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

Multiplicative performance metrics for hybrid parallel codes

The idea

1. Time distribution as a percentage

The approach is to get insight about the usage of the resources allocated to our parallel execution by computing the percentage of time spent on the key factors that affect the efficiency of a parallel execution.

We can consider that the time spent in the parallel runtimes is the price we have to pay to run in parallel, and the efficiency of the parallelization (Parallel Efficiency) is defined as the percentage of time spent computing (outside the parallel runtime).

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

2. Working with common key factors

Making an abstraction, we can simplify any parallel execution as multiple resources cooperating to achieve a common goal. That mainly requires two things: distribute the work and some kind of coordination/communication. This abstraction provides us the two metrics:

Load Balance Efficiency* = *average(comp)* / *max(comp)

***Communication Efficiency* = max(comp) / runtime**

These metrics measure the impact of the two key factors that impact the efficiency of a parallel execution and can be expressed either as a percentage of time or as a number between 0 and 1, the higher the better.

3. Combining two metric hierarchies

If we express the metrics as a number between 0 and 1, the efficiencies can be represented in a hierarchy, where each metric is equal to the product of their child metrics.

Parallel Efficiency = Load Balance Efficiency* x *Communication Efficiency

In the case of hybrid codes, we need a second hierarchy to identify the contribution of each parallel paradigm.

Parallel Efficiency = MPI Parallel Efficiency x OpenMP Parallel Efficiency

Both hierarchies share the lower level metrics, after splitting the efficiencies by programming model and key factor (for example: MPI Load Balance Efficiency).

Analysing the scalability

Additionally to the Parallelization Efficiency, the performance achieved when increasing the scale is also determined by the scaling of the computations that are not a characteristic of the run itself but its behaviour w.r.t. a reference case (usually the smallest core count).

If we define ***comp_ref*** as useful computation for the reference case, and taking into account whether the parallelization uses strong or weak scaling, we can define the Computation Scaling metric as:

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

Computation Scaling (weak) = average(comp_ref) / average(comp)

The scaling of the computations can be refined as three components: instructions, IPC and frequency. This splits the efficiency into the three main axes that determine the duration of the computation: amount of work (instructions), working speed (IPC) and speed of the resource (frequency):

Computation Scaling = Instruction Scaling x IPC Scaling x Frequency Scaling

Top level efficiency metrics

Combining the efficiency of the parallelization and the scaling of the computation we obtain the **Global Efficiency**.

Global Efficiency = Parallel Efficiency x Comp. Scaling

There is an equivalence between the global efficiency and the speed-up. The main difference is that the global efficiency also considers the inefficiencies of the reference case (Parallel Efficiency) that is not registered in the speed-up computation.

The benefits of an abstract model based on key factors

Using the same concepts to characterize the efficiency for any programming model or combination of multiple programming models allows the user to directly compare different implementations and to identify the strengths and weak points of each of them based on the values for the different key factors.

Working at this abstract level also distances us from any known performance problem and allows us to objectively identify the best path for code optimization.

Calculating the metrics

Most of the metrics can be calculated using values easily extracted from trace data (and most of them even with a profile), i.e.

- Time in useful computation (***comp***)
- Time outside MPI runtime (***out_MPI***)

BSC Basic Analysis automates metrics calculation from Extrae

Isolating MPI contribution

The first step in identifying the loss of performance due to each programming paradigm is to isolate the MPI contribution. The objective is to classify any waiting time that has been caused by MPI, even when that time is in OpenMP.

In hierarchical codes, where the MPI is done only by the master thread while other threads are either idling or inside a barrier (when the MPI is inside a parallel region), the formula for the MPI parallel efficiency is computed considering only the master threads:

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

OpenMP contribution

As we have both the efficiency at hybrid level as well as the efficiency at the MPI level, we can consider that OpenMP is responsible for the part of the hybrid level that cannot be explained by MPI. For example:

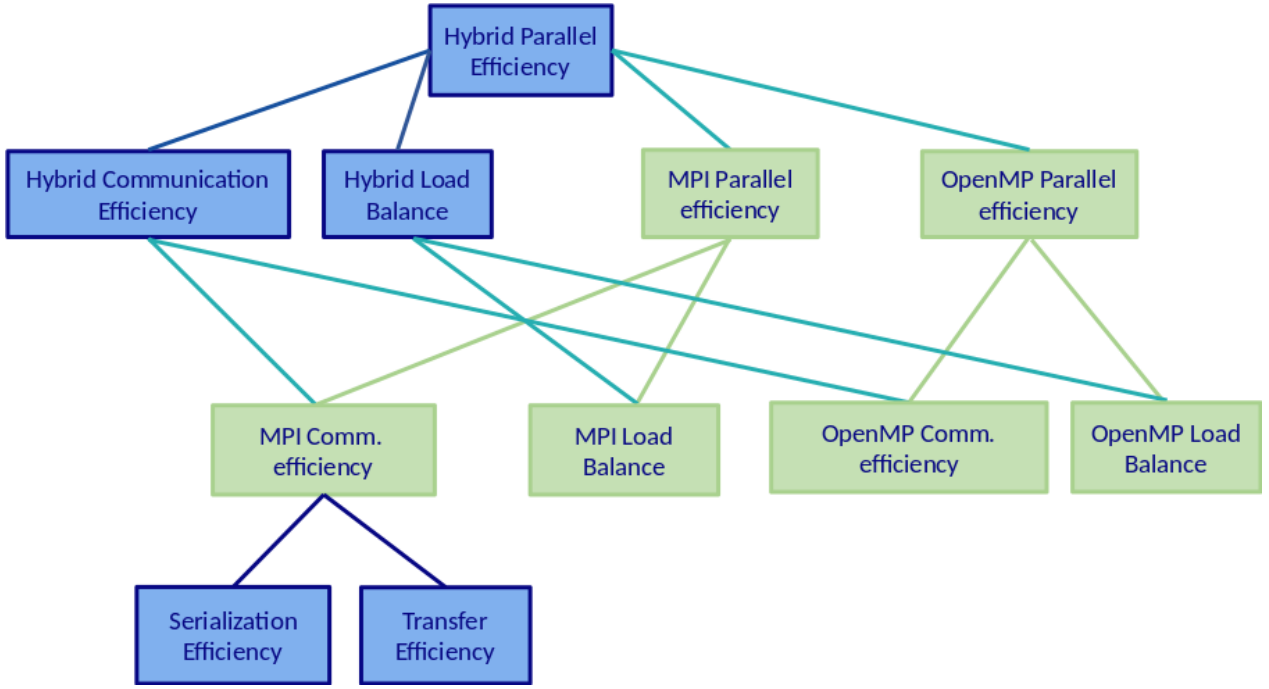
OpenMP Parallel Eff. = Hybrid Parallel Eff. / MPI Parallel Eff.

As a result, the lower level of the OpenMP efficiencies may be greater than 1, reflecting that OpenMP is able to improve a specific key factor.

Hybrid MPI+CUDA

The same approach introduced for MPI+OpenMP can also be applied to analyse the efficiency of MPI+CUDA.

In MPI+CUDA codes, it may be interesting to compute the metrics both for all resources (CPUs and GPUs) and for GPUs only, as in many cases the most efficient solution is not to compute on the CPUs.



Using the Basic Analysis tool

The latest release of the BSC basic Analysis tool automatically checks the programming model(s) and the level of detail in the trace to compute the metrics based on the data. If it detects it is a trace from a hybrid application, the full tree with the decomposition between programming models is computed. The tool also includes sanity checks to detect whether there is a significant percentage of time in either I/O or flushing the trace buffers, as they will affect to the quality of the computed metrics.

The current version of the basic analysis tool supports the scenario where MPI is called within an OpenMP construct. The restriction is that MPI must only be called from the master threads, while the other threads are idling (when the MPI is called outside the parallel regions) or waiting in a OpenMP barrier (when MPI is called from an OpenMP parallel region).

Currently under development, we are refactoring the metrics to support executions where other threads are computing during the MPI calls and where multiple threads call MPI.

Serialization and Transfer efficiencies

The Communication Efficiency can be decomposed into Serialization and Transfer efficiencies using the BSC Dimemas Simulator but it can only be applied at the MPI level (either pure MPI or Hybrid).

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