2SC7695 – Low cost optimization of acoustic wave propagation code performance

Instructors: Stephane Vialle
Department: DOMINANTE - MATHÉMATIQUES, DATA SCIENCES, DOMINANTE - INFORMATIQUE ET NUMÉRIQUE
Language of instruction: ANGLAIS
Campus: CAMPUS DE PARIS - SACLAY
Workload (HEE): 80
On-site hours (HPE): 48,00

Description

Project topic in partnership with INTEL.

Regardless of the type of application running on parallel machines in an HPC (high performance computing) environment, and regardless of their level of efficiency (in terms of performance or energy footprint), we can easily see that the impact of input parameters is generally not negligible. In particular, we can act on :

- the parameters of the parallel algorithm used, such as the size and shape of the domains performing a sub-problem partitioning,
- HPC implementation parameters controlling for example *cache blocking* to reduce data access times,
- parallel application deployment parameters, controlling the process-MPI vs. threads-OpenMP distribution and the placement of these computational tasks to efficiently occupy CPU cores,
- parameters controlling the thread scheduling policy (usually controlled through environment variables).

These different input parameters allow to improve the execution of HPC code on a computing platform. But it remains extremely difficult to understand exactly how the application behaves in the processor architecture and even if we could make the application more efficient by modifying the source code, the dependency on test cases and the execution environment would still be preponderant. In this context where the parameter space can be of important dimension, the use of optimization algorithms appears fundamental to converge towards a global (or at least local) minimum of a cost function expressing the execution time and the energy footprint. We will therefore use optimization methods and algorithms to optimize the operation of an HPC calculation code.

However, each execution of a test case of an HPC application can be long, even on a parallel machine. The parameter space is large and an optimization algorithm can require tens of thousands of executions of the application (or even more). What is achievable on the scale of a computing kernel remains unbearable on the scale of a complete application.

It is therefore necessary to choose optimization methods that do not consume too many experiments, and then to optimize their use to reduce the footprint of the targeted HPC code, without this pre-study consuming too many computing resources! This amounts to **"looking for the least expensive optimization method"** to find an optimum between convergence speed (of the optimization algorithm) and the quality of the minimization of execution time and energy footprint (of the targeted HPC application).

Technical details of the system :

We will start this project from :

- a high performance computing code (HPC code) that simulates the 3D acoustic wave equation in a homogeneous isotropic finite-difference medium, and runs on multi-core PC clusters (in MPI + OpenMP),
- a genetic algorithm capable of iterating on many parameters of the HPC code such as: size and shape of domains and *cache blocking*, compilation flags, environment variables, number and placement of threads..., but which remains sequential and limited to a single machine.

The genetic algorithm thus calls successively the acoustic wave simulation code in different configurations, and can only call it in a multithreaded version on one machine (not in a distributed version on a cluster). This allows nevertheless to search for the optimal configuration on each PC, and then to apply it on each PC when running on a cluster of PCs.

The *finite differences* being quite explicit in terms of the number of floating operations, we can easily count the number of points processed or the number of floating operations performed during an execution, and deduce a processing speed in Giga points/s or GFlops/s. We can then try to minimize the simulation time (or maximize the processing speed). We can also dedicate and collect some hardware counters to obtain the exact power consumption of the processor and memory, to study in detail the energy impact of our optimizations.

But a genetic algorithm is a population-based optimization method that is computationally resource-intensive, and the implementation provided uses only one PC. The search for an optimal configuration of the code can therefore be very long and prohibitive. To overcome this limitation we will adopt two approaches: