Matrices Over Runtime Systems @ Exascale

[Extended Abstract for related Poster]

Emmanuel Agullo, Olivier Aumage, George Bosilca, Bérenger Bramas, Alfredo Buttari, Olivier Coulaud, Éric Darve, Jack Dongarra, Mathieu Faverge, Nathalie Furmento, Luc Giraud, Abdou Guermouche, Julien Langou, Florent Lopez, Hatem Ltaief, Samuel Pitoiset, Florent Pruvost, Marc Sergent, Samuel Thibault and Stanimire Tomov

ABSTRACT

The goal of the Matrices Over Runtime Systems @ Exascale (MORSE) project is to design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale multicore systems with GPU accelerators, using all the processing power that future high end systems can make available. We propose a framework for describing matrix algorithms with a sequential expression at a high level of abstraction and delegating the actual execution to a runtime system. In this poster we show that this model allows for (1) achieving an excellent scalability on heterogeneous clusters, (2) designing advanced numerical algorithms and (3) being compliant with standards such as OpenMP 4.0 or possible extensions of this standard. We illustrate our methodology on three classes of problems: dense linear algebra, sparse direct methods and fast multipole methods. The resulting codes have been incorporated into the Chameleon, QR_MUMPS and ScalFMM solvers, respectively.

1. SCALABILITY: ILLUSTRATION WITH CHAMELEON DENSE LINEAR ALGEBRA LIBRARY

Chameleon is a dense linear algebra software similar to LAPACK but for heterogeneous/hybrid architectures. We present here how Chameleon has been extended using the StarPU / Quark runtime systems in order to handle multicore nodes enhanced with multiple GPUs and clusters of interconnected nodes. Chameleon includes one-sided factorizations (LLt, LU, QR) and relative solvers as well as main level-3 BLAS operations.

Tile algorithms. Similarly to PLASMA, the matrix is split in square blocks called tiles (see Figure 1). Based on such a layout, tile algorithms aim at improving data locality and increase parallelism. The operations performed on those tiles are executed using optimized CPU and GPU kernels. When GPU are turned on, the tile size is increased to fully benefit from the potential of the accelerators.

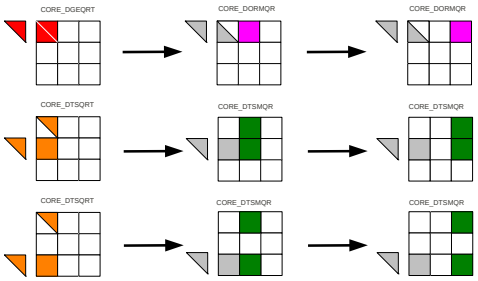

Figure 1: Tile QR algorithm.

DAG. The high-level algorithm (here a QR factorization) can then be represented as a Directed Acyclic Graph (DAG) of tasks where the vertices represent the tasks in which it is decomposed and the edges represent the dependencies among them. Figure 2 shows a 4x4 tile QR DAG. The runtime system (here StarPU) dynamically schedules the tasks on the available hardware (CPU or GPU) and ensures the data coherency.

Parallel distributed extension. We have extended the mechanism to the distributed memory. In our approach, all nodes unroll the whole task graph. They determine tasks they will execute and can infer required communications. As a consequence, no negotiation between nodes is needed, contrary to master-slave approaches. Figure 3 shows for instance how Node 0 and Node 1 unroll the task graph concurrently.

Performance. Figures 4 and 5 show the obtained performance on an homogeneous cluster (CPU only) and an heterogeneous cluster (enhanced with GPUs), respectively. The results show that the approach is competitive with respect to other state-of-the-art libraries such as Scalapack (homogeneous case) and DPlasma (both in the homogeneous and heterogeneous cases).

2. DESIGNING ADVANCED NUMERICAL ALGORITHMS: ILLUSTRATION WITH THE QR_MUMPS SPARSE DIRECT SOLVER

The qr_mumps QR solver is a software package for the solution of sparse, linear systems on multicore computers. It implements a direct solution method based on the QR fac-

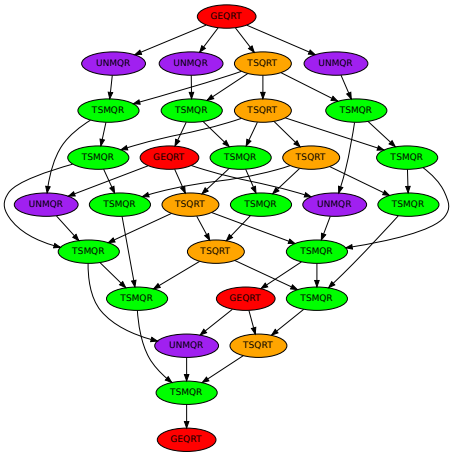

Figure 2: 4x4 Tile QR DAG.

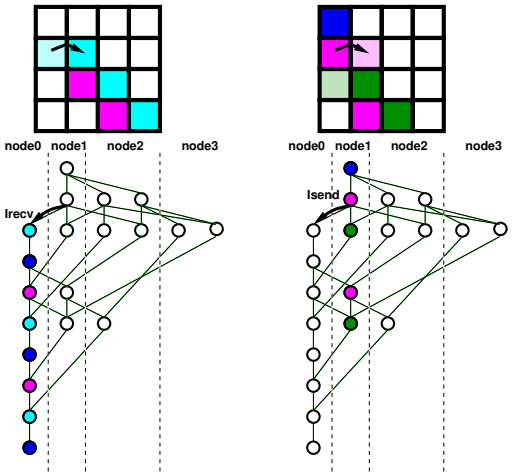

Figure 3: Node 0 execution & Node 1 execution.

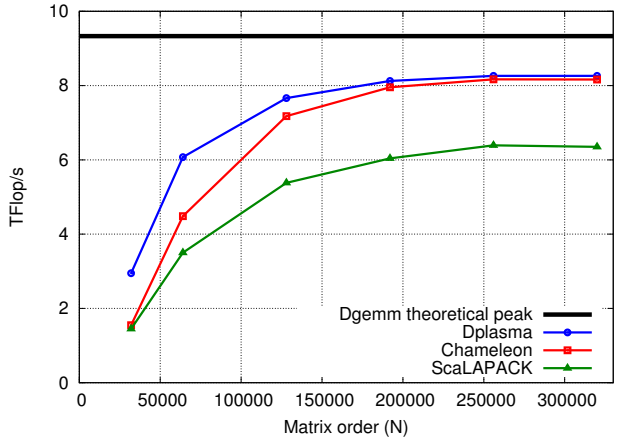

Figure 4: Performance on a 144 nodes cluster where each node is composed of two quad-core Intel Xeon X5620 (8 CPU cores)

.

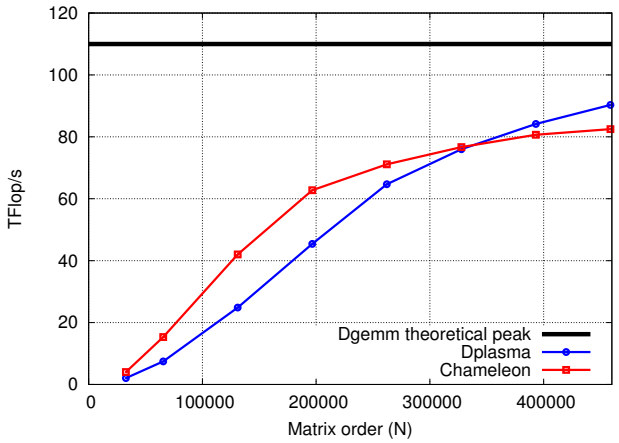

Figure 5: Performance on a 144 nodes cluster where each node is composed of two quad-core Intel Xeon X5620 (8 CPU cores) and two Nvidia Tesla M2090 (2 GPUs).

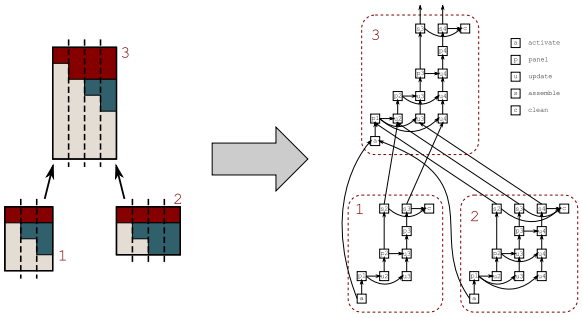

Figure 6: Fine-grain parallelism.

torization of the input matrix. It has been extended with the support of the StarPU runtime system for handling multicore platforms enhanced with a GPU.

Multifrontal Tile Algorithm. Figure 6 shows how parallelism is achieved through a decomposition of the workload into fine-grained computational tasks which basically correspond to the execution of a BLAS or LAPACK operation on a block-column. Parallelism is achieved by dividing the workload into fine grained tasks that are arranged in a Direct Acyclic Graph (DAG). The model allows for pipelining tasks at different levels in the tree, which provides high efficiency and scalability.

Granularity. Figure 7 shows how the data decomposition is performed with different grains according to the target architecture. In the multicore case, a fine grain decomposition is applied. Picture (a) on the left shows for instance a 1D fine grain decomposition. To better benefit from accelerators, a coarser grain may be used (b) such as in the case of the dense linear algebra MAGMA reference library. Finally, to better cope with heterogeneous platforms, a hierarchical decomposition is employed (c).

Performance. Figures 8 and 9 show the obtained performance on an homogeneous cluster (CPU only) and an het-

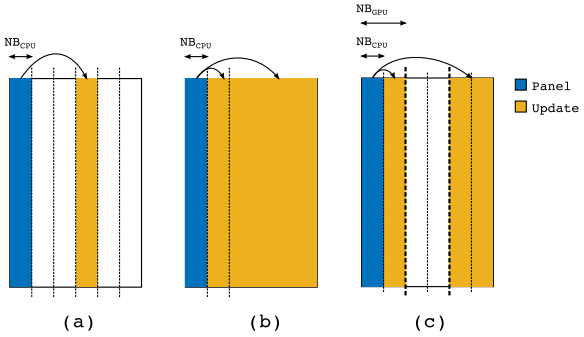

Figure 7: Granularity.

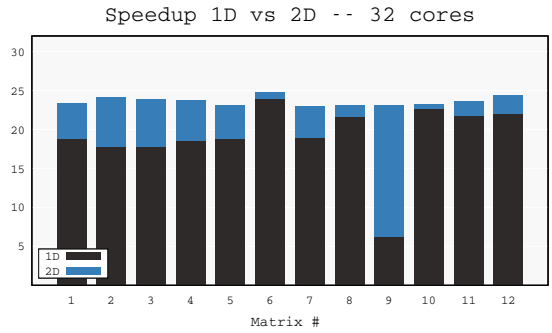

Figure 8: Performance on four octo-core Intel Sandy Bridge E5-4650 (32 CPU cores).

erogeneous cluster (enhanced with GPUs), respectively. The results show that relying on 2D tile algorithms allows for very high performance on a large range of matrices (homogeneous case) and that hierarchical partitioning allows for a better performance in the heterogeneous case.

3. TOWARDS OPENMP TASK-BASED PROGRAMMING? DISCUSSION ILLUSTRATED WITH THE SCALFMM CODE

The Parallel Fast Multipole Library for Large Scale Simulations (ScalFMM) library aims at simulating N-body inter-

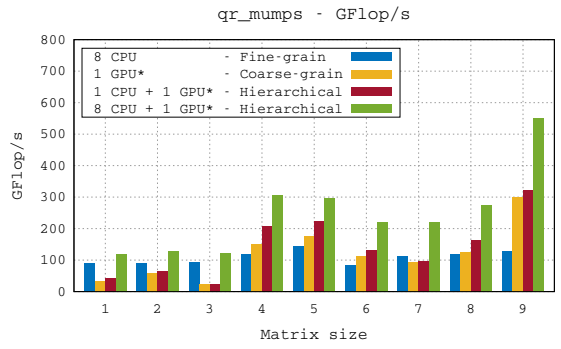

Figure 9: Performance on one octo-core Intel Xeon E5-2650 (8 CPU cores) enhanced with one Nvidia Kepler K40c GPU.

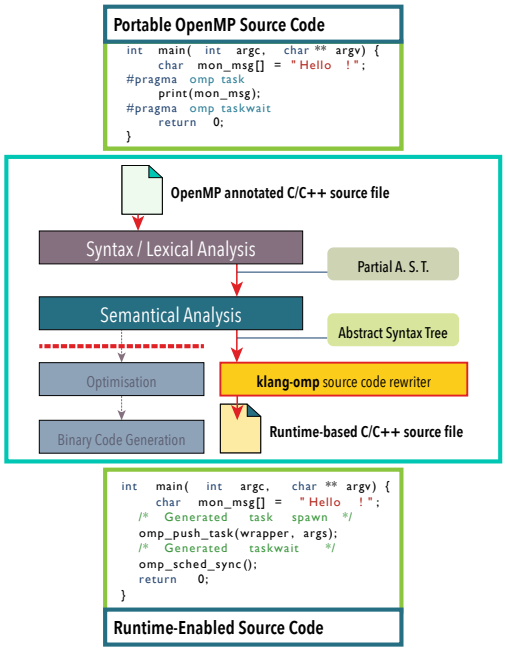

Figure 10: Klang-OMP design.

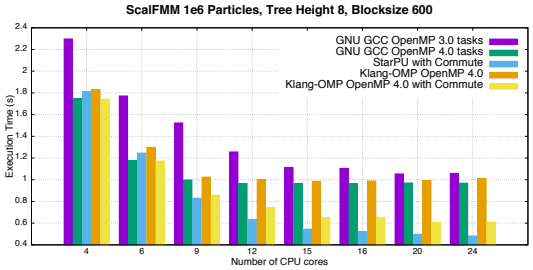

Figure 11: ScalFMM performance with according to different programming paradigms.

actions or matrix-vector products using the Fast Multipole Method (FMM). This software intends to offer all the functionalities needed to perform large parallel simulations while enabling an easy customization of the simulation components: kernels, particles and cells. The current MORSE distribution of ScalFMM proposes FMM based on interpolation.

Klang-OMP OpenMP Compiler. StarPU can be programmed natively through its dedicated API, or through the newly introduced Klang-OMP source to source OpenMP compiler. Klang-OMP is able to generate StarPU enabled code from an OpenMP 3.1 or 4.0 compliant task-based application (see Figure 10). Figure 11 compares the performance of the ScalFMM library when relying on the native low-level StarPU task-based API, to the version written in OpenMP compiled with GNU GCC as well as with Klang-OMP.