Chapter 3
Dense Linear Algebra:
Applying New Features to Traditional Paradigms

ABSTRACT

This chapter introduces innovative approaches for the efficient use of some of the most novel techniques based on tasking to optimize dense linear algebra operations. The idea is to explore high-level programming techniques that increment the programming productivity and performance for dense linear algebra operations. The authors apply these techniques on some of the most important and widely used dense linear algebra kernels, such as the GEMM and TRSM routines of the BLAS-3 standard, as well as the LU factorization and solve of the LAPACK library. The authors use as target platforms two different current HPC architectures: a CPU multi-core processor and a GPU hardware accelerator. Different approaches are presented depending on the target platform, but always based on tasking.

INTRODUCTION TO DENSE LINEAR ALGEBRA

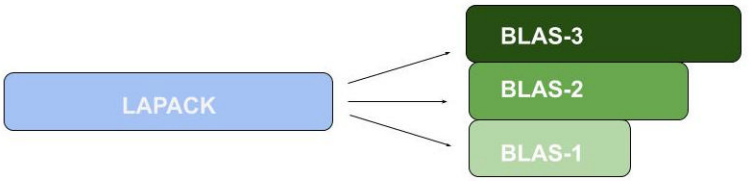
Dense Linear Algebra (DLA) operations are at the bottom of numerous engineering and scientific applications, which rely on DLA routines to make their calculations. The importance of DLA is observed in a wide variety of fields, such as macromolecular simulations, forecasting or medical care, among others. A straightforward consequence of this situation is the existence and

DOI: 10.4018/978-1-7998-7082-1.ch003

continuous improvement of several DLA libraries. Both vendor and open-source solutions exist nowadays, e.g., Intel MKL (Intel Software, 2012), IBM ESSL (IBM, 2012), PLASMA (PLASMA Project, 2020), libFLAME (Gunnels & van de Geijn, 2001), ScaLAPACK (Castaldo & Whaley, 2010), LASs (Valero et al., 2019), sLASs (Valero et al., 2020), just to mention a few.

All the mentioned libraries provide either BLAS (Netlib.org, 2017), LAPACK (Netlib.org, 2019) or both functionalities via the use of different programming models and parallelization strategies. The common target in all cases is the increase of performance, although each library focuses on a different aspect. On the other hand, LAPACK routines are usually implemented relying on BLAS level routines, making their optimization key in order to improve the performance at both levels: BLAS and LAPACK (Figure 1).

Figure 1. BLAS and LAPACK interaction



At the same time, BLAS routines are subdivided in three more levels, which rely one on the others. BLAS levels are referred as BLAS-1, BLAS-2 and BLAS-3, depending on the type of input data. In the early 1980s, when BLAS development started, the most powerful computers featured vector processors. For this reason, the original BLAS (nowadays BLAS-1) was designed to work on vectors. A few years later, in 1987, BLAS-2 appeared as an extension of the previous routines to perform matrix-vector operations. However, with the creation of new architectures that integrated multiple levels of cache memory, in order to mitigate the difference in throughput between the processor and memory, the existing BLAS levels were not able to attain reasonable performance. The ratio between the number of operations and data movement is O(1) in both cases, but this proportion is much higher between the processor frequency and the memory bandwidth. As a response to the new scenario, BLAS-3 was created in 1989 to palliate the effect of memory data transfers. By the use of blocked algorithms that increased the