Performance tuning of applications in HPC environments employing Simulated Annealing

Valentin Plugaru, Sébastien Varrette, and Pascal Bouvry

Computer Science and Communications (CSC) Research Unit,
6 rue Richard Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg
`Valentin.Plugaru@gmail.com` `Sebastien.Varrette@uni.lu` `Pascal.Bouvry@uni.lu`

1 Introduction

Obtaining the best possible speed and efficiency is the main focus in High Performance Computing (HPC) environments where the current challenge is reaching the exaflop barrier as well as offering the computational capacity of Exascale systems, at a reduced power cost. Optimization approaches focus on developing increasingly efficient architectures and power management systems while faster executions are obtained by increases in hardware clock rates, parallelism and optimized algorithms working at both hardware and software level. Applications are generally optimized by profiling, with performance tuning applied at computationally expensive spots.

Building fast software in an HPC environment raises great challenges as the software used for simulation and modelling is generally complex and has many dependencies. Current approaches involve manual tuning of compilation parameters in order to minimize the run time, based on a set of predefined defaults, however this requires expert knowledge, is not scalable and can be very expensive in person-hours. Another approach, explored in the present work, is optimization by traversing the search space given by the compiler-level heuristics, which can be activated through specific flags, and the library compile-time options that perform internal tuning and architecture-specific optimizations in order to achieve better application performance. A well-known compile-time tuning system that produces platform-optimized library builds is ATLAS: Automatically Tuned Linear Algebra Software, a code generation system targeting optimal BLAS and LAPACK routines, while metaheuristics-based compiler flag selection have been explored by Hoste and Eeckhout [1] and Zhong et al. [2].

The present work builds on the existing concepts, proposing and developing a modular and generic framework called poHPC that uses a Simulated Annealing (SA) metaheuristic algorithm to automatically search for the optimal set of both library options and compilation flags that can give the best execution time for a library-application pair on a selected hardware architecture without changing the software internals. The framework can be used in modern HPC clusters using a variety of batch scheduling systems as execution backends for the optimization runs, and will discover optimal combinations as well as invalid sets of options and flags that result in failed builds or application crashes through executions with real world test cases. We demonstrate the optimization of the FFTW library working in conjunction with the high-profile community codes GROMACS and QuantumESPRESSO (QE), whereby the suitability of the technique is validated.

2 The poHPC framework

The poHPC framework, presented in Figure 1 is composed of four modules: the poHPC core, the Optimization engine, Fitness evaluator and Job management.

The poHPC core is the main user application, utilizing specific classes defined in the other modules based on user's selection of application/library pair to be optimized, and configuration parameters specified in several manifest files. The libraries and compilers manifest files contain lists of compilation parameters particular to each configured library (*e.g.* for FFTW: `--enable-sse2, --enable-avx, --enable-float, --enable-long-double`), respectively compiler-specific optimization flags (*e.g.* for GCC: `-O0 ...-O3, -Ofast, -funroll-loops, -fvect-cost-model, -ftracer, -mssse3, -msse4.2, -mavx`).

The Optimization engine is the module meant to implement different optimization algorithms in order to allow maximum flexibility in choosing an appropiate one based on the user's needs, considering the accuracy/performance tradeoffs inherent in algorithms and the available resources.

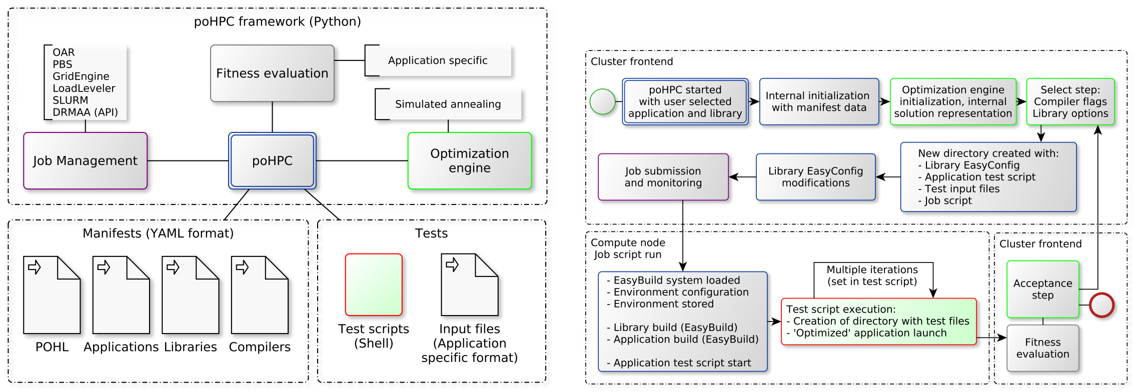
poHPC framework (Python)
OAR
PBS
GridEngine
LoadLeveler
SLURM
DRMAA (API)
Fitness evaluation
Application specific
Simulated annealing
Job Management
poHPC
Optimization engine
Manifests (YAML format)
POHL
Applications
Libraries
Compilers
Tests
Test scripts (Shell)
Input files (Application specific format)
Cluster frontend
poHPC started with user selected application and library
Internal initialization with manifest data
Optimization engine initialization, internal solution representation
Select step: Compiler flags Library options
New directory created with:
- Library EasyConfig
- Application test script
- Test input files
- Job script
Library EasyConfig modifications
Job submission and monitoring
Compute node
Job script run
- EasyBuild system loaded
- Environment configuration
- Environment stored
- Library build (EasyBuild)
- Application build (EasyBuild)
- Application test script start
Multiple iterations (set in test script)
Test script execution:
- Creation of directory with test files
- 'Optimized' application launch
Cluster frontend
Acceptance step
Fitness evaluation

Fig. 1: The poHPC framework: architecture (left) and workflow in a HPC environment (right).

Several metaheuristics algorithms can be used to perform combinatorial optimization, such as Tabu Search, Simulated Annealing or population-based Evolutionary Algorithms. In the present work a SA algorithm has been implemented as a class of the optimization engine. For poHPC the SA algorithm has been chosen instead of a simple Random Search as better solutions may be found in the immediate neighbourhood of a known good solution. As an example the *-O3* compiler flag should result in better performing code than when using the *-O2* flag, with the solution containing the former being in close proximity to the latter, all other parameters being the same. However, the choice of *-O3* may have the consequence of producing an inviable application due to aggresive optimization by the compiler thus the search space cannot be reduced to configurations containing only *-O3*. Also, compared to population-based optimization approaches, SA needs to evaluate only one solution's cost per iteration which makes it much more efficient, as computing the cost is very expensive due to the long time (at least on the order of minutes) required in order to compile the library, application and run representative test cases with the application.

The Fitness evaluator module contains application-specific classes that are able to parse output files, extract and aggregate the required information that ultimately serves as the cost function (fitness) value, to be minimized by the optimization module. For the current work, fitness evaluating classes have been developed for the QuantumESPRESSO and GROMACS applications - well known simulation packages for quantum chemistry, respectively molecular dynamics.

Finally, the Job management module is intended for classes that are specific to the Batch Job Scheduler systems commonly in use in HPC clusters such as Torque/PBS, GridEngine, LoadLeveler and OAR. OAR-specific functionality has been implemented, allowing the submission, monitoring the status and waiting for the termination of a batch job executed under the Batch Scheduler in use at the HPC Platform of University of Luxembourg, utilized in this study.

3 Experimental results

Several sets of experiments have been performed with the poHPC framework for the optimization of QuantumESPRESSO and GROMACS applications on real-world test cases involving FFT-intensive calculations. Sample optimization results for QuantumESPRESSO/FFTW are presented in Table 1, showing the best solution performing 36.7% better (less time spent in fftw routines as measured by QE) than with compilation defaults, when the SA algorithm was executed with a mode of operation that generated close neighbourhoods for the accepted solution.

Application	**Compiler flags**	**FFTW library options**	**Gain over defaults**
QE	-O1 -msse3	–enable-sse2	18.3%
QE	-O1 -funroll-loops -fvect-cost-model -ftracer -mavx	–enable-sse2	36.7%

Table 1: Sample optimized solutions found by poHPC for QuantumESPRESSO.

References

1. Hoste, K., and Eeckhout, L. Cole: compiler optimization level exploration. In *CGO* (2008), M. L. Soffa and E. Duesterwald, Eds., ACM, pp. 165–174.
2. Zhong, S., Shen, Y., and Hao, F. Tuning compiler optimization options via simulated annealing. In *Proceedings of the 2009 Second International Conference on Future Information Technology and Management Engineering* (2009), FITME '09, pp. 305–308.