LU, QR and Cholesky factorizations: Programming Model, Performance Analysis and Optimization Techniques for the Intel Knights Landing Xeon Phi

DUFOYER Benjamin, CHEVALIER Arthur

1 Introduction

Today's modern computers have a wide variety of heterogeneous compute resources, ranging from multicore CPUs to GPUs and to face up to the increase quantity of numerical informations and modern systems complexity, it is necessary to enhance the structures that manage this task. The main thing of the heterogeous system is the capacity of every ressources to work in the same time. A good task scheduling is paramount to increase performance on this type of system. The biggest problem is to chop every task to execute in the same time while avoiding the competitor accesses. With the multiplication of core number, the developers must change their way of programming in order to slice the problem to be able to parrallelize their application with high-performance. Intel released a new Xeon PHI called KNL (Knight Landing) which has up to 72 cores and this paper is about testing a library called MAGMA in this new highly parallelizable platform. In the matrix factorizations algorithm the MAGMA library provide a way to slice the matrix to ensure that all the tasks will run at the same time and with minimal idle time. The library also use a way to divide all the cores of the KNL in order to simulate multiple devices to use. The article will compare performances over three different matrix factorization algorithms: LU, QR and Cholesky.

Presentation of KNL, KNL is the new version of Xeon Phi. He is on the Intel Family of Many Integrated Core architecture (MIC). He is available in two form, coprocessor and host processor. He will be built using up to 72 cores with four threads per core and use LGA 3647 socket. Each core are engrave at 14nm. It supports for up 384Gb of DDR4 RAM and for up 16Gb of MCDRAM. The MCDRAM is a innovation for the Xeon, it's a 3D-stacked DRAM. The favours of the RAM is the fast access to data (better than classical RAM) on the test, they use the MCDRAM in flat mode, so the MCDRAM have her own adress space and can be adress explicitly. The next innovation of the KNL is the addition of two 512-bit vectorial unit on each core. The KNL achieave a double precision theorical peak of 2662 Gflops/s. The version used in the test is the selfhosted preproduction Intel Xeon Phi card with 16Gb of MCDRAM, each core run at 1,3 GHz. It consists of 64 cores with 4 hyperthreads each. The KNL have different mod of utilisation. You can use in InterConnect mode, in this mode, all core are connected with his neighbor's. Other mode are Cluster Mode with a "All to All", quadrant and Sub-Numa. In the test, they use the Quadrant Mode to increase performance. The principe is simple, the card is cut into four part, every part have a unique Master and the core remaining are a slave.

2 Contribution

The MAGMA (Matrix Algebra on GPU and Multi-core Architectures) project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multi-core and GPU" systems. The goal of MAGMA is to combine the strengths of different algorithms within a single framework. They want to design linear algebra algorithms and frameworks for hybrid manycore and GPUs systemes that can enable applications to fully exploit the power that each of the hybrid components offer. In practice MAGMA want to reduce the communication into every function to have a better parallelism.

3 Implementation elements

LU, QR and Cholesky are three matrix factorizations algorithms and in order to understand the implementation of magma here is the cholesky explaination:

The cholesky decomposition of a Hermitian-definitive matrix A is a decomposition of the form $A = LL^T$ where L is a lower triangular matrix with real and positive diagonal entries, and L^T denotes the conjugate transpose of L. Every hermitian positive-definite matrix (and thus also every real-valued symmetric positive-definite matrix) has a unique Cholesky decomposition. The naive implementation of Cholesky is pretty simple:

We want to decompose the matrix:

$$L = \begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & & \\ . & . & . & & \\ . & . & & . & \\ l_{n1} & l_{n2} & . & . & l_{nn} \end{bmatrix}$$

From the equality $A = LL^T$ we can deduct that:

$$a_{ij} = (LL^T)_{ij} = \sum_{k=1}^{n} l_{ik}l_{jk} = \sum_{k=1}^{min\{i,j\}} l_{ik}l_{jk}, 1 \leq i,j \leq n$$

because $l_{pq} = 0$ if $1 \leq i \leq j \leq n$.

Because A is a symmetric matrix, the above formula has just to be verified for $i \leq j$, in other words all elements $l_{ij}$of the L matrix has to satisfy:

$$a_{ij} = \sum_{k=1}^{i} l_{ik}l_{jk}, 1 \leq i \leq j \leq n$$

For $i = 1$, we determine the first column of L:

$$a_{11} = l_{11}l_{11} \Rightarrow l_{11} = \sqrt{a_{11}}$$

$$a_{1j} = l_{11}l_{j1} \Rightarrow l_{j1} = \frac{a_{1j}}{l_{11}}, 2 \leq j \leq n$$

We can then compute the i-th column of L ($2 \leq i \leq n$) after the computing the (i-1) firsts columns:

$$a_{ii} = l_{i1}l_{i1} + ... + l_{ii}l_{ii} \Rightarrow l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$$

$$a_{ij} = l_{i1}l_{j1} + ... + l_{ii}l_{ji} \Rightarrow l_{ji} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik}l_{jk}}{l_{ii}}, i+1 \leq j \leq n$$

We can see that we have a strong dependance of previous computing to get the next one. MAGMA actually uses the BLAS libraries to compute the LU, QR and Cholesky matrix factorizations. The algorithm uses the Look-Ahead technique and works as follow:

First we factorize the first row and column of the matrix and when we finish this task we update the trailing matrix on the GPU. To use parallelism on this algorithm the idea is to start to factorize the first line and column of the trailing matrix as soon as possible so when the GPU updated them. The factorization works on the CPU to maximize the performances. The BLAS2 library is used on the factorization of single vectors (column and row) and the BLAS3 is used on the update of the trailing matrix. We can also vary the depth level of the algorithm because we can have at the same time multiple updating task on the trailing matrix. When the second factorization ended we launch a second update on the trailing matrix while the first is still running. The number of updating tasks is the depth level.

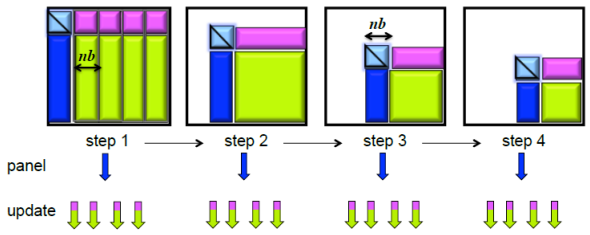

Figure 1. Two-phase implementation of a one-sided factorization

4 Validation/Experiments

Three different configurations has been used for the performances tests:

- The first one configuration has two 18-core Intel Xeon E5-2697 (Broadwell) processors, running at 2,6 GHz. Each socket as 35 MiB of shared L3 cache, and each core has a private 3,5 MiB L2 and 448 KB L1 cache. The system is equipped with 52 GB of memory and the theoretical peak in double precision is 20,8 Gflop/s per core.
- The second system is equipped with two 8-core Intel Xeon E5-2670 (Sandy Bridge) processors and two Intel Xeon Phi 7120 (KNC) cards with 15,8 GB per card. She is equip with 61 core runng at 1,23GHz. And achieving a double precision theoretical peak of 1180 Gflops/s. The KNC is the old version of Xeon Phi and don't have MCDRAM and vector unit.
- The last is a selfhosted preproduction Intel Xeon Phi Knights Landing (KNL) card, with 16 GB of MCDRAM used in flat mode, running at 1.3GHz and achieving a double precision at a theorical peak of 2662 Gflop/s. It has 64 cores with 4 hyper-threads each.

For the KNL, they try with 2 configurations, the first use the KNL with 4 master cores and 60 devices cores and the second one with 4 devices of 15 cores each. The performances are similar with less than 5% difference. It showns that when the kernels of the update reoutine are optimized to perform on more number of core, split hardware over many device is useless. Only one device is enough to have good performance. With this configuration, the device is more simple to program, optimize, observe and debug.

The results ,expressed in GFlops/s, are performed in double precisions and executed on 3 types of configurations in Hybrid mode and native mode. They have performed the tests with the same code for every configuration to see his protability, durability and his capacity to delivered performance near the peak where it is used in native mode.

The tests are performed over the three differents factorizations algorithms with varying matrix size (from 2k to 40k cells for border size). For the LU factorization we can see that behind the 6k size the two KNL and the first one are far over the others with aproximatively twice the performance. The KNL reached its limit of performance at 28k with a peak of 1500Gflop/s et come behind the second configuration that reach the 1700Gflop/s with the 40k matrix. We can also see that the KNL cannot perform tests over 30k matrix because of its memory.

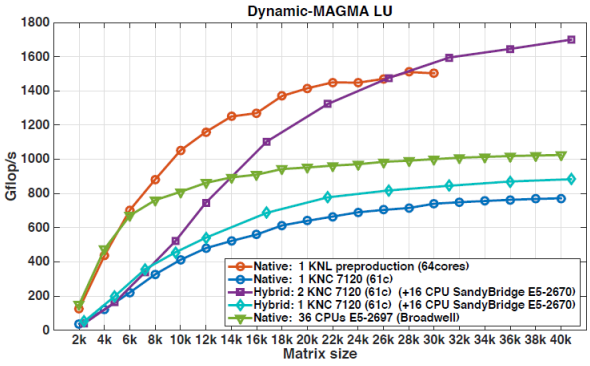

Figure 2. LU performances with MAGMA

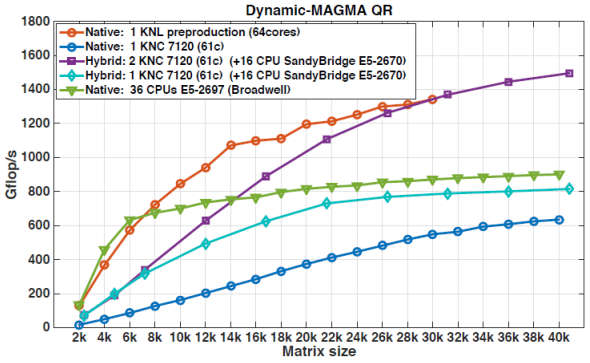

Figure 3. QR performances with MAGMA

For the QR factorization we have the same schema, the third configuration get to follow the KNL until 8k matrix size but get overtaked with taller matrix. With medium matrix size the KNL is still leading with the third configuration like the LU factorization.

Finally for the Cholesky factorization the second config is ahead on small matrix (6k size) but the KNL leads the way until 16k matrix size and the last configuration (two KNC and one processor) get better performance on large matrix.

To sum up this differents tests, the new Xeon Phi proposed by Intel obtain better performances than the older version (KNC) but is not uneatable on performance contest. We can see that it obtains remarquable performances on small matrix but we easily reached the limit of performance on medium matrix.

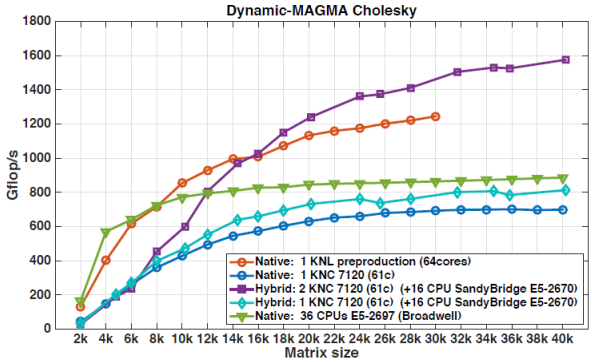

Figure 4. Cholesky performances with MAGMA

5 Conclusion

5.1 Discussion

This article explains the differents optimisations technics on the matrix factorization LU, QR and Cholesky on many-cores systems. The authors show how wise modifications of task management can bring higher performances on heterogeneous systems while using the same alogirthm pattern on all platforms. The performances here allow us to realize that their approach is very good on many-cores systems and make it possible to get high-performance with high scalability along with easy development on different algorithms. Concerning the KNL, we can see that this platform is better than the others on some points compared to the present configurations but show its limits in the actual version. We must remind that this is just a preproduction version and maybe for the future ones they will fix the impossibility to make tests on bigger matrix than 30k of size.

5.2 Perspectives

At this time, the KNL is in preproduction phase and a lot of testing are made to get the maximum performance of this platform. In short terms, we will see some other papers about the decomposition of the algorithm on many-core and others about the necessity to have dynamic scheduling over static. In the long term, the KNL will be used in clusters for high performance processing or in all sort of many-process servers. Finally in this paper they speak about future work including releasing a dynamic MAGMA library that merges CUDA, OpenCL and Intel Xeon Phi into a single software. This is very promising, and if it work, future development on heterogeneous platforms will be easy.

References

Haidar, A. et al. “LU, QR, and Cholesky factorizations: Programming model, performance analysis and optimization techniques for the Intel Knights Landing Xeon Phi”. In: *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. Sept. 2016, pp. 1–7. DOI: 10.1109/HPEC.2016.7761591.

Kurzak, Jakub and Jack Dongarra. *Implementing Linear Algebra Routines on Multi-Core Processors with Pipelining and a Look Ahead*. LAPACK Working Note 178. Also available as UT-CS-06-581. Knoxville, TN 37996, USA: Department of Computer Science, University of Tennessee, Knoxville, Sept. 2006, p. 11. URL: http://www.netlib.org/lapack/lawnspdf/lawn178.pdf;%20http://www.netlib.org/lapack/lawnspdf/lawn178.ps.

Menon, Vijay and Keshav Pingali. “Look Left, Look Right, Look Left Again: An Application of Fractal Symbolic Analysis to Linear Algebra Code Restructuring”. In: *International Journal of Parallel Programming* 32.6 (2004), pp. 501–523. URL: http://dx.doi.org/10.1023/B:IJPP.0000042084.99636.a0.