HPC user has a limited amount of time to adapt an application to a given system and to configure the environment in an optimal way.

Consequently, even if very efficient algorithms are implemented resources are often wasted. The following lists a few reasons why resources might be utilized suboptimally:

- Limited scalability of the algorithm adds computational and communication overhead.
- A wrong mapping of processes to hardware leads to unnecessary communication.
- An inefficient I/O access pattern degrades performance of the parallel file system.
- Load imbalance in I/O or computation causes bottlenecks and idle resources on the system.

Efficiency also depends on the system characteristics, that is the hardware characteristics of the basic components: network, storage and compute nodes; the topologies of all interconnects, software layers involved and finally, on the configuration made by the administrator and user.

In software layers the system's algorithms for communication and I/O have a large influence on the observable performance[14]. Performance of the communication library itself is influenced by a number of factors. First, it is obvious that the library should be geared towards a given system, after which the right communication algorithm must be chosen. However, the tuning is complicated because performance of each communication algorithm depends on the parameters specified by the user. For example, while the broadcasting of small messages could be faster on the given hardware by sending data sequentially from the root, a tree algorithm might perform better for larger data, and a pipelined peer-to-peer alike algorithm would be preferable for big data. While the performance of the algorithm could be optimized for itself, efficiency also depends on the context of the application, that is, the number of processes and the current and future instructions executed by the processes.

Heterogeneous systems like Grids and Clouds increase complexity of the optimization exponentially, as more inhomogeneous components are involved.

It certainly is not easy to understand the interplay between all the hardware characteristics and potential bottlenecks, and unfortunately, mechanisms designed to optimize the system make it even harder to assess achieved performance and relate that performance to the system's capability. Further, on a high level of abstraction the communication interface implementation and I/O layers use techniques that, in most cases, improve performance. This is achieved by manipulating the request, deferring the operation, or fusing operations into one compound operation.

The complexity of analyzing an application on a distributed supercomputer becomes clear when a developer tries is to understand performance of a particular sequential code, which is only executed on a single processor. On the one hand, observed performance depends on the CPU architecture – branch-prediction, caches, the translation lookaside buffer, to name just a few mechanisms for optimizing hardware. On the other hand, it also depends on the compiler, which tries to transform the given high-level code to machine instructions in the best way possible.

From the user's perspective, post-mortem performance optimization is state-of-the-art. In this process performance of an application is measured on an existing system and analyzed to identify bottlenecks. Nowadays, developers are happy to achieve 10% peak performance on a given system[15]. It is important to optimize from the most promising and performance-boosting bottleneck to the least one. When confronted with this issue an important question arises concerning how much work is necessary to tune or modify the algorithm, the code and the system, and what will be gained by these modifications.

Estimating performance Modeling the system allows assessing obtained performance and therewith estimate the performance potentially gained by optimizations. Simple approaches to optimize floating point operations are to measure them and compare them with the theoretical peak performance of the system.

[14]More information of relevant performance factors is given in Section 2.2.

[15]This information is derived from available presentations of compute centers and scientific applications.