Dynamically scheduled Cholesky factorization on multicore architectures with GPU accelerators.

Emmanuel Agullo*, Cédric Augonnet*, Jack Dongarra†‡§, Hatem Ltaief†,
Raymond Namyst*, Jean Roman*, Samuel Thibault*, Stanimire Tomov†

*INRIA – LaBRI – University of Bordeaux
†Department of Electrical Engineering and Computer Science – University of Tennessee, Knoxville
‡Computer Science and Mathematics Division – Oak Ridge National Laboratory
§School of Mathematics & School of Computer Science – University of Manchester

{emmanuel.agullo, cedric.augonnet, raymond.namyst, jean.roman, samuel.thibault}@inria.fr
{dongarra, ltaief, tomov}@eecs.utk.edu

***Abstract*—Although the hardware has dramatically changed in the last few years, nodes of multicore chips augmented by Graphics Processing Units (GPUs) seem to be a trend of major importance. Previous approaches for scheduling dense linear operations on such a complex node led to high performance but at the double cost of not using the potential of all the cores and producing a static and non generic code. In this extended abstract, we present a new approach for scheduling dense linear algebra operations on multicore architectures with GPU accelerators using a dynamic scheduler capable of using the full potential of the node [1]. We underline the benefits both in terms of programmability and performance. We illustrate our approach with a Cholesky factorization relying on cutting edge GPU and CPU kernels [2], [3] achieving roughly 900 Gflop/s on an eight cores node accelerated with three NVIDIA Tesla GPUs.**

I. Cholesky Factorization

The Cholesky factorization (or Cholesky decomposition) is mainly used for the numerical solution of linear equations $Ax = b$, where A is symmetric and positive definite. Such systems arise often in physics applications, where A is positive definite due to the nature of the modeled physical phenomenon. This happens frequently in numerical solutions of partial differential equations. The Cholesky factorization of an $n \times n$ real symmetric positive definite matrix A has the form

$$A = LL^T,$$

where L is an $n \times n$ real lower triangular matrix with positive diagonal elements. In LAPACK the double precision algorithm is implemented by the DPOTRF routine. A single step of the algorithm is implemented by a sequence of calls to the LAPACK and BLAS routines: DSYRK, DPOTF2, DGEMM, DTRSM. The tile Cholesky algorithm is identical to the block Cholesky algorithm implemented in LAPACK, except for processing the matrix by tiles. Otherwise, the exact same operations are applied. The algorithm relies on four basic operations implemented by four computational kernels: DPOTRF which performs the Cholesky factorization of a diagonal tile, DTRSM which performs a triangular solve, DSYRK which executes a symmetric rank-k update and DGEMM which applies a matrix multiplication. Figure 1 shows the pseudocode of the Cholesky factorization.

```
FOR k = 0..TILES-1
  A[k][k] ← DPOTRF(A[k][k])
  FOR m = k+1..TILES-1
    A[m][k] ← DTRSM(A[k][k], A[m][k])
  FOR n = k+1..TILES-1
    A[n][n] ← DSYRK(A[n][k], A[n][n])
    FOR m = n+1..TILES-1
      A[m][n] ← DGEMM(A[m][k], A[n][k], A[m][n])
```

Fig. 1. Pseudocode of the tile Cholesky factorization.

II. The StarPU Framework

StarPU is a runtime system that schedules tasks onto accelerator-based platforms. It is meant to be used as a back-end for *e.g.* parallel language compilation environments and High-Performance libraries. The two basic principles of StarPU is firstly that tasks can have several implementations, for some or each of the various heterogeneous processing units available in the machine, and secondly that transfers of data pieces to these processing units are handled transparently by StarPU.

Thanks to auto-tuning facilities, StarPU transparently predicts execution time and data transfer overhead. This permits StarPU's dynamic scheduler to avoid load imbalance while enforcing data locality to reduce the pressure on the memory subsystem, which is a critical resource for accelerator-based platforms.

What is required to port an application on top of StarPU?

- Register the different data to StarPU (in our case, we register the tiles).
- Create wrappers for the different kernels implementations (eg. for SGEMM).

- Describe the algorithm as a set of tasks. A StarPU task is defined by a codelet, the list of data that should be accessed by the task as well as their access modes.
- Task dependencies are either given explicitly or automatically derived from data dependencies if the different tasks are submitted in an order that corresponds to a valid sequential execution.

Note that we do not need all tasks to perform the algorithm, so that it is possible that the application dynamically adapts itself provided performance feedback from StarPU. We can also cap the amount of resources (e.g., memory or number of processing units) required to perform the computations

III. Preliminary Results

We briefly present preliminary results obtained for the Cholesky factorization in single precision. The GPU kernels were taken from the MAGMA library [2], and we used kernels from the PLASMA library on the multicore CPUs [3]. The platform used for this experiment is composed of two quad-core Intel Nehalem X5550 CPUs (8 CPU cores total) running at 2.67 GHz with 48 GB of memory divided in two Non Uniform Memory Access (NUMA) nodes. It is enhanced with three NVIDIA Quadro FX5800 GPUs of 240 cores each (720 GPU cores total) running at 1.3 GHz with 4 GB of GDDR3 per GPU. Using the three GPUs (associated to three CPU cores), the Cholesky factorization achieved 780 Gflop/s (Figure 2) corresponding to a perfect speedup equal to 3 (Figure 3). The use of five supplementary cores allows to increase the performance up to 900 Gflop/s. These supplementary 120 Gflop/s are above the *SGEMM* peak of the corresponding cores (100 Gflop/s). Although this result is non intuitive, it corresponds to a natural property of parallel computing. Indeed, the *SPOTRF* operation (level-2 BLAS) that is applied on diagonal blocks achieves a very low percentage of the theoretical peak of a GPU. By mainly performing this operation on CPUs (80% of the SPOTRF operations have been performed on CPUs), GPUs can be dedicated to operations for which they are very efficient such as *SGEMM* (level-3 BLAS).

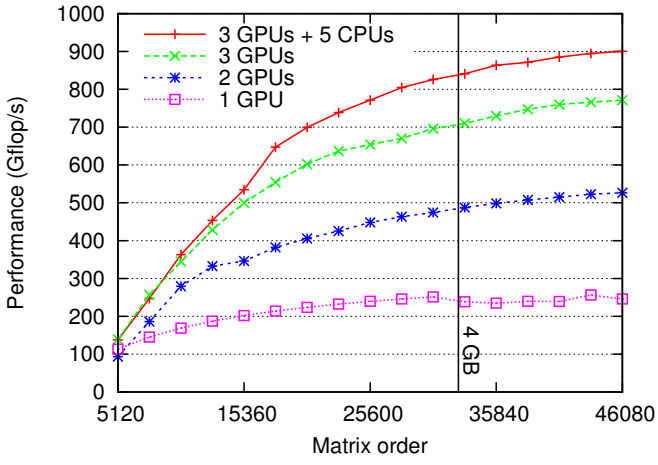

Fig. 2. Performance for the Cholesky factorization in single precision.

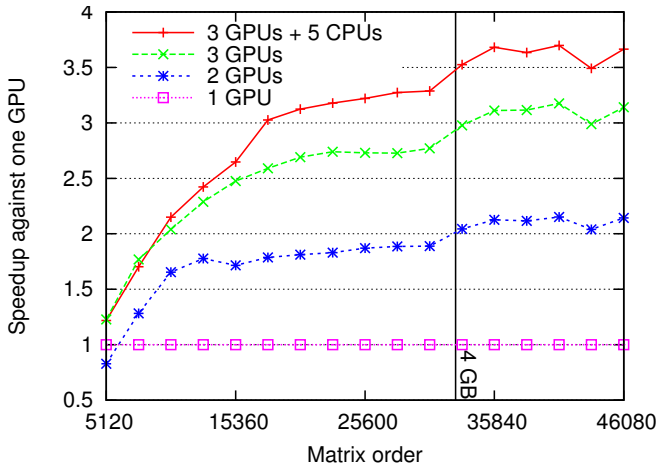

Fig. 3. Speedup for the Cholesky factorization in single precision.

IV. Summary and Future Work

We have shown that a dynamic scheduler allows to achieve a perfect speed up in the case of a node with multiple GPU accelerators. We have furthermore shown that the use of supplementary cores allowed a superlinear speedup. We are now extending this work to other one-sided kernels (*ie.* QR and LU) and two-sided (*eg.* Hessemberg) factorizations. We are also studying the extension of our approach to clusters of accelerator-based platforms.

Acknowledgments

Research reported here was partially supported by the National Science Foundation, the Department of Energy, Microsoft Research, and NVIDIA. This work has also been supported by the ANR through the COSINUS (PROHMPT ANR-08-COSI-013 project) and CONTINT (MEDIAGPU ANR-09-CORD-025) programs. This work was partially supported by the EU as part of FP7 Project PEPPHER (www.peppher.eu) under grant 248481.

References

[1] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Proceedings of the 15th International Euro-Par Conference, Lecture Notes in Computer Science*, 5704:863–874, August 2009.

[2] R. Nath, S. Tomov, and J. Dongarra. Accelerating gpu kernels for dense linear algebra. *Proceedings of VECPAR'10*, 2010.

[3] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov. Numerical linear algebra on emerging architectures: The plasma and magma projects. *Journal of Physics: Conference Series*, Vol. 180, 2009.