A Communication Optimization Scheme for Basis Computation of Krylov Subspace Methods on Multi-GPUs

Langshi Chen[1(✉)], Serge G. Petiton[1,2], Leroy A. Drummond[3], and 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, USA
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 computation of Krylov subspace bases suffers from the overhead of performing global reduction operations when computing the inner vector products in the orthogonalization steps. In this paper, a hypergraph based communication optimization scheme is applied to Arnoldi and incomplete Arnoldi methods of forming Krylov subspace basis from sparse matrix, and features of these methods are compared in a analytical way. Finally, experiments on a CPU-GPU heterogeneous cluster show that our optimization improves the Arnoldi methods implementations for a generic matrix, and a benefit of up to 10x speedup for some special diagonal structured matrix. The performance advantage also varies for different subspace sizes and matrix formats, which requires a further integration of auto-tuning strategy.

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 [1]. 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 [2,3], but it is proved to be time-consuming due to its blocking scalar product from its orthogonalization

M. Daydé et al. (Eds.): VECPAR 2014, LNCS 8969, pp. 3–16, 2015.
DOI: 10.1007/978-3-319-17353-5_1

process. In a parallel framework, the matrix-vector product in Arnoldi method also can cause 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. [4] has proposed a pipelined variation of GMRES, hiding the global communication latencies by overlapping them with other communication or computations. Hoemmen [5] 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 hypergraph 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 to reach convergence.

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. In this paper, we focus on the Arnoldi process in the sparse matrix case. Arnoldi consists of a BLAS 2 bandwidth bounded Sparse matrix-vector multiplication (SpMV) operations, and a vector inner product operations which incurs a global data reduction across all the MPI processes. In order to reduce the communication overhead, we first introduce a variant of *Arnoldi* named *Incomplete Arnoldi*, which truncates the number of inner product operation to lower down the communication from global reduction. Then, we introduce our hypergraph based optimization and apply them to both *Arnoldi* and *Incomplete Arnoldi*. A time complexity analysis is given in Sects. 2.1, 2.2, 2.3 and 2.4 for the four algorithms we have studied here.

2.1 Arnoldi

Arnoldi method uses a Classic Gram-Schmidt (CGS) process to form a full-sized orthogonal subspace basis. The CGS is preferred to Modified Gram-Schmidt (MGS) because it is easier to implement in parallel and is better for the scalability of our implementations. We evaluate its computational complexity in Eq. 1.

$$\begin{aligned} T_{Arnoldi}(s,p,N,n) = \alpha(2sn/pL) \ & +\alpha(3Ns^2+9Ns)/2pL \\ & +\beta(2s\log_2(p)) + G(Ns+2s^2)(p-1)/p \end{aligned} \quad (1)$$

The variable s is the size of Krylov subspace; p is the number of MPI processes; N is the row number of matrix and n is the number of nonzero entries. We divide the overall time into three parts: (1) $\alpha(2sn/pL)$ is the time of the matrix-vector multiplication in Gram-Schmidt process. The parameter α denotes the time per arithmetic operation, and L is the number of simultaneous parallel processing elements in each MPI process (e.g. maximal threads within a GPU). (2) $\alpha(3Ns^2 + 9Ns)/2pL$ is the time spent in the vector inner product of Gram-Schmidt process, which is quadratic to the subspace size s. (3) $\beta(2s\log_2(p)) + G(Ns + 2s^2)(p-1)/p$ is the communication overhead. It includes a latency part β and a bandwidth part G. The complexity analysis indicates that the communication overhead augments with the number of MPI processes p. When p is relatively large, the latency part will become the dominant increment to the communication cost.

2.2 IArnoldi(q)

Secondly, we review the *IArnoldi(q)* based on the Classic Gram-Schmidt process. *IArnoldi(q)* truncates the number of orthogonal vectors so that each new generated basis vector should only be orthogonal to its q previous basis vectors [6]. Its time complexity is evaluated in Eq. 2.

$$\begin{aligned} T_{IArnoldi(q)}(s,p,N,n,q) = \ & \alpha(2sn/pL) + \alpha(2q+3)Ns/pL \\ & + \beta[(s+q)\log_2(p)] + G[(Ns+2qs)(p-1)/p] \end{aligned} \quad (2)$$

s, p, N, n is the same parameters in Eq. 1, and q is the number of vectors each new generated basis vector should be orthogonal. Thus, a small q value means a large extent of truncation and a less time both in computation and communication. The second term $\alpha(2q+3)Ns/pL$ reduces the time of inner products from $O(Ns^2/pL$ to $O(Nsq/pL)$. The time of latency part also drops from $\beta(2s\log_2(p))$ to $\beta(s\log_2(p))$ (we assume that $q \ll s$). However, it suffers from a less orthogonal basis, which incurs more iterations to reach convergence.

2.3 ArnoldiHG

ArnoldiHG is an optimized *Arnoldi* based on an hypergraph model proposed in [7]. As the Power iteration of SpMV in *Arnoldi* is communication bounded, we model it as an hypergraph. Each vertex in hypergraph refers to a single row in matrix A, and two vertex (row i and j) are connected by a edge if and only if the entry $A(i,j)$ is nonzero. The edge (i,j) means that in each power iteration of SpMV, the multiplication of row $A(i,;)$ and vector X requires the data $X(j)$ stored in row j. In a parallel implementation, the rows of A are partitioned and assigned to different processes (CPU or GPU computation element). Thus, an edge across two groups of vertex (rows) incurs a data exchange within processes, and this dependency is invariant during the power method iteration. Then the optimization takes two steps. It first finds an optimal partition of the rows in A to minimize the data exchange. Then the communication only occurs

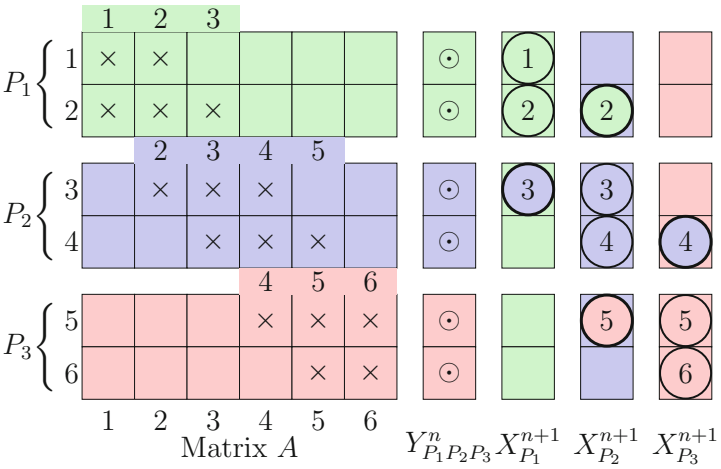


Fig. 1. Matrix A is divided into 3 partitions. When update X^{n+1} from Y^n, each partition only requires some X data depending on their nonzero entries. E.g. P_2 (Blue color) requires the data from $2, 3, 4, 5$ row positions, because only column $2, 3, 4, 5$ has nonzero entries. When forming $X^{n+1}_{P_2}$, data $3, 4$ could be directly copied from P_2's own memory, but communication occurs when requires data $2, 5$ which belongs to P_1, P_3. Thus for P_2, the minimal communication is receiving data $2, 5$ and sending data $3, 4$ (Color figure online)

among rows that have data dependency in each iteration of SpMV. Figure 1 gives an example of the optimization scheme. After the optimization, the communication overhead $\beta(2s\log_2(p)) + G(Ns + 2s^2)(p-1)/p$ in Eq. 1 is replaced by $\{\beta[sf(A,p) + s\log_2(p)] + G[sg(A,n) + 2s^2(p-1)/p]\}$, where the terms $f(A,p)$ and $g(A,n)$ depends on the structure and partition of matrix A. In a best case, we have $f(A,p) = O(1)$, $g(A,n) = O(n/p)$ which greatly reduces the communication overhead in SpMV part.

2.4 IArnoldiHG(q)

Similarly, *IArnoldiHG(q)* is the hypergraph optimized *IArnoldi(q)*. Due to the truncation and the hypergraph optimization, the second and third terms in Eq. 3 show a significant improvement at execution time.

$$T_{IArnoldi(q)}(s,p,N,n,q) = \alpha(2sn/pL) + \alpha(2q+3)Ns/pL + \beta[q\log_2(p) + sf(A,p)] + G[sg(A,n) + 2qs(p-1)/p] \quad (3)$$

2.5 Comparison of Four Algorithms

In Table 1, we compare the four algorithms in various aspects, like the execution time, orthogonality and scalability. Here the total execution time for computing a basis is divided into three parts as presented in Sects. 2.1, 2.2, 2.3 and 2.4. The orthogonality evaluates the quality of the basis, which shall affect the total

Table 1. Comparison of 4 parallel Arnoldi Algorithmic Implementations

Algo name	Arnoldi	ArnoldiHG	IArnoldi(q)	IArnoldiHG(q)
Computation	$O(2sn/pL + 3s^2N/2pL)$	$O(2sn/pL + 3s^2N/2pL)$	$O(2sn/pL + 2qsN/pL)$	$O(2sn/pL + 2qsN/pL)$
Latency	$O(2s\log_2(p))$	$O(sf(A,p) + s\log_2(p))$	$O(s\log_2(p))$	$O(sf(A,p) + q\log_2(p))$
Bandwidth	$O(Ns)$	$O(sg(A,n) + 2s^2)$	$O(Ns)$	$O(sg(A,n) + 2qs)$
Orthogonality	Full	Full	Depend q	Depend q
Strong scalability	Medium	High	Low	Medium
Weak scalability	Low	High	Low	High
Optimized communication	NO	Yes	NO	Yes

time for finding a converged solution in iterative methods. The Strong scalability reflects the parallelism of the computational workload, a more computational intensive algorithm shall have a better strong scalability. While the weak scalability measures the communication overhead, where a communication intensive algorithm has a worse performance. According to our analysis, *IArnoldiHG* has the best execution time due to the truncation and our hypergraph optimization. Both of the hypergraph optimized methods have better scalability because of their reduced communication overhead. Nevertheless, they are affected by the structure of the matrix which determines the factor $f(A,p)$ and $g(A,n)$. The truncated versions of *Arnoldi* method do not have a full orthogonality of its basis, which depends on the choice of value q. A relatively small q could reduce the execution time but reduce the orthogonality at the same time, which may be remedied by methods like reorthogonalization.

3 Experimentation

In the experiment, we test the four methods on two heterogeneous clusters. One is the cluster *Poincare* from *Maison de la Simulation*, which has 4 GPU nodes and each node consists of 2 Xeon CPU and 2 Nvidia K20m GPUs. Another cluster *HAPACS-TCA* resides in the University of Tsukuba. It has a total 64 GPU nodes, and each node contains 2 Xeon CPUs and 4 Nvidia K20x GPUs (See Table 2). According to this heterogeneous architecture, the four algorithms presented in Table 1 are also implemented under a CPU-GPU hybrid programming paradigm (See Fig. 2). The matrix A is evenly row partitioned and the workload is distributed over GPUs. Each GPU has its own CPU process to handle the communication operations via MPI. The GPU codes are written in CUDA 5.5 framework, and the MPI codes are compiled by openmpi 1.6.3. In order to

Table 2. Two GPU cluster's characteristics

Name	Nodes	CPU/node	GPU/node	GPU	Interconnect
Poincare	4	2	2	K20m	Infiniband
HA-PACS/TCA	64	2	4	K20x	Infiniband

test the influence of different matrix structure to our hypergraph model, we use sparse matrices with different structures. The Krylov Subspace size is also varied to test its effect on the truncated methods *IArnoldi* and *IArnoldiHG*.

3.1 The Test Matrices

In the experiment, we use four test sparse matrices. The Continuous Diagonal Matrix (C-Diag) in Fig. 3(a), has all its subdiagonals continuously aggregates around the main diagonal. It represents a major type of matrices generated by PDE applications, and it maps well to a row partition where each row part only communicates with its neighbours. The second matrix is the Equidistributed Diagonal Matrix (E-Diag) in Fig. 3(b), which is a contrast to the C-Diag matrix

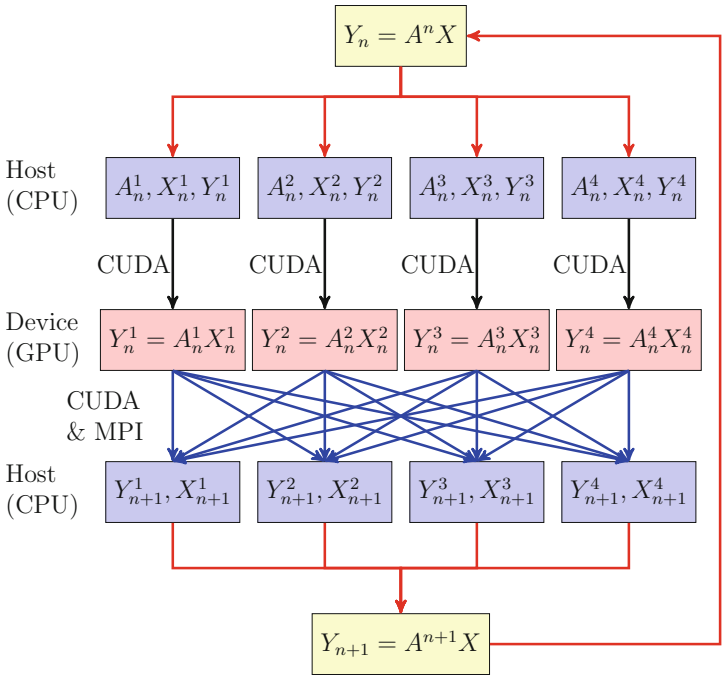
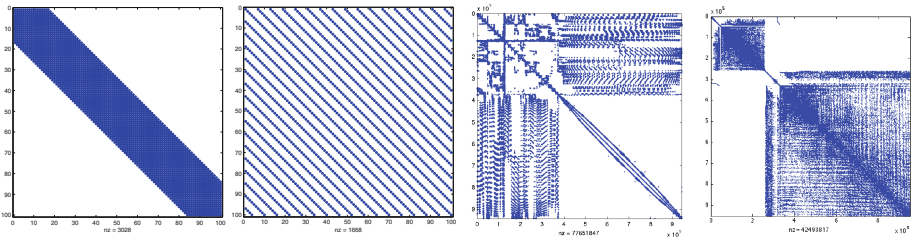


Fig. 2. A hybrid CPU-GPU programming scheme for the *SpMV* part of *Arnoldi*. At every iteration, a group of rows of the matrix A is attached to a CPU and a GPU. A CPU first delivers the computation task to a GPU. After the GPU has finished its work, each CPU gathers the Y vector from its own GPU, and it gathers Y vector or part of Y vector from other CPUs via MPI. Then a new X vector is formed in CPU for the next iteration



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

Fig. 3. (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

where subdiagonals are distributed evenly across all columns. In equal row partition of E-Diag, each row part shall exchange data with distant rows, which may examine the worst case of hypergraph optimization model. Both of C-Diag and E-Diag matrices are generated in runtime, which could be extended to an arbitrary scale in the strong scaling and weak scaling tests. The third and fourth matrices are real matrices collected from the Florida University's Sparse matrix database as shown in Fig. 3(c) and (d). They have more complicated structure which is taken to test the influence of matrix structure on the hypergraph model in the strong scaling tests. We also use two different Sparse Matrix formats. One is the Compressed Sparse Row format (CSR format), the other is the ELLPACK format provided by *Nvidia*, and we also evaluate 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 s (See Sect. 2). The value of s would significantly affect the performance of *IArnoldi* methods. Typically, a small value of s is preferred in the cases of *IArnoldi* and *IArnoldiHG* because the loss of orthogonality would be reduced. In the test, we also use some large values of s to have a complete range of tests and study the influence of the subspace size over the performance of time and scalability.

4 Results and Analysis

Figures 4 and 5 shows the scalability results of the four algorithms on C-Diag matrix presented in Sect. 3.1. The Krylov subspace size is set to 64, and we use a commonly adopted sparse matrix format CSR. In Fig. 4, the strong scalability of the four algorithms is not good, which is due to the low computation/communication ratio in the sparse matrix case. While the result shows a significant benefit from hypergraph optimization. When the algorithms are scaled to more than 100 GPUs, *ArnoldiHG* could achieve a nearly 10x speed up over

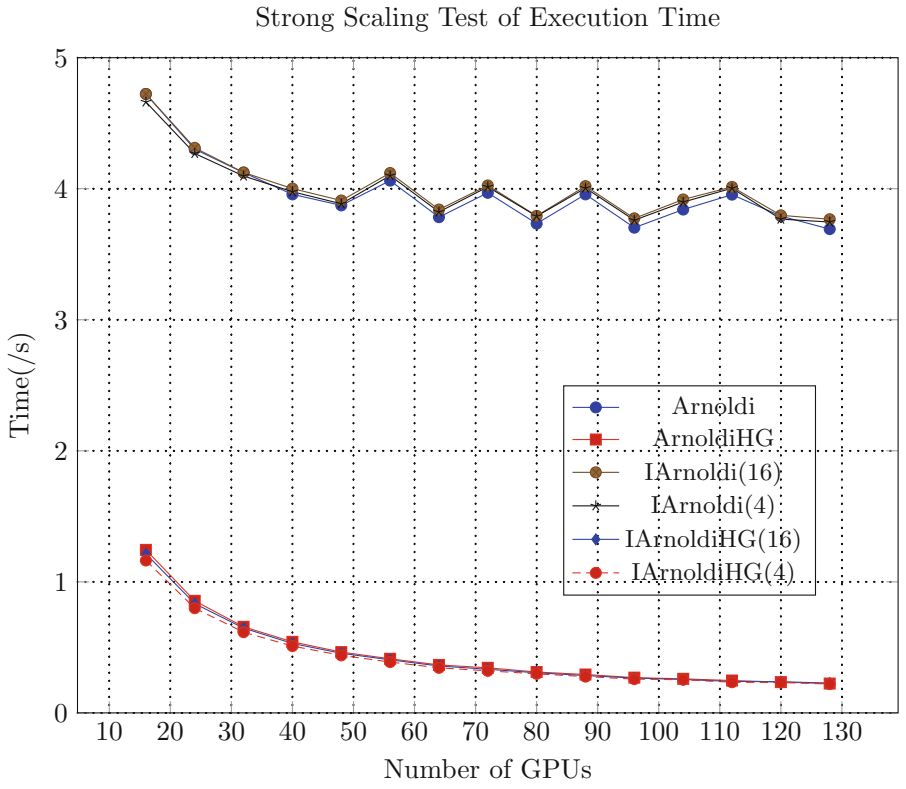


Fig. 4. Scalability Test on *HAPACS-TCA* with C-Diag matrix stored in CSR format double precision. Strong scaling with total matrix size 9600000×9600000, diagonals of 33 and Krylov subspace size 64

non-optimized *Arnoldi*. This benefit comes from the reduction of communication overhead in the C-Diag matrix, where each process (GPU) shall only exchange data with its neighbours in the hypergraph model. The weak scaling test in Fig. 5 endorses our model as well, where *ArnoldiHG* and *IArnoldiHG* show a perfect weak scalability over 100 GPUs. In contrast, the non-optimized methods suffer a lot from their communication overhead. In Fig. 6, we have a comparison for the two methods. When the number of GPUs is relatively small, *IArnoldiHG(4)* has the best performance. It comes from the benefit of *IArnoldiHG*'s truncation on the number of vector inner product operations. A smaller q value leads to a better performance gain (*IArnoldiHG(4)* outperforms *IArnoldiHG(16)*). However, this advantage disappears as the number of GPUs increases, and the performances of the three curves in Fig. 6 converge. It means that the dominance of computation is eliminated when p is large in the computation part of Table 1. As the communication is also optimized by hypergraph model, a performance convergence is expected. Thus, we prefer *IArnoldiHG* when p is small; otherwise *ArnoldiHG* shall be chosen as it has a better orthogonality.

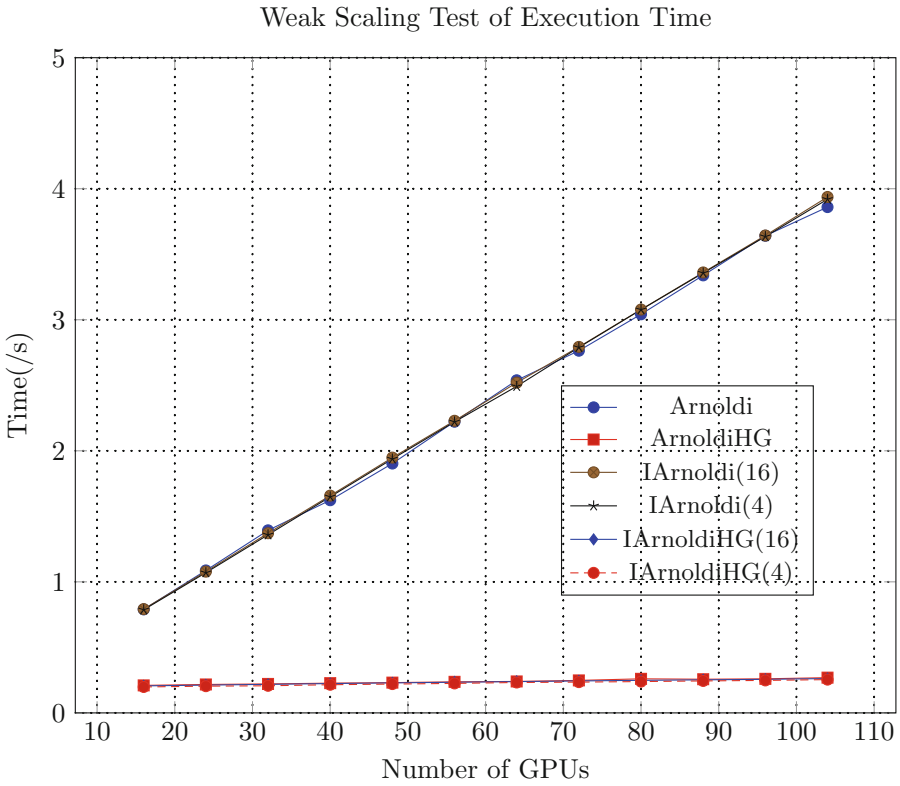


Fig. 5. Scalability Test on *HAPACS-TCA* with C-Diag matrix stored in CSR format double precision. Weak scaling with total matrix size 9600000 × 9600000, diagonals of 33 and Krylov subspace size 64

4.1 Different Structure of Input Matrix

The structure of sparse matrix may also affect the performance of hypergraph optimization. We will first test an E-Diag matrix both in strong scaling and weak scaling. E-Diag could be considered as a worst-case test for our optimization model, where each process should communicate with distant processes. In Fig. 7(a), we find a cross point at around 50 processes (GPUs). As the bandwidth part of *ArnoldiHG* $O(sn/p + 2s^2)$ is much lower than that of *Arnoldi*'s $O(Ns)$, the crossing point could be explained by the latency part of *ArnoldiHG* $O(sf(A,p) + slog_2(p))$ with $f(A,p) = min(p, diags)$. In our case, the number of subdiagonals is 33. When p increases to pass a particular value, the latency part of *ArnoldiHG* is surpassed by that of *Arnoldi* ($log_2(p)$), and a benefit of our optimization is expected. The weak scaling result also supports our analysis, where a crossing point is found in the same place around 50 GPUs. Besides the E-Diag matrix, we also test two real sparse matrices *Audikw1* and *Ldoor* presented in Fig. 3. As they have a fixed size, we only take the strong scaling test on them. In Fig. 8, we find that our hypergraph model optimization is effective on both of

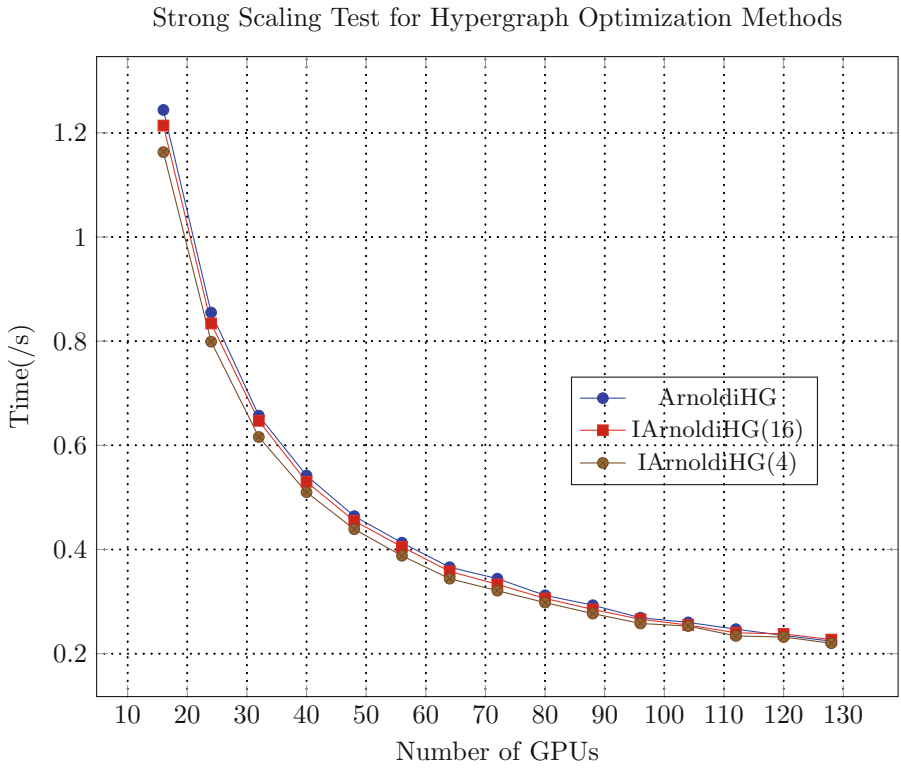
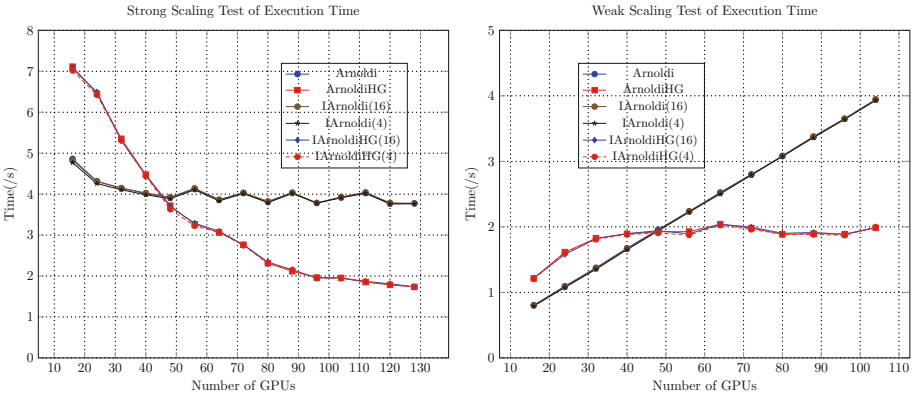


Fig. 6. Scalability Test of Hypergraph Optimization methods on *HAPACS-TCA* with C-Diag matrix stored in CSR format double precision. Strong scaling with total matrix size 9600000×9600000, diagonals of 33 and Krylov subspace size 64

the matrices with complicated structure. The matrix *Ldoor* gains more benefits from our optimization compared to the matrix *Audikw*. As the two matrices have a row size and number of nonzeros of the same order, the difference should come from their different structures. According to the Fig. 3, *Ldoor* has more of its nonzero entries aggregated around its main diagonals than that of *Audikw*, which shall reduce the communication among distant processes (GPUs in our test). Thus, we may have a conjecture that the benefit of our model depends on the structure of the matrix (e.g. the number of subdiagonals in E-Diag matrix) in some cases.

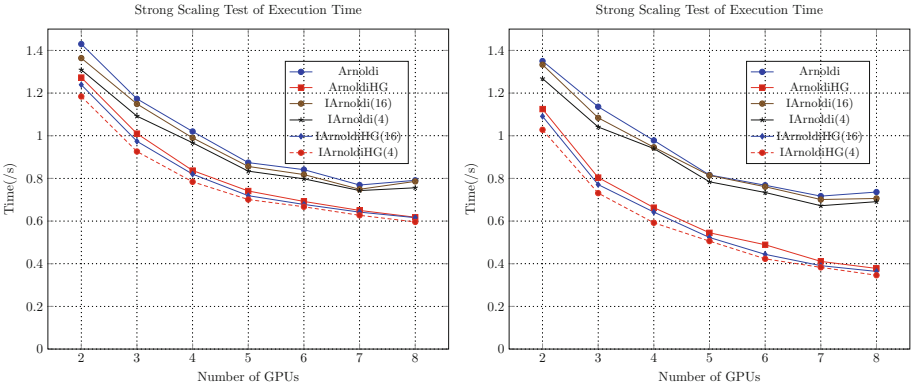
4.2 Varying the Size of the Krylov Subspace

Furthermore, we study the influence of the various Krylov subspace sizes on our hypergraph optimization. Since the Krylov subspace size is one of the most important parameters in many Krylov iterative methods, we choose three different values $8, 64, 256$ in our test. A much larger value is not preferred in KSMs because it shall involve a very large workload. Instead, this value is generally set



(a) Strong Scaling **(b)** Weak Scaling

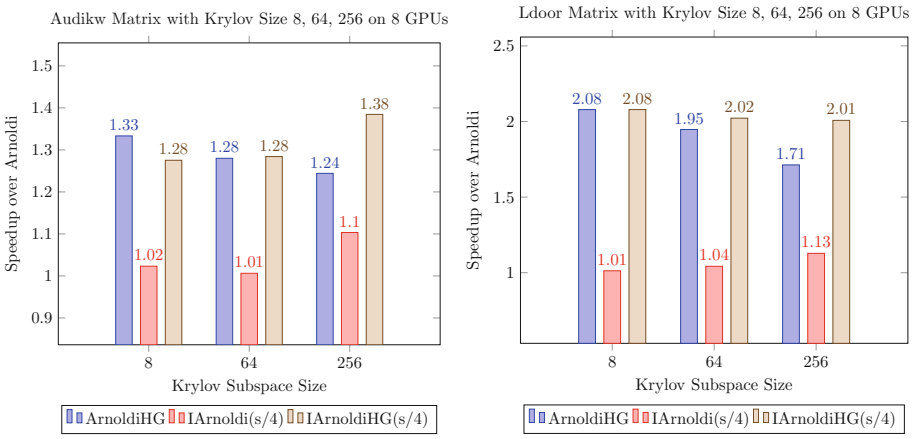
Fig. 7. Scalability Test on *HAPACS-TCA* with E-Diag matrix stored in CSR format double precision. (a) Strong scaling with total matrix size 9600000×9600000, diagonals of 33 and Krylov subspace size 64; (b) Weak scaling with each submatrix size 96000×96000, diagonals of 33 and Krylov subspace size 64



(a) Audikw **(b)** Ldoor

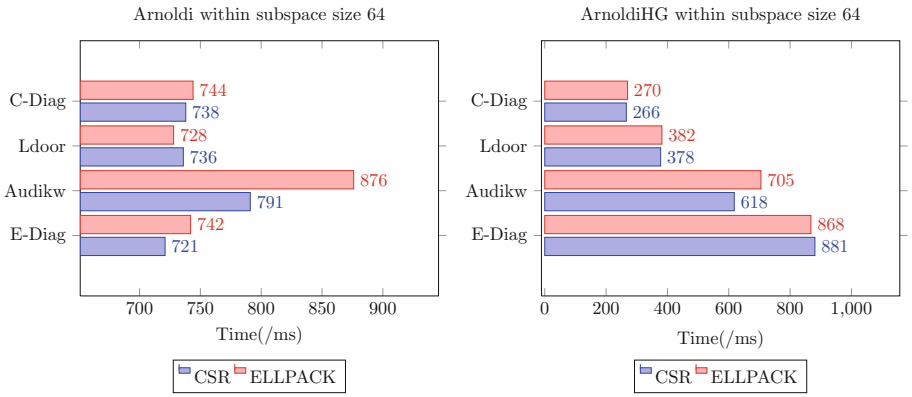
Fig. 8. Strong Scalability Test on *Poincare* with Audikw and Ldoor stored in CSR format double precision. (a) Audikw matrix size of 943695, nonzeros of 77651847, and Krylov subspace size 64; (b) Ldoor matrix size of 952203 nonzeros of 42943817, and Krylov subspace size 64

to around 10 to 30. In Fig. 9, the x-axis denotes three testing Krylov subspace sizes, and the y axis is the time speedup of *ArnoldiHG*, *IArnoldi*, *IArnoldiHG* over the original *Arnoldi*. Here, we set the q value of the truncation to be $s/4$ with s equals the Krylov subspace size. We find that our *ArnoldiHG* performs better in small subspace size rather than large subspace size. It is due to an increase of computation/communication intensity when the value s augments ($O(3s^2N)$). Thus, the algorithm is computation-bounded rather than



(a) Audikw **(b)** Ldoor

Fig. 9. Krylov Subspace Size Test on *Poincare* with Audikw and Ldoor stored in CSR format double precision. (a) Audikw matrix size of 943695, nonzeros of 77651847; (b) Ldoor matrix size of 952203 nonzeros of 42943817



(a) Arnoldi **(b)** ArnoldiHG

Fig. 10. Comparison of format CSR and ELLPACK on *Poincare* with 4 matrices with subspace size 64 running on 8 GPUs with double precision

communication-bounded, and the hypergraph optimization shall have a not significant performance impact. In terms of *IArnoldi*, the truncation of the computational workload has better performance impact when the subspace size s is larger, which leads to a better speedup over *Arnoldi*. Finally for *IArnoldiHG*, the two effects coexist, and the influence of subspace size varies according to many factors like the structure of the matrix. In Fig. 9, the test on *Audikw* shows a better speedup of *IArnoldiHG* within larger s values. While the *Ldoor* matrix shows a worse speedup of *IArnoldiHG* within larger s values.

4.3 Impact of Sparse Matrix Format

In [8], 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 our implementation, both of the CSR and ELLPACK format use the vectorization version which uses half a warp of 16 threads for each row of the matrix. Because the data storage of CSR is more contiguous than that of ELLPACK (padding zeros for some rows), the retrieval of data has a better efficiency. In Fig. 10, the implementation of format CSR runs slightly better than that of ELLPACK. In matrix *Audikw*, the gap is more evident because *Audikw* has a more irregular structure which causes much more padding of zeros for rows.

5 Conclusion

In this paper, we presented a hypergraph model to optimize the communication cost in computing the Krylov subspace basis from sparse matrices. The Classical Gram-Schmidt orthogonalization based Arnoldi method and its truncated version *Incomplete Arnoldi* have been chosen, and the optimization scheme have been applied to them. We study the time complexity and scalabilities of the four algorithmic implementation in a theoretical way and compare their features. According to the experiments on a CPU-GPU hybrid platform, we arrived on the following concluding remarks: (1) Our hypergraph based optimization is useful for communication-bounded kernels in Krylov subspace methods. (2) The nonzeros distribution pattern of sparse matrix has an influence on our optimization, where a matrix with more nonzeros aggregated around the main diagonal shall have more benefits from our scheme. (3) Factors like the subspace size and sparse matrix format also have affected the results of optimization, and an auto-tuning framework is expected to choose different parameters intelligently in the optimization. In order to further remove the communication bottleneck of the applications, we will consider the adoption of more aggressive strategies like Communication Avoiding method (e.g. TSQR, CAQR) [5] in future work.

Acknowledgments. The research presented in this work is partly supported by the ANR-JST Japanese-French project FP3C (Framework and Programming for Post-Petascale Computing). We thank Professor Tetsuya SAKURAI, Professor Taisuke BOKU and his team in University of Tsukuba for their support in providing the use of cluster HAPACS-TCA. We also thank the anonymous reviewers for their comments and recommendations, which help us continually improving the work.

References

1. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia (2003)
2. Arnoldi, W.E.: The principle of minimized iterations in the solution of the matrix eigenvalue problem. Q. Appl. Math. **9**, 17–29 (1951)

3. Saad, Y.: Numerical Methods for Large Eigenvalue Problems. Society for Industrial and Applied Mathematics, Philadelphia (2011)
4. Ghysels, P., Ashby, T.J., Meerbergen, K., Vanroose, W.: Hiding global communication latency in the GMRES algorithm on massively parallel machines. SIAM J. Sci. Comput. **35**, C48–C71 (2013)
5. Hoemmen, M.: A communication-avoiding, hybrid-parallel, rank-revealing orthogonalization method. In: 2011 IEEE International Parallel Distributed Processing Symposium (IPDPS), pp. 966–977 (2011)
6. Saad, Y., Wu, K.: DQGMRES: a direct quasi-minimal residual algorithm based on incomplete orthogonalization. Numer. Linear Algebra Appl. **3**, 3–329 (1996)
7. Catalyurek, U.V., Aykanat, C.: Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans. Parallel Distrib. Comput. **10**, 673–693 (1999)
8. Hugues, M., Petiton, S.: Sparse matrix formats evaluation and optimization on a GPU. In: 2010 12th IEEE International Conference on High Performance Computing and Communications (HPCC), pp. 122–129 (2010)