Additive performance metrics for MPI + OpenMP parallel code

The idea

1. Ideal runtime

Firstly, for each class of performance bottleneck we define an 'ideal' runtime.

This ideal runtime is the runtime which would be achieved if the bottleneck(s) were removed.

For example, consider **all** parallel bottlenecks. Assuming 1 thread per CPU core, if we define ***comp*** as time in useful computation on each core, then for perfect parallelisation on ***n*** cores, i.e. zero load imbalance and with zero overheads from the parallelism:

ideal_runtime **= sum(*comp*)/*n* = average(*comp*)**

2. Efficiency and inefficiency metrics

Secondly, after calculating an ideal runtime (e.g. from trace data) and measuring the actual runtime, for each performance bottleneck define the following metrics:

efficiency = ideal_runtime / runtime

inefficiency* = 1 - *efficiency

These metrics measure how far we are from the ideal performance.

For optimal performance, efficiency = 1 and inefficiency = 0.

For example, the efficiency which measures all parallel bottlenecks is defined by:

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

3. A metrics hierarchy

Thirdly, note we can split a class of bottlenecks into individual contributions, and define efficiency and inefficiency metrics for each of these contributions.

The sum of the 'child' inefficiencies from each contribution equals the 'parent' inefficiency, which is why we call these 'additive' metrics.

For example, we can split Parallel Efficiency into:

1. **Process Efficiency**, which measures efficiency of the processes, i.e. ignores threading bottlenecks.
2. **Thread Efficiency**, which measures efficiency of the threading, i.e. ignores process bottlenecks.

Scaling metrics

Scaling metrics are a different type of metric. They measure a quantity relative to a reference case.

The reference case is usually the smallest unit of computational hardware used, e.g. a single compute node.

If we define ***comp_ref*** as useful computation per core for the reference case, for strong scaling we can define the following top-level scaling metric:

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

A value < 1.0 tells us how much performance is degrading because total useful computation is increasing. We can define similar scaling metrics for ***IPC*** (instructions per cycle), **sum(*instructions*)** and ***frequency*** during useful computation, where:

Computation Scaling = IPC Scaling x Instruction Scaling x Frequency Scaling

Top level efficiency metrics

Parallel Efficiency measures cost of all bottlenecks arising from the parallelism.

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

Global Efficiency combines Parallel Efficiency and Computation Scaling. It uses an ideal runtime of **sum(*comp_ref*)/*n***, i.e. ideal parallelisation of the reference computation on n cores.

Global Efficiency = Parallel Efficiency x Comp. Scaling = [sum(*comp_ref*)/*n*] / *runtime*

Process Efficiency measures inefficiency from inter-process imbalance, and from time **outside** OpenMP and within MPI (inefficiency from time **inside** OpenMP and within MPI contributes to Thread Efficiency).

Therefore, Process Efficiency considers only time outside OpenMP and within MPI as non-useful. If ***serial_comp*** is time each process is doing useful computation outside OpenMP, and ***omp*** is time each process is inside OpenMP, then:

useful = serial_comp + omp

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

Thread Efficiency measures average cost of serial computation outside OpenMP plus average cost of time inside OpenMP when threads are not doing useful computation.

Thread Efficiency = [*runtime* - average(*useful*) + average(*comp*)] / *runtime*

Calculating the metrics

Most of the metrics can be calculated using values easily extracted from trace data, i.e.

- Time in useful computation (***comp***)
- Time in useful computation within OpenMP (***omp_comp***)
- Time in useful computation outside OpenMP on the master thread (***serial_comp***)
- Time in OpenMP (***omp***)

PyPOP automates metrics calculation from Extrae traces (Transfer and Serialisation Eff currently not supported for MPI inside OpenMP).

Thread Efficiency child metrics

OpenMP Parallel Efficiency measures the cost of bottlenecks within OpenMP. If ***omp_comp*** is useful computation within OpenMP, then:

Efficiency = [*runtime* - avg(*omp*) + avg(*omp_comp*)] / *runtime*

Serial Region Efficiency measures average cost of serial computation outside OpenMP. If n_t is number of threads per process then:

Efficiency = [*runtime* – avg(*serial_comp*) x (n_t-1) / n_t] / *runtime*

OpenMP Parallel Efficiency child metrics

OpenMP Parallel Efficiency can be split into a contribution per parallel region, if suitable trace data exists for analysis.

Alternatively OpenMP Parallel Efficiency can be split into a contribution per source of inefficiency, e.g. computational imbalance between the threads within OpenMP, or time spent in MPI, scheduling, synchronisation, etc.

PyPOP can be used to measure efficiency contributions per region, and contributions from OpenMP imbalance, using Extrae data.

Process Efficiency child metrics

Process Load Balance Efficiency measures cost of process imbalance.

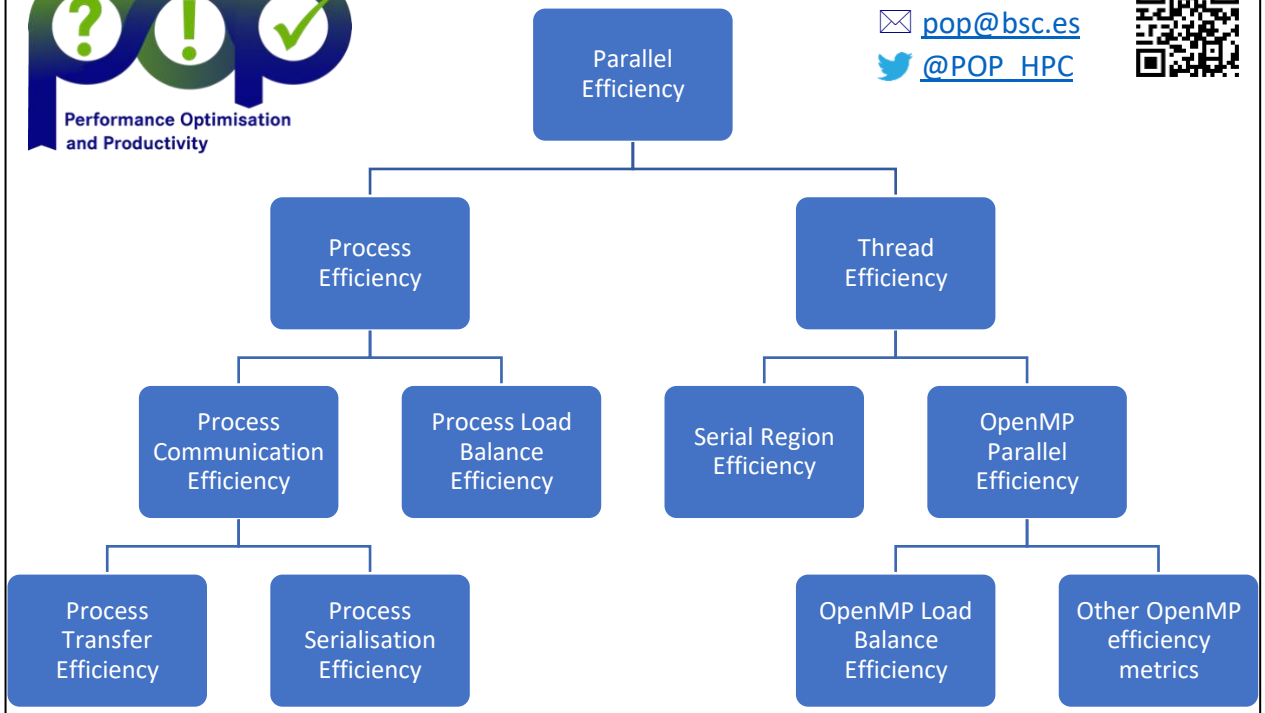
Efficiency = [*runtime* - max(*useful*) + avg(*useful*)] / *runtime*

Process Communication Efficiency measures the cost of adding MPI (outside OpenMP) from data transfer and waits due to dependencies.

Efficiency = max(*useful*) / *runtime*

Performance Optimisation and Productivity

www.pop-coe.eu
pop@bsc.es
@POP_HPC



Process Communication Efficiency child metrics

These metrics require being able to measure the time cost of MPI data transfer for MPI **outside** OpenMP on the process with **max(useful)**, i.e. on the process with minimum time in MPI outside OpenMP, since this is the process which defines the Process Communication Efficiency.

So, if mpi_p is the time each process spends in MPI and outside OpenMP, and mpi_ideal_p is this time on an ideal network with zero latency and infinite bandwidth, then we need to find $\min(mpi_p)$ and $\min(mpi\_ideal_p)$. Dimemas can be used to find mpi_ideal_p from Extrae traces. Then:

Process Transfer Efficiency measures cost of MPI (outside OpenMP) due to network data transfer.

Efficiency = [runtime - min(mpi_p) + min(mpi_ideal_p)] / *runtime*

Process Serialisation Efficiency measures time cost of MPI dependencies (for MPI outside OpenMP).

Efficiency = [runtime - min(mpi_ideal_p)] / *runtime*

The time cost of MPI **inside** OpenMP is treated as a bottleneck by OpenMP Parallel Efficiency, this is because of the difficulty of defining an ideal runtime for MPI inside OpenMP.

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