Toward Auto-tuned Krylov Basis Computations with minimized Communication on Clusters of Accelerators

Langshi CHEN[1],Serge PETITON[1,2],Leroy DRUMMOND[3],Maxime HUGUES[4]

[1] Maison de la Simulation
USR3441, Digiteo Labs Bât 565-PC 190
91191 Gif-sur-Yvette, France
langshi.chen@etudiant.univ-lille.fr
[2] Laboratoire d'informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
59650 Villeneuve d'Ascq, France
Serge.Petiton@lifl.fr
[3] Lawrence Berkeley National Laboratory
One Cyclotron Road Berkeley, CA 94720, US
ladrummond@lbl.gov
[4] INRIA Saclay
1 rue Honor d'Estienne d'Orves
Bât Alan Turing, 91120 Palaiseau, France
maxime.hugues@lifl.fr

Abstract. Krylov Subspace Methods (KSMs) are widely used for solving large scale linear systems and eigenproblems. However, the computing of Krylov subspace basis for KSMs suffers from its intensive blocking scalar product computation and communication, especially in large clusters with accelerators like GPUs. In this paper, a Hyper Graph based communication optimization is applied to Arnoldi and incomplete Arnoldi methods of forming Krylov basis, and we compare their performance with classic Arnoldi methods within a CPU-GPU framework. Results show the benefits from optimization and its drawbacks which require further integration of auto-tuning technologies.

Keywords: Krylov Subspace, Auto-tuning, Arnoldi Orthogonalization

1 Introduction

Krylov Subspace Methods (KSMs) (such as GMRES and Arnoldi method) are kinds of iterative solvers frequently used in large scale linear problems. In KSMs, a basic and important part is to generate an orthogonal basis for the Krylov subspace. Arnoldi method is commonly adopted to form the basis, but it is proved to be time-consuming due to its blocking scalar product from its orthogonalization process. In the parallel framework, the matrix-vector product in Arnoldi

method also causes a heavy communication cost. It is even worse in clusters equipped with accelerators like GPU, since the data exchange among GPUs is still expensive. Thus, efforts are made to reduce the communication in KSMs. Ghysels et al. [1] has proposed a pipelined variation of GMRES, hiding the global communication latencies by overlapping them with other communication or computations. Hoemmen [2] has implemented a Communication Avoiding version of the Power method for computing non-orthogonal bases, which replaces data exchange by redundant local computation. In this paper, we apply a Hyper Graph based communication optimization to parallel Arnoldi and incomplete Arnoldi orthogonalization methods. Together with the non-optimized Arnoldi and incomplete Arnoldi methods, the four algorithms are tested within a CPU-GPU framework. Our evaluation and comparison of performance concentrate only on the time spent in the computing of a Krylov subspace basis. While the number of restarts and the total time for obtaining a converged solution also depend on conditions such as the features of target matrices, which makes it difficult to have a general analysis. For example, methods like incomplete Arnoldi could generate fast a less orthogonal basis but later require more iterations and time for obtaining a convergent solution.

2 Methods for Generating Krylov Subspace Basis

In order to obtain a vector basis for the Krylov subspace, the Gram-Schmidt orthogonalization based Arnoldi process is commonly used. This Arnoldi method could be implemented in a parallel framework, and we will also study the incomplete Arnoldi method and two communication optimized Arnoldi and incomplete Arnoldi methods.

2.1 Arnoldi

First, we focus on a parallel version of the *Arnoldi* algorithm that uses a Classic Gram-Schmidt process to form a full-sized orthogonal subspace basis. We evaluate its computational complexity in Equation 1

$$T_{Arnoldi}(r,p,n,\sigma) = \frac{2\alpha\sigma nr}{p} + \frac{3\alpha nr^2+(2p+1)\alpha nr}{2p} + r\Big[2\beta\Theta(p) + nG\Big] \qquad (1)$$

where r is the size of Krylov subspace, p is the number of MPI tasks, n is the dimension of test square matrix and σ is the density of nonzero entries. α is the time of an arithmetic operation; β is the latency of the sending (receiving) messages, and G is the time per byte (Gap between sending or receiving long message).The term $(2\alpha\sigma nr)/p$ corresponds to the time consumed by matrix-vector product to form a non-orthogonal basis. The term $[3\alpha nr^2+(2p+1)\alpha nr]/2p$ represents the scalar product from the orthogonalization process, and the last term $r[2\beta\Theta(p) + nG]$ is the cost of communication among MPI processes.

2.2 IArnoldi(q)

Secondly, we view the *IArnoldi(q)* which is based on the Classic Gram-Schmidt process. Meanwhile, it truncates the number of orthogonal vectors so that each new generated basis vector should only be orthogonal to its q previous basis vectors [3]. We also evaluate its complexity in Equation 2.

$$T_{IArnoldi}(r,p,n,\sigma,q) = \frac{2\alpha\sigma nr}{p} + \frac{\alpha nr(p+3q+2)}{p} + r\Big[2\beta\Theta(p) + nG\Big] \quad (2)$$

The truncation accelerates the second term in Equation 1 by replacing it with $[\alpha nr(p+3q+2)]/p$. This term is only proportional to the r and thus greatly reduces the time in the case of large subspace. However, the truncation also reduces the orthogonality of the subspace basis, which may increase the number of iterations to obtain a convergent solution.

2.3 ArnoldiHG

Similarly, *ArnoldiHG* is a communication optimized version of *Arnoldi*. It uses a Hyper Graph model [4] to describe and optimize the communication cost in the matrix-vector multiplication $Y = AX$ of Arnoldi methods. The matrix A is partitioned by rows in every MPI task, and every MPI task requires communication with other tasks to update Y for the next iteration. Thus, the communication optimization could be abstracted as a problem of hypergraph (row based) partition.

The modeled complexity in Equation 3 shows that the last term $r[n\sigma G\Theta(p)+\beta\Theta(p^2)]$ is the communication part after applying optimization which is much smaller than that of original Arnoldi implementation.

$$T_{ArnoldiHG}(r,p,n,\sigma) = \frac{2\alpha\sigma nr}{p} + \frac{3\alpha nr^2+(2p+1)\alpha nr}{2p} + r\Big[n\sigma G\Theta(p) + \beta\Theta(p^2)\Big] \quad (3)$$

2.4 IArnoldiHG(q)

IArnoldiHG(q) is the Hyper Graph based communication optimized *IArnoldi(q)*. Due to the truncation and the communication optimization, the second and third terms in Equation 4 show a significant benefit in execution time.

$$T_{IArnoldiHG}(r,p,q,n,\sigma) = \frac{2\alpha\sigma nr}{p} + \frac{\alpha nr(p+3q+2)}{p} + r\Big[n\sigma G\Theta(p) + \beta\Theta(p^2)\Big] \quad (4)$$

2.5 Comparison of four Algorithms

In Table 1, we compare the features of the four algorithms. Both of them adopt the Classic Gram-Schmidt process (CGS). Though it is less numerically stable than Modified Gram-Schmidts (MGS), it is more suitable within a parallel Arnoldi process. *ArnoldiHG* and *IArnoldiHG(q)* have optimized communication, leading to better scalability than the others. However, *IArnoldi(q)* and *IArnoldiHG(q)* have a weak orthogonality, which requires methods like Reorthogonalization to improve the basis.

Table 1: Comparison of 4 parallel Arnoldi Algorithmic Implementations

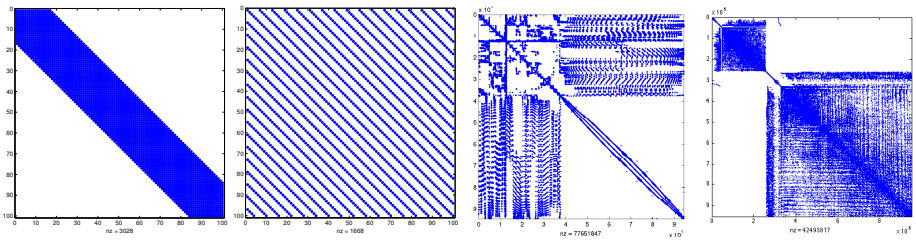
Algo Name	Arnoldi	ArnoldiHG	IArnoldi(q)	IArnoldiHG(q)
Orthogonality	Full Size	Full Size	q previous	q previous
Optimized Communication	NO	Yes	NO	Yes
Execution Time	High	Medium	Medium	Low
Strong Scalability	Medium	High	Medium	High

3 Experimentation

The four algorithms described in Table 1 are implemented and tested within a CPU-GPU framework. The sparse matrix A is partitioned by rows in a blockwise way. Each partition is assigned to a host CPU process to handle the communication and another GPU to handle local computation task.

3.1 The Test Matrices

The experiment has tested the four Algorithms on sparse matrices with different structures. First, we choose the Continuous Diagonal Matrix (Figure 1 (a)), where each process's submatrix only communicates with its neighbour processes. Secondly, we adopt the Equidistributed Diagonal Matrix (Figure 1 (b)), where nonzero entries are distributed evenly across all columns. Finally, we use two real sparse matrices of the sparse matrix collection from the University of Florida. We also use two different Sparse Matrix formats. One is the Compressed



(a) C-Diagonal **(b)** Equi-Diagonal **(c)** Audikw1 **(d)** Ldoor

Fig. 1: (a) Continuous Diagonal Matrix, size of 960000, diagonals of 33; (b) Equidistributed Diagonal Matrix, size of 960000, diagonals of 33; (c) Audikw1, size of 943695, nonzeros of 77651847; (d) Ldoor, size of 952203 nonzeros of 42943817

Sparse Row format (CSR format), the other is the ELLPACK format, and we study the potential impact of these formats in the overall performance of the implementations

3.2 The Krylov Subspace Size

During the construction of the subspace basis, one of the most important parameters is the subspace size r (See Section 2). The value of r would greatly affect the performance of Arnoldi process. Normally, a small value of r is preferred in the cases of *IArnoldi(q)* and *IArnoldiHG(q)*, because the loss of orthogonality would be reduced. In the test, we also use some large values of r to have a good range of tests and study the influence of the subspace size over the performance of the algorithmic implementation.

4 Results and Analysis

The results are from tests performed on the GPU cluster of machine *Poincare* from *Maison de la Simulation*. The machine is composed of IBM's iDataPlex dx360 M4 servers. Its GPU cluster has four nodes, and each of them is equipped with two processors of Sandy Bridge E5-2670, and two GPU Tesla K20 (CUDA Capability 3.5, 4.8 G memory for each GPU); Each node also has shared memory of 64G. The connection among nodes uses the QLogic QDR InfiniBand solution. The GPU codes are compiled by the CUDA version 5.5, and the CPU codes are compiled by openMPI version 1.6.3.

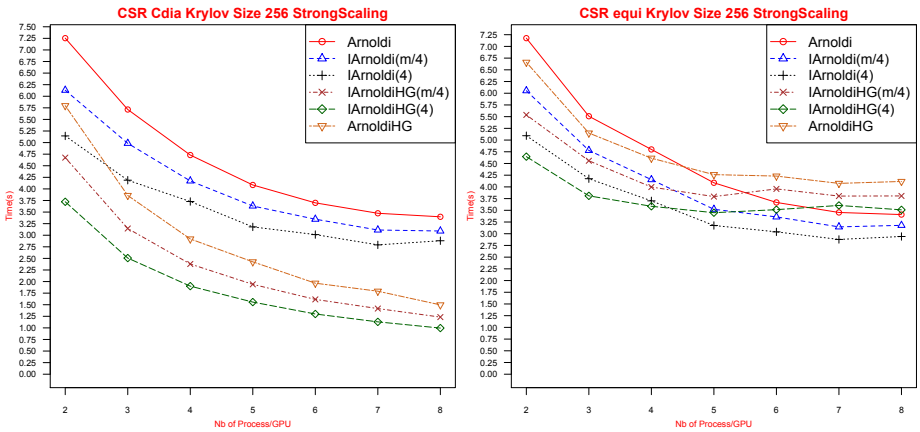
4.1 Different Structure of Input Matrix

The results in Figure 2 show their strong scalability performance in total execution time, which are coherent with the complexity analysis we made in Section 2. The computation time of $(2\alpha\sigma nr)/p$ and $[3\alpha nr^2 + (2p+1)\alpha nr]/2p$ (or $[\alpha nr(p+3q+2)]/p$ in incomplete Arnoldi) reduces when p increases. For the communication part, the time increases with the process number p, but also depends on the structure of matrices. In Figure 2 (a), MPI process in C-Diagonal matrix only has communication with its neighbours, making the communication part negligible (best case), so *ArnoldiHG* outperforms *Arnoldi*, and *IArnoldiHG(q)* is better than *IArnoldi(q)*.

In Figure 2 (b) we find that the communication optimized methods have lost their advantage in strong scalability. In Equation 4, their communication cost is $r[n\sigma G\Theta(p) + \beta\Theta(p^2)]$, while for *Arnoldi* algorithm, this part becomes $r[2\beta\Theta(p) + nG]$. Here G has the order of $10^{-10} \sim 10^{-9}$ s, while β has the order of $10^{-6} \sim 10^{-4}$s. When n is relatively small, the former is of the order of $\beta\Theta(p^2)$, while the latter is of the order of $\beta\Theta(p)$. Thus, the execution time of optimized methods would surpass that of *Arnoldi* in the worst case (e.g. Equi-Diagonal Matrix). In Figure 2 (c) and (d), the performance of the optimized algorithms are between that of Figure 2 (a) and Figure 2 (b).

4.2 Varying the size of the Krylov subspace

In Figure 3, *IArnoldi(q)*'s benefit over *Arnoldi* decreases when the size of Krylov Subspace goes down. It can be explained by a difference term $\Theta[(r-q)\alpha nr]$ from

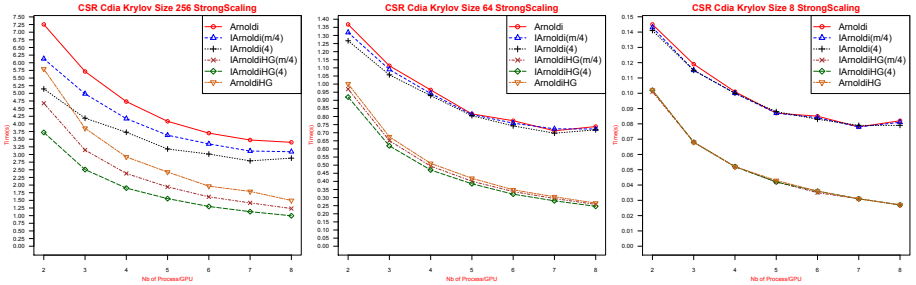


(a) C-Diagonal **(b)** Equi-Diagonal

(c) Audikw1 **(d)** Ldoor

Fig. 2: The Execution time for Strong Scaling tests with 4 different sparse matrices. The Krylov subspace is fixed to 256, we use the CSR matrix format and double precision arithmetic

the scalar product part of the two methods. The advantage of communication optimized methods is unaffected by the change of Krylov subspace size. However,



(a) Krylov Size 256 **(b)** Krylov Size 64 **(c)** Krylov Size 8

Fig. 3: Execution time evaluation of 4 algorithms for 960000 × 960000, 33 diagonals C-Diagonal Matrix, with format CSR, double precision and 3 different Krylov Size: 256, 64 and 8

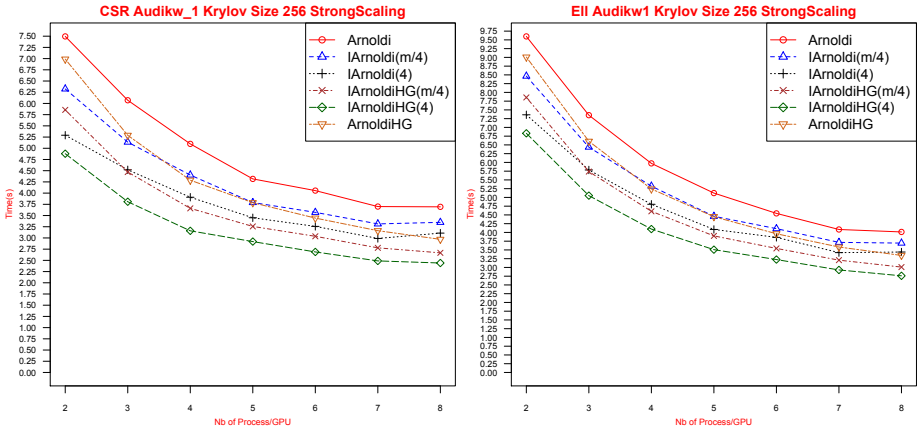
the subspace size may affect the number of iterations for obtaining a convergent solution. Thus, an auto-tuning strategy for dynamically optimizing the size of the subspace is required [5].

4.3 Impact of Sparse Matrix Format

In [6], the influence of formats on the performance of SpMV has been evaluated. Similarly, we compare the two formats CSR and ELLPACK in our test. In Figure 4, the format CSR runs slightly better than the ELLPACK for matrix Audikw1, and we observed the same phenomena from tests on the other three matrices.

5 Conclusion

In this paper, we presented a Hyper-Graph based parallel implementation of a Krylov based methods using optimized communication schemes. The performance results corroborate our theoretical performance models (Eqs.1,2,3,4) and our test-cases studied the influence of the Krylov subspace size, the sparse matrix structure and storage format on the performance of the algorithmic implementations on CPU-GPU platforms. Furthermore, we arrived on the following concluding remarks; 1) The hyper graph based optimization is effective for different Krylov subspace sizes and different sparse matrix formats. While the incomplete Arnoldi method with our optimization performs best among the four algorithmic implementations, it still needs an auto-tuning strategy to optimize parameters like subspace size and number q of truncated vectors in runtime. 2) The hyper graph optimized methods do not perform well in sparse matrices



(a) Audikw1 CSR **(b)** Audikw1 ELLPACK

Fig. 4: Execution time of 4 algorithms for matrix Audikw1 using CSR and ELLPACK format. The Krylov Subspace size is 256 and a double precision is used

with certain structures. A high level auto-tuning strategy is required to choose optimized or non-optimized communication methods according to the inputs. 3) Communication Avoiding strategy shall be integrated to further eliminate the communication bottleneck in parallel Krylov basis computing. To attain further improvements, we will consider the adoption of Communication Avoiding strategies [2] in our future work.

References

1. Ghysels, P., Ashby, T.J., Meerbergen, K., Vanroose, W.: Hiding global communication latency in the gmres algorithm on massively parallel machines. SIAM journal on scientific computing **35** (2013) C48–C71
2. Hoemmen, M.: A communication-avoiding, hybrid-parallel, rank-revealing orthogonalization method. In: Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International. (2011) 966–977
3. Saad, Y., Wu, K.: Dqgmres: a direct quasi-minimal residual algorithm based on incomplete orthogonalization. Numerical Linear Algebra Appl **3** (1996) 3–329
4. Catalyurek, U.V., Aykanat, C.: Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. IEEE trans. on parallel and distributed computing **10** (1999) 673–693
5. Katagiri, T., Aquilanti, P.Y., Petiton, S.G.: A smart tuning strategy for restart frequency of gmres(m) with hierarchical cache sizes. In: VECPAR. (2012) 314–328
6. Hugues, M., Petiton, S.: Sparse matrix formats evaluation and optimization on a gpu. In: High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on. (2010) 122–129