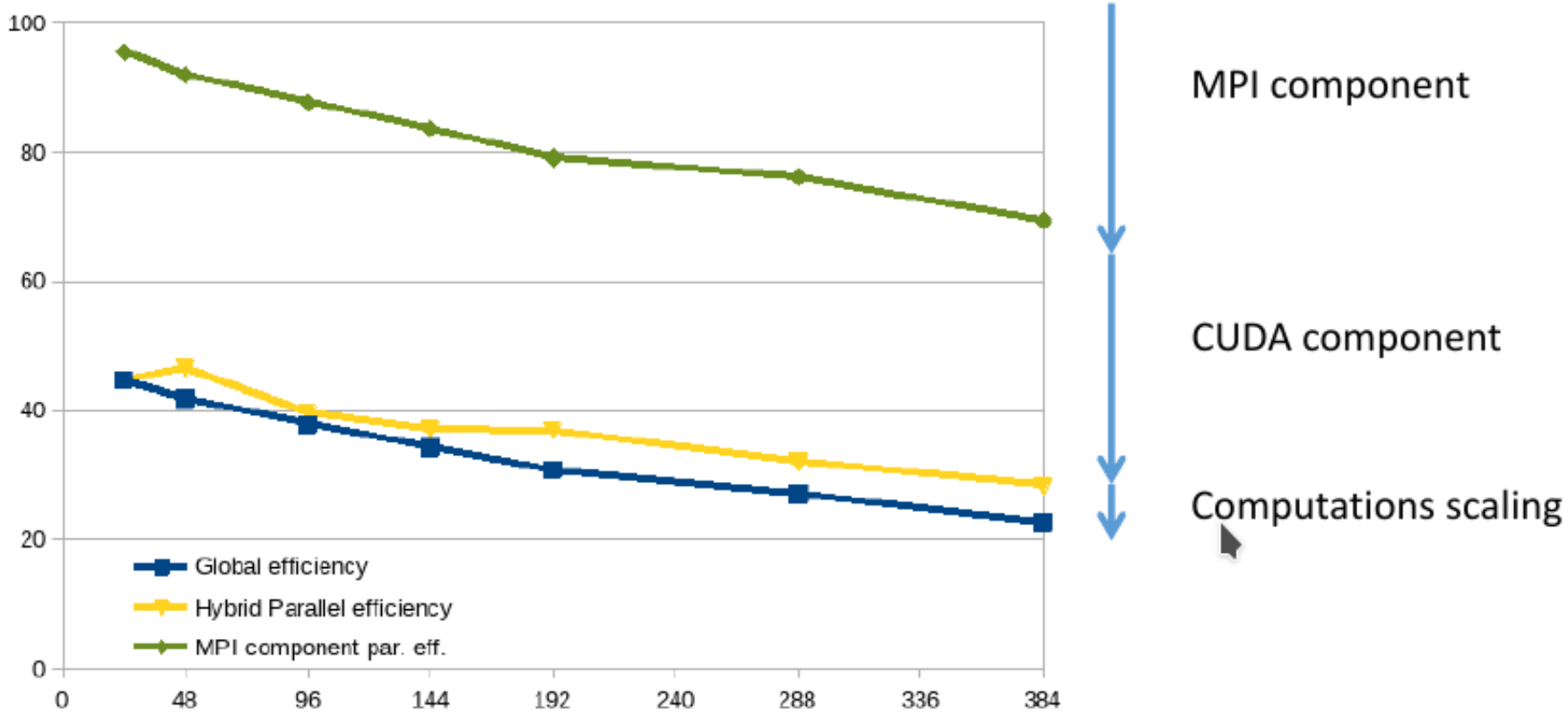
OpenMP parallel functions



OpenMP efficiencies [%]
- Parallel efficiency 93.51
- Load Balance 96.05
- Communication 97.39

→ Phase 2 only reports MPI imbalance as expected

Same approach applied to MPI+CUDA



- Bigger impact from CUDA component with a similar contribution in all the scales
- MPI contribution increases with the scale, but still lower than CUDA

How to calculate the metrics?

- We recommend the following Python tools to automatically calculate metrics from Extrae data
  - **NAG-PyPOP** for additive metrics (pip install [--user] NAG-PyPOP) (https://pypi.org/project/NAG-PyPOP)
  - **BSC's Basic Analysis** module for multiplicative metrics (https://tools.bsc.es/downloads)

- Scalasca + Cube have some POP metrics support (Score-P traces)
- Other tracing tools can be used as metrics can be calculated manually (see handouts for details)

- POP can do it for you!

Conclusions

- We presented two generic models to characterize the efficiency of hybrid codes by splitting the contribution of each programming model.
- Each model has its own strengths:
  - The additive approach is based on concepts specific to each programming model, making them easier to map to known problems.
  - Not focusing on the specifics of each parallel programming paradigm, the multiplicative approach allows us to use the same model with MPI+OpenMP, MPI+CUDA,…
- The two models are being used in the framework of the POP2 Center of Excellence (https://pop-coe.eu)

Performance Optimisation and Productivity

A Centre of Excellence in HPC

Contact:

- https://www.pop-coe.eu
- pop@bsc.es
- @POP_HPC
- youtube.com/POPHPC

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553 and 824080.