Introduction

The classic POP MPI metrics have been around for several years now and are a powerful and innovative tool for identifying MPI performance bottlenecks. With the advent of the additive metrics for MPI + OpenMP, this guide has been written as a reference to explain what each of the additive hybrid metrics tell us. The additive hybrid metrics hierarchy is shown in Figures 1-3.

Additive Efficiency and Inefficiency metrics

Efficiency tells us what fraction of our actual execution time would remain after removing a specific bottleneck or set of bottlenecks.

Inefficiency tells us what fraction of our actual execution time would be removed by eliminating a specific bottleneck or set of bottlenecks.

Top Level Metrics

Parallel Efficiency tells us what fraction of our measured execution time would remain after removing all bottlenecks caused by the parallelisation of the code.
If Parallel Efficiency is low, look at the sub-metrics to identify if the problem lies within the process parallelism, the thread parallelism, or both.
Child metrics: Process Efficiency and Thread Efficiency.

Computation Scaling tells us if time in useful computation is increasing or decreasing. It measures ratio of total useful computation in a reference case relative to the actual total useful computation.
Useful computation is the time executing useful user code, i.e. excluding time when threads or processes are idle, and excluding time in the overheads of parallelisation, e.g. MPI calls, OpenMP thread synchronisation, etc.
Values less than one indicate total useful computation is increasing, i.e. performance is reducing. If Computation Scaling is low, look at the sub-metrics to identify the relative contributions from increasing instruction count, reducing IPC (instructions per cycle) or reducing frequency (cycles per time).
Child metrics: Instruction Scaling, IPC Scaling, Frequency Scaling.

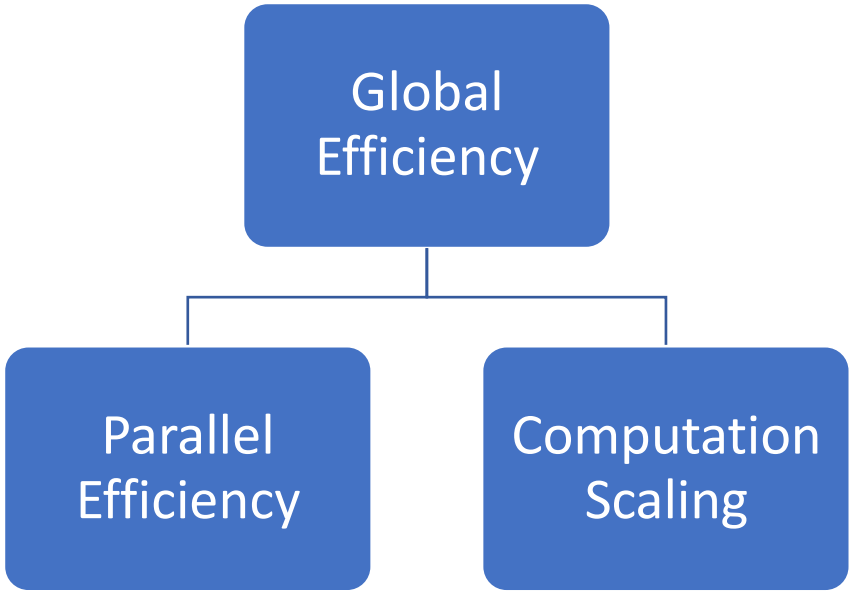

Figure 1: Top level metrics

Global Efficiency tells us what fraction of actual execution time would remain if all parallel bottlenecks were removed and assuming the total time in useful computation to be the same as in the Computational Scaling reference case.
Global Efficiency is the product of Parallel Efficiency and Computation Scaling. It is therefore the only efficiency metric in this scheme which can be larger than one, since Computation Scaling can be greater than one. This can be avoided by choosing the Computation Scaling reference case to be that with the smallest amount of useful computation.
If the value of Global Efficiency is low, look at the sub-metrics to identify if the cause is within the parallelisation, or due to an increase in useful computation, or both.
Child metrics: Parallel Efficiency and Computation Scaling.

Computation Scaling sub-metrics

IPC, Frequency and Instruction Scaling measure the ratios of useful instructions per cycle, useful cycles per time and sum of useful instructions, relative to the Computational Scaling reference case. Values less than one indicate reducing performance, i.e. reducing IPC, reducing frequency, or increasing instructions.

If Instruction Scaling is low, try to identify where in the execution this occurs. Possible causes are increasing iteration counts, e.g. in an iterative solver with degrading convergence; computation which is replicated over each process or thread; or reducing vectorisation.

Low IPC Scaling (and low IPC in general) is typically caused by memory bottlenecks, i.e. cycles are wasted waiting for data to arrive from main memory or lower-level caches. This should be investigated using suitable tools, e.g. Intel's VTune, or using PAPI to access hardware counters.

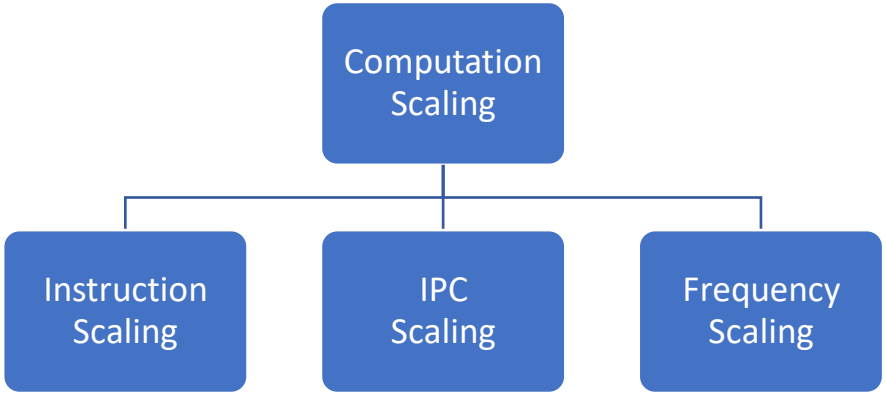

Figure 2: Computation Scaling sub-metrics

Low Frequency Scaling may be hard to fix, as it is related to processor power usage. However, it is useful to identify this contribution to Computation Scaling, in order to understand the impact.

Mid-Level Efficiency Metrics

Process Efficiency ignores threading by treating the following as useful:

- Time in OpenMP parallel regions
- Time in useful computation outside OpenMP

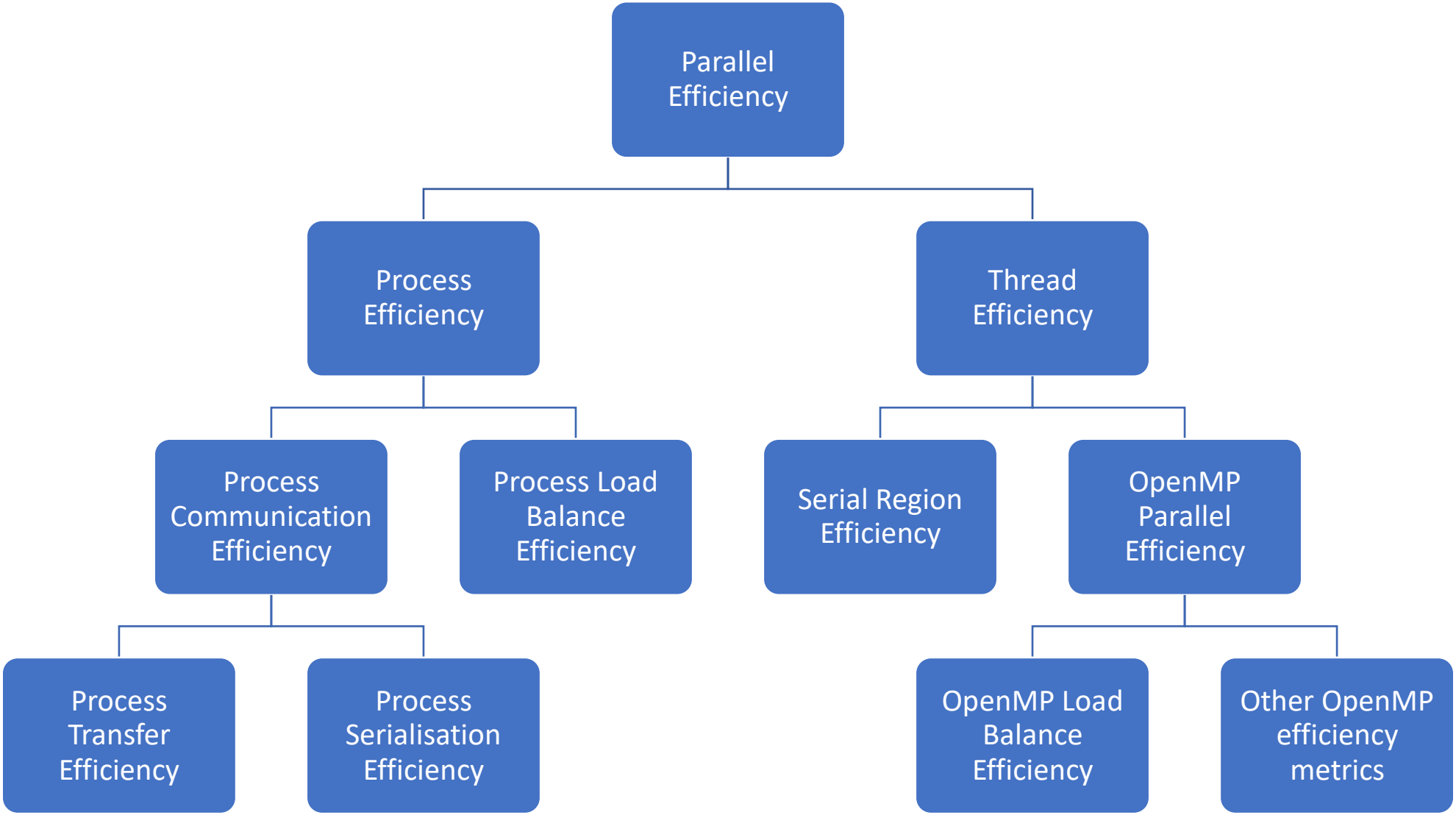

Figure 3: Parallel Efficiency sub-metrics

Hence it measures inefficiency arising from the time processes spend within MPI and outside OpenMP due to:

- Process imbalance of total useful work
- Process MPI data transfer
- Process serialisation due to dependencies.

Inefficiency due to MPI inside OpenMP is ignored, as it is treated as a thread inefficiency.
If Process Efficiency is low, look at the sub-metrics to identify whether to investigate process imbalance, or to investigate where processes spend time in MPI due to data transfer or dependencies.
Child metrics: Process Load Balance & Process Communication Efficiency.

Thread Efficiency measures inefficiency arising from idle CPU cores during serial computation outside OpenMP, and from any time outside useful computation within OpenMP parallel regions.
If this metric is low, use the sub-metrics to identify if the cause is due to serial computation outside OpenMP (i.e. Amdahl's Law) or due to inefficiencies within the OpenMP parallel regions.
Child metrics: Serial Region Efficiency & OpenMP Parallel Efficiency

Process low-level metrics

These metrics consider inefficiency arising from process imbalance and by MPI outside OpenMP. Inefficiency due to MPI inside OpenMP is ignored and treated as a thread inefficiency.

Process Load Balance Efficiency measures the inefficiency arising from imbalance in the distribution of total useful work over the processes, where useful work is defined as the following two states:

- Time in OpenMP
- Time in useful computation outside OpenMP.

It is possible, but unlikely, that the sum of the time in these two states is balanced with imbalance in each individual state.
This metric measures an average imbalance cost, i.e. it ignores the cost of any imbalances / serialisations which average out over the full execution time. Essentially, this metric measures the cost from the process imbalance that would occur if all MPI dependencies between processes were removed. See the section on serialisation and load balance efficiency for details.
If Process Load Balance Efficiency is low, first identify if the imbalance lies within OpenMP, or in computation outside OpenMP, or both. Then try to identify where within the execution this occurs, e.g. in which OpenMP regions. It may be useful to also measure the imbalance of useful computation over the processes.

Process Communication Efficiency measures the time cost introduced by MPI communications, i.e. the time processes spend in MPI due to data transfer over the network and due to MPI dependencies. It will exclude the inefficiency arising from time in MPI measured by Process Load Balance Efficiency (see the section on serialisation and load balance efficiency for details).
If this value is low, look at the sub-metrics to identify if the inefficiency results from data transfer over the network or is due to dependencies.
Child metrics: Process Transfer Efficiency and Process Serialisation Efficiency.

Process Transfer Efficiency measures process inefficiency arising from data transfer over the network, i.e. inefficiencies that would

be removed if data transfer on the network was instantaneous. It includes time in MPI where data transfer is occurring, and time waiting in MPI calls where the wait states are a result of data transfer on other processes.
If this metric is low, explore the MPI communication patterns and try to identify if the issue is due to latency or bandwidth. A comparison of simulated ideal network trace data with actual trace data can help identify regions of execution with large amounts of time in data transfer. Ideal network traces can be generated from Extrae traces using Dimemas.

Process Serialisation Efficiency measures the process inefficiency arising from time in MPI caused by dependencies, i.e. time in MPI after excluding inefficiency due to data transfer and average imbalance of useful work. See the section on serialisation and load balance efficiency for more details.
If this value is low, try to identify where time is spent in MPI which isn't caused by data transfer or average imbalance, e.g. identify where time in MPI occurs in an Extrae ideal timeline generated by Dimemas.

Thread low-level metrics

Serial region efficiency measures the cost of useful computation outside OpenMP where cores associated with slave threads are idle. The value is the inefficiency averaged over the processes.
If this value is low, try to identify regions of serial execution where OpenMP can be added. VTune is a useful tool for identifying these regions.

OpenMP Parallel Efficiency measures the cost of inefficiencies within OpenMP parallel regions, averaged over the processes. Inefficiency is any time spent outside useful computation within an OpenMP parallel region and can include time in MPI.
If this value is low, look at the OpenMP Parallel Efficiency values for each region to identify which regions contribute most to the inefficiency.
Also, look at the OpenMP Load Balance Efficiency and other OpenMP Parallel Efficiency sub-metrics to identify if the problem is load imbalance within OpenMP, or something else.

OpenMP Parallel Efficiency per region measures the individual contributions to OpenMP Parallel Efficiency from each OpenMP region. If a value is low, examine the source code and trace data for the region to identify the cause.

OpenMP Load Balance Efficiency measures the time cost of computational imbalance within OpenMP, averaged over the processes.
If this value is low, look at the OpenMP Load Balance Efficiency value for each OpenMP region to identify where in the execution the imbalance occurs. Try alternative load balancing strategies for this region, e.g. dynamic scheduling, loop collapsing.

Other OpenMP Parallel Efficiency sub-metrics can be calculated based on average values (over all threads) of non-useful time within OpenMP per source of inefficiency, e.g.

- OpenMP MPI Efficiency
- OpenMP Synchronisation Efficiency
- OpenMP Scheduling and Fork/Join Efficiency

If these values are low for a specific OpenMP region, examine the source code and trace data to understand why.

Serialisation and load balance metrics

There are subtleties which are important to understand when it comes to using the load balance and serialisation efficiency metrics described here. As an illustration, consider the artificial case in Figure 4, which shows execution on two CPU cores. The first core spends 10s in useful computation, then 10s waiting to synchronise with the execution on the second core, followed by 20s in useful computation and negligible time waiting at a second synchronisation point. The second core spends 20s in useful computation, then negligible time at the first synchronisation point, followed by 10s in useful computation and 10s waiting for the second synchronisation.
Is this load imbalance, or is this serialisation?
There is no clear answer to this question, as it depends on personal preference and on context. In this example, the total time in useful computation is balanced, as each core spends 30s in computation, so we might say this case has perfect load balance. However, it is also valid to say that in each 20s segment there is imbalance, or perhaps serialisation. Do we want to measure imbalance between the start and the end of the execution, or over sub-regions of execution defined by specific synchronisation points? And how do we differentiate between imbalance and serialisation?
First of all, let's consider OpenMP. If the pattern represents two 20s OpenMP parallel regions, as in figure 5, it is natural to describe the inefficiency as imbalance within OpenMP. Now consider the OpenMP case in figure 6, with the same pattern of computation and waiting states, but with the last 20s of execution now consisting of a 10s OpenMP region and 10s of serial computation outside OpenMP. It is natural now to describe the inefficiency as a combination of OpenMP imbalance from the first OpenMP region, and serialisation outside OpenMP in the last 10s of execution, and to think of the second OpenMP region as balanced.

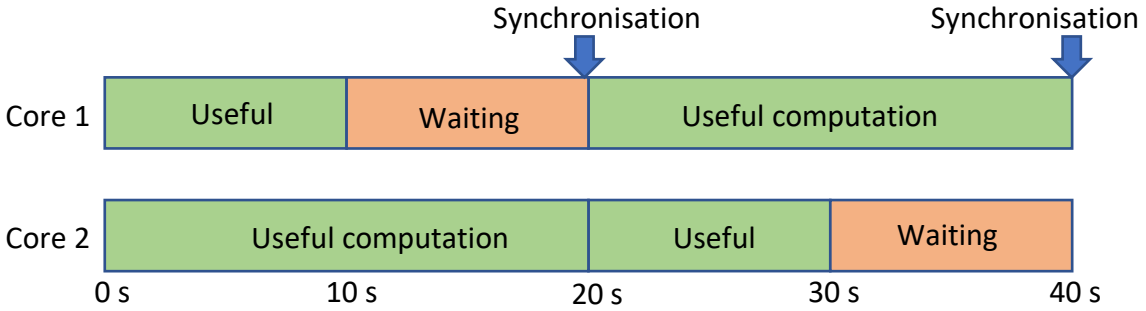

Figure 4

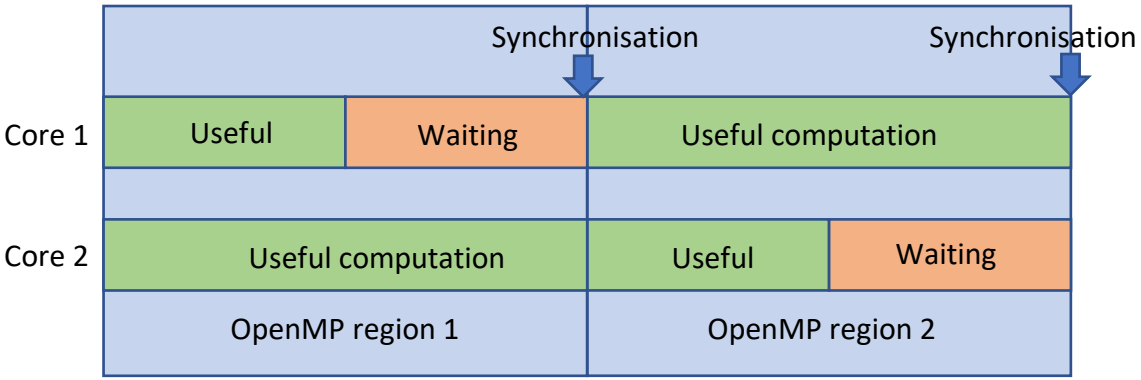

Figure 5

This is the convention chosen for OpenMP parallelisation in these additive metrics, i.e. to describe serialisation and imbalance in terms of load balance within OpenMP regions and serialisation outside OpenMP. This approach is possible because OpenMP provides identifiable and convenient synchronisation points between which to measure serialisation and imbalance.
For MPI the situation is quite different, as it is possible to write MPI code with no global synchronisation other than the start and end of the execution, and even if global synchronisation points do exist, they may not be at convenient locations between which to measure imbalance. Hence with MPI, it is much more natural to think of load balance in terms of distribution of total useful work between the start and end of the region of interest, and to think of serialisation as the additional inefficiency from time waiting in MPI due to MPI dependencies combined with the local imbalances which average out over the full region of interest. This is the approach adopted here, as well as that used in the original POP MPI metrics. In this approach the pattern in Figure 4 is considered as perfectly load balanced, with inefficiency arising from serialisation

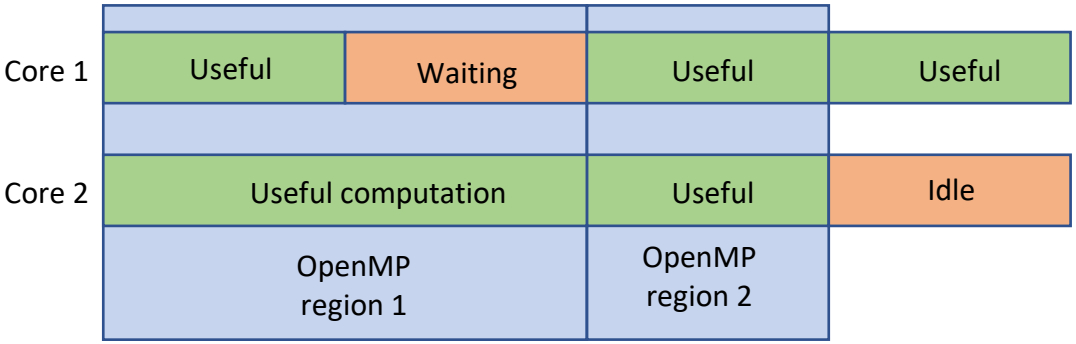

Figure 6

due to the dependencies which arise because of the synchronisation points.

Hence, the Thread Efficiency sub-metrics include an OpenMP Load Balance Efficiency to measure the absolute cost of imbalance within the OpenMP regions, based on the imbalance for each execution of each OpenMP region, and a Serial Region Efficiency to measure the cost of serial computation outside OpenMP regions. The table below illustrates the value of these efficiency metrics for Figures 5 & 6.

	Parallel Efficiency	Serial Region Efficiency	Open MP Load Balance Eff.
Figure 5	0.75	1.0	0.75
Figure 6	0.75	0.875	0.875

Calculating Serial Region Efficiency is relatively straightforward, assuming trace tools can measure the time in useful computation outside OpenMP. Calculating the cost of the imbalance within OpenMP is trickier, as it requires measuring the difference between maximum useful computation and average useful computation for every execution of every OpenMP parallel region. NAG-PyPOP provides a tool with this functionality, which can calculate OpenMP Load Balance Efficiency from Extrae trace data. VTune also can be used to calculate the cost of the imbalance within OpenMP.

The Process Efficiency sub-metrics follow the philosophy used in the classic POP MPI metrics, other than redefining Process Load Balance Efficiency and Process Serialisation Efficiency as additive metrics. With this approach, Process Load Balance Efficiency measures the cost of imbalance in the distribution of total useful work, i.e. the imbalance cost which would be measured if MPI dependencies between processes were removed. Serialisation Efficiency measures inefficiency from process serialisation which arises because of MPI dependencies, i.e. imbalances which average out over the region of interest.
To illustrate this, if Figure 4 represented two processes, the Process Load Balance Efficiency would be 1.0, i.e. perfect load balance, and Serialisation Efficiency would be 0.75.
Both Process Load Balance and Serialisation Efficiency measure inefficiency arising from time within MPI which is unrelated to data transfer, i.e. the MPI time that would remain if data transfer were instantaneous.

Process Load Balance Efficiency is relatively easy to calculate, assuming trace data can be used to measure time per process in OpenMP and in serial computation outside OpenMP.
Process Serialisation Efficiency can be trickier to obtain, as it requires first calculating the proportion of execution time resulting from process MPI data transfer over the network. This is usually possible for Extrae traces using the Dimemas simulation tool, which can calculate the runtime that would be achieved on a theoretical ideal network, i.e. with simultaneous data transfer.