Analytical Performance Modelling is a method of software performance testing generally used to evaluate design options and system sizing based on actual or anticipated system behaviour. It is used in many contexts and, for the specific case of scientific software development, it may be adopted
• for hardware/software co-design in order to make predictions on the efficacy of a specific hardware solution, to monitor hot-spots and bottlenecks as the hardware is designed and to avoid longer and more expensive performance testing,
• for code development, since it helps understanding where there is room for the improvement of parallel performances and to monitor hot-spots and bottlenecks as the hardware evolves,
• to provide indications to the users concerning estimated job durations and/or for the auto tuning of the parallel execution parameters.

Quantum ESPRESSO [1, 2] is one of the most used codes based on plane-wave DFT in the community of material science. It has been successfully ported and optimized on a large number of HPC infrastructures all over the world. The parallel structure of Quantum ESPRESSO is mainly based on several layers of MPI communicators, plus a finer grain OpenMP parallelization [3, 4].
For what concerns the algorithmic and the software-engineering side, a continuous effort has been devoted to the improvement and the extension of the parallelization options, the interoperability possibilities with other codes and the increase of code modularity.

A detailed profiling performed using medium to large sized input data revealed that most of the execution time is spent in MPI, Linear Algebra and FFT calls. Data I/O has negligible impact at the scale considered here.

The following relevant kernels were identified:
• Distributed FFT: 1D FFT calls + MPI Alltoall + memory access.
• Matrix Multiplication kernel: DGEMM and ZGEMM.
• Generalized eigenvalue problem: serial LAPACK functions zhegv, zhegvx.

Load unbalance originating from k-points distribution is also considered.

The number of FLOPS and memory access events for each of these kernels was described as a function of a selected number of input parameters. The final model reads:

\[ T_{\text{kernel}}(f, BW, NB; \text{input}) = \sum T_{\text{kernel}}(f, BW, NB; \text{input}) + T_{\text{other}} \]

where \( f, BW, NB \) are the CPU frequency, the memory bandwidth and the network bandwidth and the number of elements in the sum depends on the input parameters. For each kernel, a number of micro-benchmarks provide a simple interface to acquire the effective underlying hardware characteristics.

The effectiveness of the model was tested on two representative inputs: a bulk system (MnSi) and a 2D system in vacuum (Graphene + Fe).

The problem size is parsed from the executable input files and used to specify the number of operations to be performed in each kernel.

In the panels on the right the comparison between the measured and the estimated execution time of a single self consistent loop cycle is compared.

Results show a very good agreement between the estimated and real execution times and, as a consequence, it also predicts correctly the most effective system for a given input data and parallel execution scheme.

A number of conclusions can be drawn:
• No rocket science! Select relevant kernels and find meaningful variables to evaluate the performances.
• It’s just \( E \), \( + \) and /, the tricky task is the call tree.
• Takes little time! For pw.x, the preliminary work presented here was obtained in 1 week of profiling and two weeks of development/test.
• Already presented and used in co-design calls. Auto-tuning of parallel execution parameters is planned.

References