Introduction to the POP metrics

Jonathan Boyle, Numerical Algorithms Group

EU H2020 Centre of Excellence (CoE)

1 December 2018 – 30 November 2021

Grant Agreement No 824080

What you'll learn

Why it's difficult to understand poor parallel performance

- The limitations of traditional speed-up and efficiency plots
- What trace data is
- Challenges of interpreting trace data

The POP performance metrics

- The philosophy behind the POP metrics
- Introducing some POP metrics for general CPU parallelism
  - i.e. multiple threads and/or processes
- Additional metrics

The importance of performance

Q: Are we making good use of parallel hardware?

- **To speed up computation we run on multiple cores**
  - Modern processors are multicore e.g. desktops, mobile devices
  - Computers may contain multiple processors e.g. supercomputers
- **Huge speed ups are possible on large HPC machines**
  - Single processor speed-up is important too

Q: But is our speedup close to the maximum possible?

- Ideal speedup = number of CPU cores used
  - Relative to 1 core
- Anything less is a waste of resources
  - e.g. hardware, electricity, money

Traditional efficiency and scaling

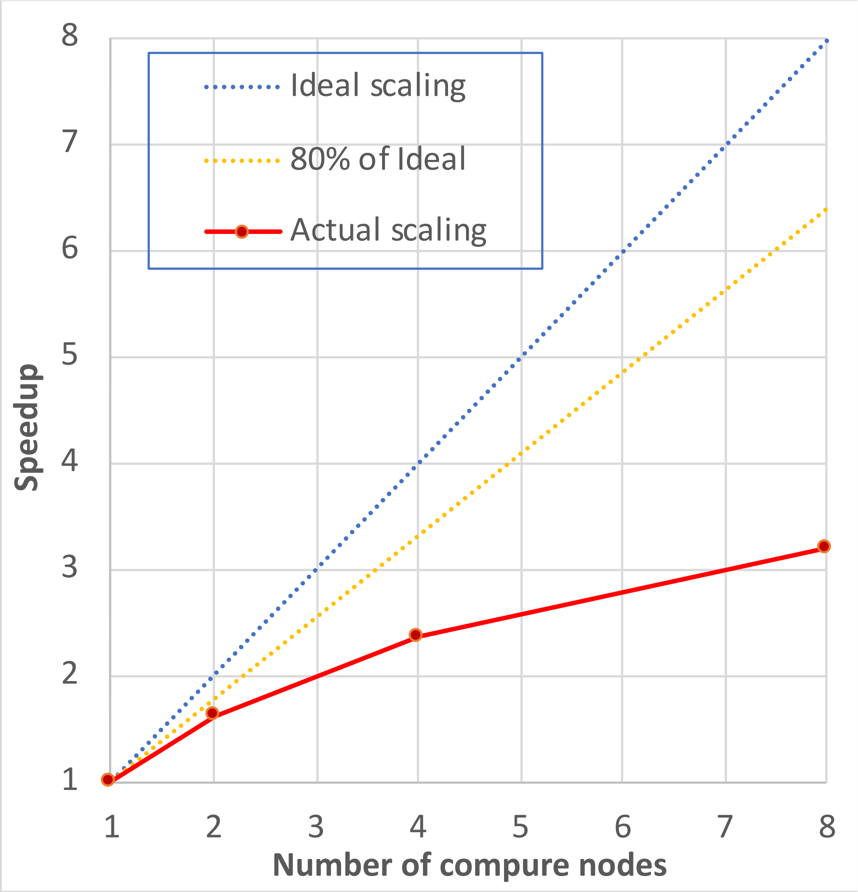
- We can plot **speed-up** or **efficiency** to measure **relative** performance, i.e.

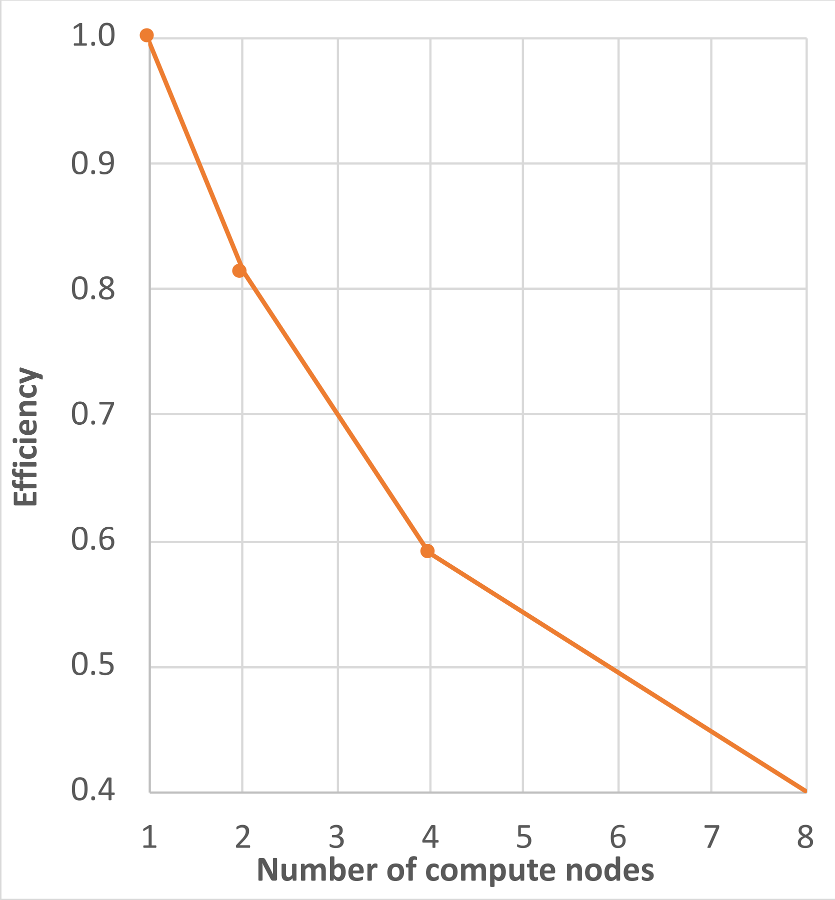
- **Step 1**: measure run time T_N for some range of N
  - N is usually number of compute nodes on HPC hardware
  - Or number of CPU cores

- **Step 2**: plot scaling or efficiency

$$\textbf{Speed-up} = T_{reference} / T_N$$
$$\textbf{Efficiency} = T_{reference} / (N \times T_N)$$

Traditional scaling & efficiency plots





- Reference case is a **68 core** compute node
  - Speed-up and efficiency is always 1 for reference case!!!

Measuring performance is difficult

- There is a problem using these **relative** scaling / efficiency plots

- **These metrics tell us nothing about a <u>parallel</u> reference case**
  - Scaling and efficiency for the reference case is always 1
  - But the reference case is often parallel itself **e.g. a multicore compute node**
- **They tell us nothing about absolute performance**
- **And they tell us nothing about the causes of poor performance**
  - e.g. load imbalance, idle cores, parallelism overheads, etc

- **So we add:**
  - **Step 3**: generate trace data to profile the performance
  - **Step 4**: interpret the trace data

What is trace data?

- Tracing tools record data **at specific points** during program execution
  - i.e. a timestamp plus various information about what's going on at that point
    - Tracing tools will vary in what they record
- For parallel performance typically record (at least) all parallel events
  - Also hardware counter data
    - e.g. number of processor cycles and instructions

- Trace files usually contain a huge amount of data
  - There are often many parallel events

- Trace visualisation tools display trace data, e.g.
  1. Timelines showing selected events per core – often too detailed to interpret
  2. Metrics – usually an overwhelming amount of data

Understanding trace data is usually a big challenge

Why analysing trace data is hard

- There are typically a huge number of parallel events during execution
- **The trace data is too complex to view on a single timeline**

- Some trace visualisation tools post-process trace data to calculate a range of metrics
- **But the amount of metrics and data is often overwhelming**

Q: How do we know where to start with the trace data? What are we looking for?

What do we need to know first?

What are the causes of poor performance? e.g.

- Imbalance in the amount of computation per core
- Dependencies between computation on different cores
  - e.g. synchronisation issues leading to idle cores
- Additional work from the parallelism e.g.
  - Useful work which can't be parallelised & must be replicated over the cores
  - Parallelism overheads
- Memory issues
  - e.g. NUMA (non-uniform memory access)
- Reduction in processor instruction throughput

Qs: Which are impacting performance? Which issues to fix first?

Recap: why does it matter?

- The hardware is often very expensive to use
- And improving parallel software can add a lot of value
  - Reduced expenditure
  - Faster results
  - Novel solutions

There is a lot of value in understanding and improving performance

We need a method to help us understand trace data!

A solution – The POP metrics

The idea is simple but extremely powerful

- Devise a simple set of performance metrics using values easily obtained from the trace data i.e.
    - Absolute efficiency metrics
    - Scaling metrics
- Low values indicate **specific** causes of poor parallel performance

We use these metrics to understand

1. What are the causes of poor performance
2. What to look for in the trace data

What do we want to know first?

- There are some obvious first questions
  1. How good is the parallelism?
  2. Is the total time in useful computation constant?
     - ‘Useful’ means computation outside parallel libraries i.e. executing your code

- And using a trace visualisation tool we can usually quickly find:
  1. **Sum of all time in useful computation**
  2. **Maximum time in useful computation over the cores**
  3. **Total number of processor useful cycles**
  4. **Total number of processor useful instructions**

What can we calculate using this data?

How good is the parallelism?

- Ideally we split useful computation evenly over all cores, with no overheads from parallelism i.e.

Ideal runtime = sum of time in useful comp / N_c
= average useful computation

N_c = number of cores

- Hence define: **Parallel Efficiency**
= Average useful computation / Runtime

- Note: this measures **absolute** efficiency

Is time in computation constant?

- Ideally the total useful computation remains constant as we increase the number of cores
  - But in practice total useful computation often increases

- Define:

Computational scaling =

Reference total useful comp / Total useful comp

Child-metrics for comp. scaling

- Define:

  Useful IPC = Useful instructions / Useful cycles
  Frequency = Useful cycles / Sum of useful computation

- Note: Time = Instructions / (IPC x Frequency)

- Hence, we can split computational scaling into 3 metrics
  1. Useful IPC scaling
  2. Useful instruction scaling
  3. Useful frequency scaling
- Multiplying these three scaling gives us computational scaling

How good is performance overall?

- We can also combine Parallel Efficiency and Computational Scaling by multiplying

Global efficiency = parallel efficiency x computational scaling

- This is the parallel efficiency that would be obtained if time in useful computation remained the same as the reference case

A hierarchy of metrics

- We now have a nice set of useful metrics
  - They can be used with MPI, OpenMP, Pthreads, etc.
- There is a hierarchy
  - **Global efficiency** splits into **parallel efficiency** & **computational scaling**
  - **Computational scaling** splits into **instructions**, **IPC** and **frequency scaling**
- The metrics give us insight into
  - Overall performance
  - Is the problem in the parallelism or the computation?
  - Is poor computational scaling due to
    - Increasing useful instructions
    - Reducing IPC
    - Reducing Frequency
- We can also use these for benchmarking
  - e.g. to compare performance before and after code modifications

Example of using the POP metrics

#nodes	1	2	4	8
Global efficiency	0.95	0.38	0.24	0.14
↳ Parallel efficiency	0.95	0.53	0.42	0.34
↳ Computational scaling	1.00	0.73	0.57	0.41
↳ IPC scaling	1.00	0.85	0.67	0.50
↳ Instructions scaling	1.00	0.94	0.95	0.94
↳ Frequency scaling	1.00	0.91	0.89	0.89

- We immediately see **parallel efficiency is very low** on > 1 compute node
  - Around 2/3 of run time on 8 nodes is overhead from poor parallelism
- **Computational scaling is also poor**
  - Time in useful computation on 8 nodes > twice that on 1 node
  - Caused mostly by poor IPC scaling
- **Instructions & frequency scaling is good**

Example 2

Number of nodes	1	2	4	8
Global efficiency	0.86	0.70	0.50	0.34
↳ Parallel efficiency	0.86	0.72	0.60	0.47
↳ Computational scaling	1.00	0.96	0.84	0.74
↳ IPC scaling	1.00	0.97	0.94	0.98
↳ Instructions scaling	1.00	0.96	0.88	0.76
↳ Frequency scaling	1.00	1.03	1.02	0.99

- The main problem here is the parallelism
  - 50% of the run time is due to parallelism inefficiencies
  - Our next question: what is causing this?
- Computational scaling is low
  - The instruction count is increasing!
- IPC and frequency scaling is good

Example 3

Number of CPU cores	1	4	8	12	16	20	24
Global efficiency	0.99	0.76	0.53	0.44	0.38	0.36	0.30
↳ Parallel efficiency	0.99	0.85	0.76	0.73	0.68	0.65	0.58
↳ Computational scaling	1.00	0.89	0.69	0.60	0.56	0.55	0.51
↳ IPC scaling	1.00	0.92	0.78	0.74	0.70	0.69	0.65
↳ Instructions scaling	1.00	1.00	1.00	1.00	1.00	0.99	0.99
↳ Frequency scaling	1.00	0.97	0.89	0.82	0.81	0.80	0.78

- Poor parallel efficiency yet again!
  - This needs further investigation
- And computational scaling contributes to 50% of run time on 24 cores
  - Main contribution is reducing IPC
  - But reducing processor frequency also plays a part

Q: Causes of low parallel efficiency?

- We next extend these metrics for specific parallel methodologies
  - i.e. split the parallel efficiency into suitable child metrics
    - Ideally one child metric per source of inefficiency
- For example we use metrics to analyse performance in:
  - **MPI** - we typically want to understand costs due to
    - Load imbalance
    - Time inside MPI
      - And is time inside MPI due to data transfer or wait states?
  - **OpenMP** - usually want to understand costs due to
    - Serial execution outside OpenMP regions (Amdahl's law)
    - Which are the inefficient OpenMP regions
      - And why these OpenMP regions are inefficient
- POP MPI & OpenMP metrics are topics for further training

POP tracing tools

We use the following tools (developed by some of the POP partners)

- Extrae (tracing) + Paraver (visualisation)
- Score-P (tracing) + Scalasca (post processing) + Cube (visualisation)
- PyPOP for automated generation of POP metrics from Extrae traces

To understand how to generate trace files & calculate POP metrics

- See POP website learning material & online training
    - https://pop-coe.eu/further-information/learning-material

Other tracing tools can be used e.g. Intel's VTune

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.