HPC Challenges and New Computing Frontiers

Serge G. Petiton
serge.petiton@univ-lille.fr

Workshop on HPC challenges for new extreme scale applications
Paris, March 6th, 2023

Preamble

I survey some results obtained for sparse linear algebra for iterative methods and for machine learning methods.

I also discuss on the potential evolution we would face to be able to mix computational science, data science and machine learning on future faster supercomputers, based on some published examples.

<u>Workshop</u>

End-users and scientists have to face a lot of challenge associated to these evolutions and the increasing size of the data. The convergence of Data science (big Data) and the computational science to develop new applications generates important challenges.

Outline

- Introduction
- New levels of programming (Graphs of Tasks, Network on chip)
- New methods and algorithms (Unite&Conquer, Stochastic Matrix,..)
- HPC and Machine Learning (GCN, Transformer,..)
- Generators of Data Sets and matrices for brain-scale applicatons
- What post-exascale plateforms and programin paradigms

Outline

- **Introduction**
- New levels of programming (Graphs of Tasks, Network on chip)
- New methods and algorithms (Unite&Conquer, Stochastic Matrix,..)
- HPC and Machine Learning (GCN, Transformer,..)
- Generators of Data Sets and matrices for brain-scale applicatons
- What post-exascale plateforms and programin paradigms

Computational science : IEEE 64 bit arihmetic,
exascale supercomputers, C++, CSR-ELLPACK

BigData : MapReduce, exascale 16-32 bits supercomputers, COO, Python,
Data Center, (HPC on) Cloud,

Machine Learning : Diferent arithmetics (16, 32, not always IEEE), Tensors, Python

- Mainly developped by GAFAM and BATX, Nvidia and others

Exascale supercomputers are now availables

- Frontiers,…
- Sunway,…
- Cerebras,…

Nevertheless, the target applications used different arithmetics and programming paradigms, and **only a few aplications reach the exascale (HPCG : 16 Pflops, Fugaku)**

Machine learning and AI applications are now requiring exascale machines, which were not first designed for them. New machines and processors (and the next generation of. post-exascale machines) are (would) be targeting "mainly" these applications.

The requiered arithmetic, data structures, linear algebra are often diferents.

The most expensive (time, energy) are the data migrations and communications, especially the I/O : Distributed and Parallel computing where are the data (HPC on Cloud or on DataCenter), or **generation of the data in Parallel**.

Back to

- "true" data parallelism (history : Connection Machines): Cerebras
- data flow programing (history : the Arwind MIT data flow machine): SambaNova

We have to experiments on "new" methods and propose the generation of "brain-scale" data sets (graphs-matrices) for computational science and machine learning applications.

Outline

- Introduction
- **New levels of programming (network on chip, graph of tasks)**
- New methods and algorithms (Unite&Conquer, Stochastic Matrix,..)
- HPC and Machine Learning (GCN, Transformer,..)
- Generators of Data Sets and matrices for brain-scale applicatons
- What post-exascale plateforms and programin paradigms

1 - Network on Chip

and irregular “local” communications

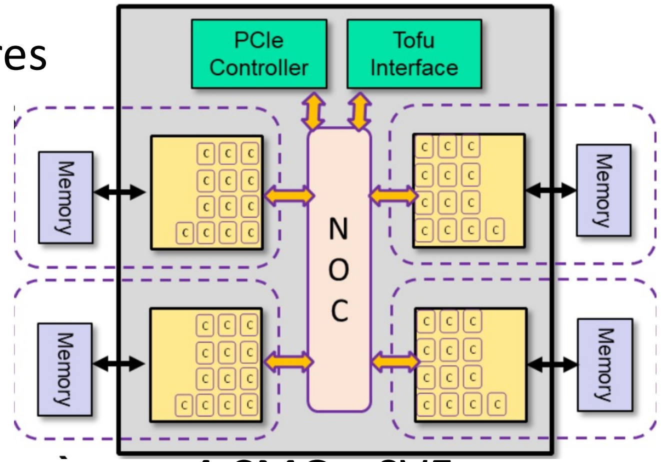
Larger number of nodes
(task programming)

Network on chip :
Distributed and data parallelism

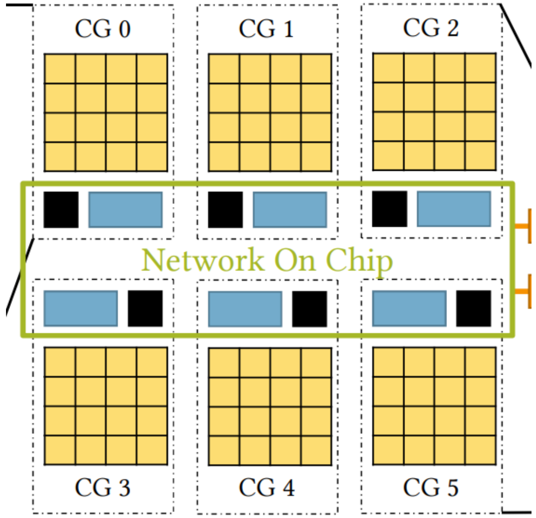
High hiearachy of execution models, which lead to several programming paradims for a given method : graph of tasks, PGAS, data parallel

Experiments :
- 1 MPI “task” per chip (1 openMP per chip + SVE)
- 4 MPI “tasks” per chip (1 openMP per CMG + SVE)

4 x 12 cores



4 CMGs, SVE

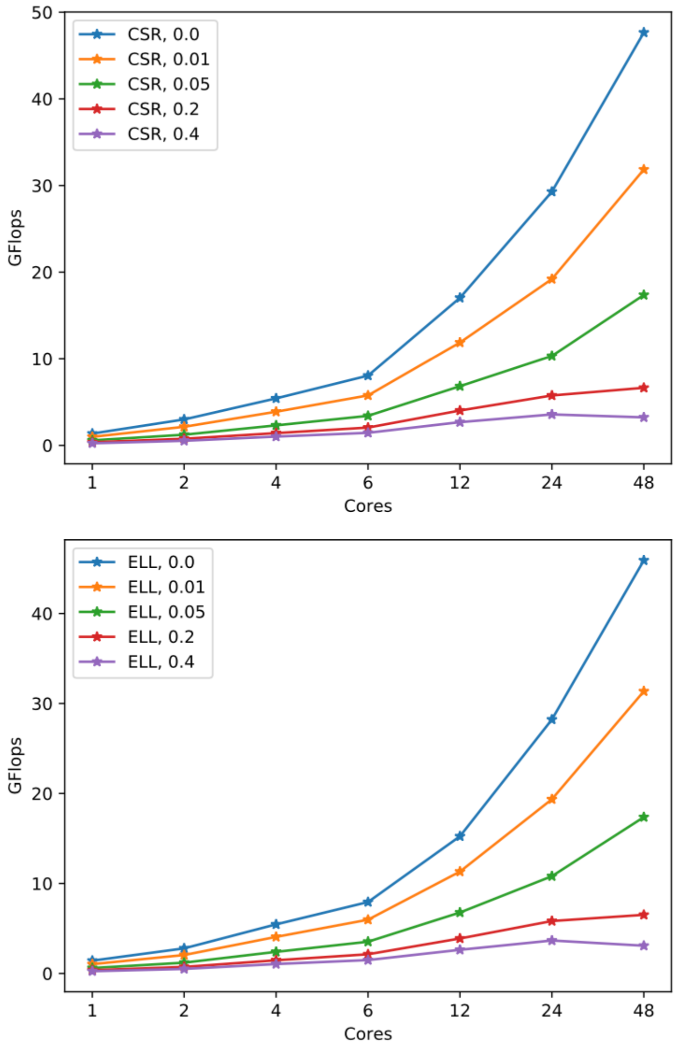


Sunway

2021 IEEE International Conference on Cluster Computing (CLUSTER)

Sequences of Sparse Matrix-Vector Multiplication on Fugaku’s A64FX processors

Jérôme Gurhem*, Maxence Vandromme*, Miwako Tsuji†, Serge G. Petiton*‡, Mitsuhisa Sato†



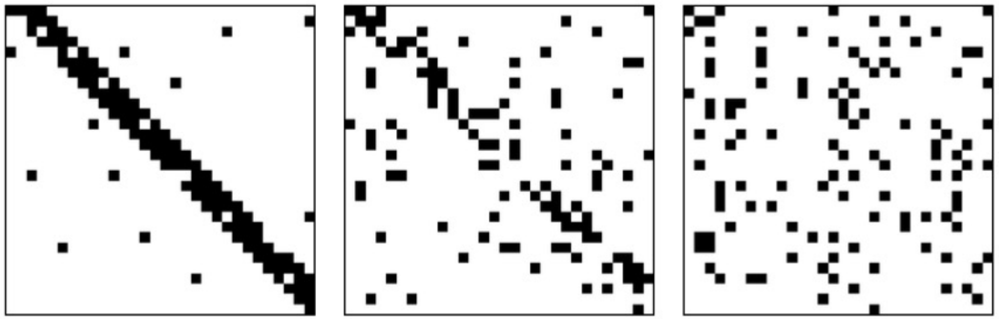


Fig. 1. C-diagonal Q-perturbed sparse matrix with C=4 and Q=0.05 on the left, Q=0.5 on the middle and Q=0.9 on the right

Fig. 8. $A(Ax + x) + x$ with OpenMP for C-diagonal Q-perturbed matrices with C = 16

2021 IEEE International Conference on Cluster Computing (CLUSTER)

PageRank on Fugaku

threads	0.0		0.4		0.8	
	1	2	1	2	1	2
1	32.5	35.7	9.4	15.3	6.2	10.7
2	32.2	36.3	9.4	15.3	6.2	10.8
4	35.8	55.4	9.7	18.2	6.6	12.2
6	42.5	55.0	10.2	18.2	7.0	12.3
12	51.7	64.2	11.2	19.8	8.7	13.6
24	72.4	81.3	8.1	14.4	7.7	13.2
48	50.3	103.9	2.7	9.8	1.3	5.2
MPI	32.5	33.6	9.6	14.5	6.2	10.4

TABLE VIII

MPI AND MPI+OPENMP PERFORMANCE (IN GFLOP/S) FOR $A(Ax+x)+x$ WITH C-DIAGONAL Q-PERTURBED MATRICES ON 1 AND 2 NODES FOR CSR STORAGE FORMAT WITH DIFFERENT NUMBER OF THREADS PER MPI PROCESS, KEEPING 48 THREADS PER NODE

Sequences of Sparse Matrix-Vector Multiplication on Fugaku's A64FX processors

Jérôme Gurhem*, Maxence Vandromme*, Miwako Tsuji†, Serge G. Petiton*‡, Mitsuhisa Sato†

N = 4 000 000, NNZ = 50, or 100 per node

1 MPI_OpenMP/node better if the Matrices are not ”too” iregular

threads	0.0		0.4		0.8	
	1	2	1	2	1	2
1	32.5	35.2	9.5	15.3	6.2	10.7
2	31.9	36.8	9.5	15.6	6.2	10.9
4	35.4	53.8	9.9	18.5	6.5	12.2
6	41.5	55.3	10.3	18.6	6.7	12.4
12	51.3	59.3	11.4	20.3	8.7	13.0
24	71.8	83.3	8.2	14.7	7.2	10.5
48	50.3	106.1	2.9	10.3	1.3	5.6
MPI	32.6	33.1	9.6	14.8	6.3	10.4

TABLE IX

MPI AND MPI+OPENMP PERFORMANCE (IN GFLOP/S) FOR $A(Ax+x)+x$ WITH C-DIAGONAL Q-PERTURBED MATRICES ON 1 AND 2 NODES FOR ELL STORAGE FORMAT WITH DIFFERENT NUMBER OF THREADS PER MPI PROCESS, KEEPING 48 THREADS PER NODE

Otherwise, 4 MPI-OpenMP per node are more efficient.

nodes	CSR		ELL		SCOO	
	node	CMG	node	CMG	node	CMG
1	1.89	0.91	1.30	0.91	5.19	2.17
2	2.17	0.76	1.41	0.79	4.24	2.01
4	1.98	0.69	1.33	0.71	3.28	1.84
8	1.58	0.54	1.02	0.55	2.57	1.47
16	1.39	0.47	0.98	0.48	2.24	1.28
32	1.39	0.46	0.88	0.54	2.25	1.33
64	1.40	0.46	0.93	0.47	2.24	1.32
128	1.20	0.43	1.22	0.40	1.89	1.10
256	1.00	0.36	1.02	0.35	1.56	0.95
512	1.00	0.35	1.00	0.35	1.13	0.84
1024	1.00	0.37	1.01	0.37	1.16	0.86

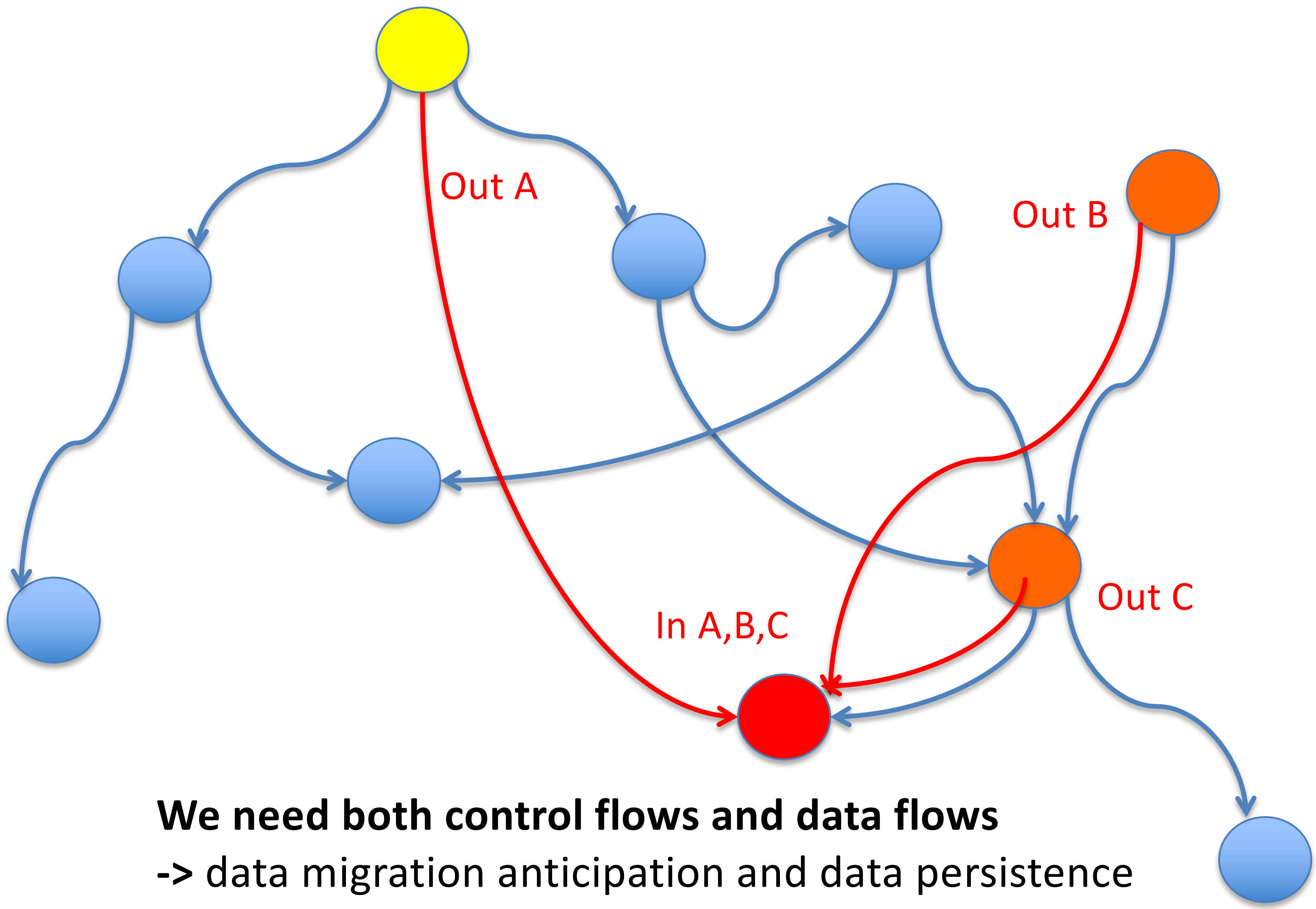
Table 1: Median runtime for the PageRank, scaling the number of nonzero elements per row with the number of compute nodes, from a base of $nnz = 50$

nodes	CSR		ELL		SCOO	
	node	CMG	node	CMG	node	CMG
1	5.90	1.28	2.59	1.30	5.59	3.53
2	3.92	1.15	2.46	1.19	4.55	3.24
4	3.14	0.90	1.89	0.92	4.37	2.59
8	2.75	0.78	1.70	0.79	3.77	2.27
16	2.78	0.77	1.69	0.81	3.83	2.27
32	2.77	0.77	1.83	0.81	3.81	2.31
64	2.38	0.67	2.20	0.70	3.26	1.99
128	1.98	0.56	2.00	0.58	2.68	1.63
256	1.99	0.56	2.00	0.56	2.68	1.64
512	1.96	0.56	1.96	0.57	2.26	1.55
1024	1.98	0.59	1.97	0.59	2.28	1.58

Table 2: Median runtime for the PageRank, scaling the number of nonzero elements per row with the number of compute nodes, from a base of $nnz = 100$

2 - Graph of Tasks programming

Other new level on programming : graph of task programming



We need both control flows and data flows

-> data migration anticipation and data persistence

YML (since 2000) - yml.prism.uvsq.fr
(opensource, Cecil Licence)

We have a Virtual Machine with tutorials

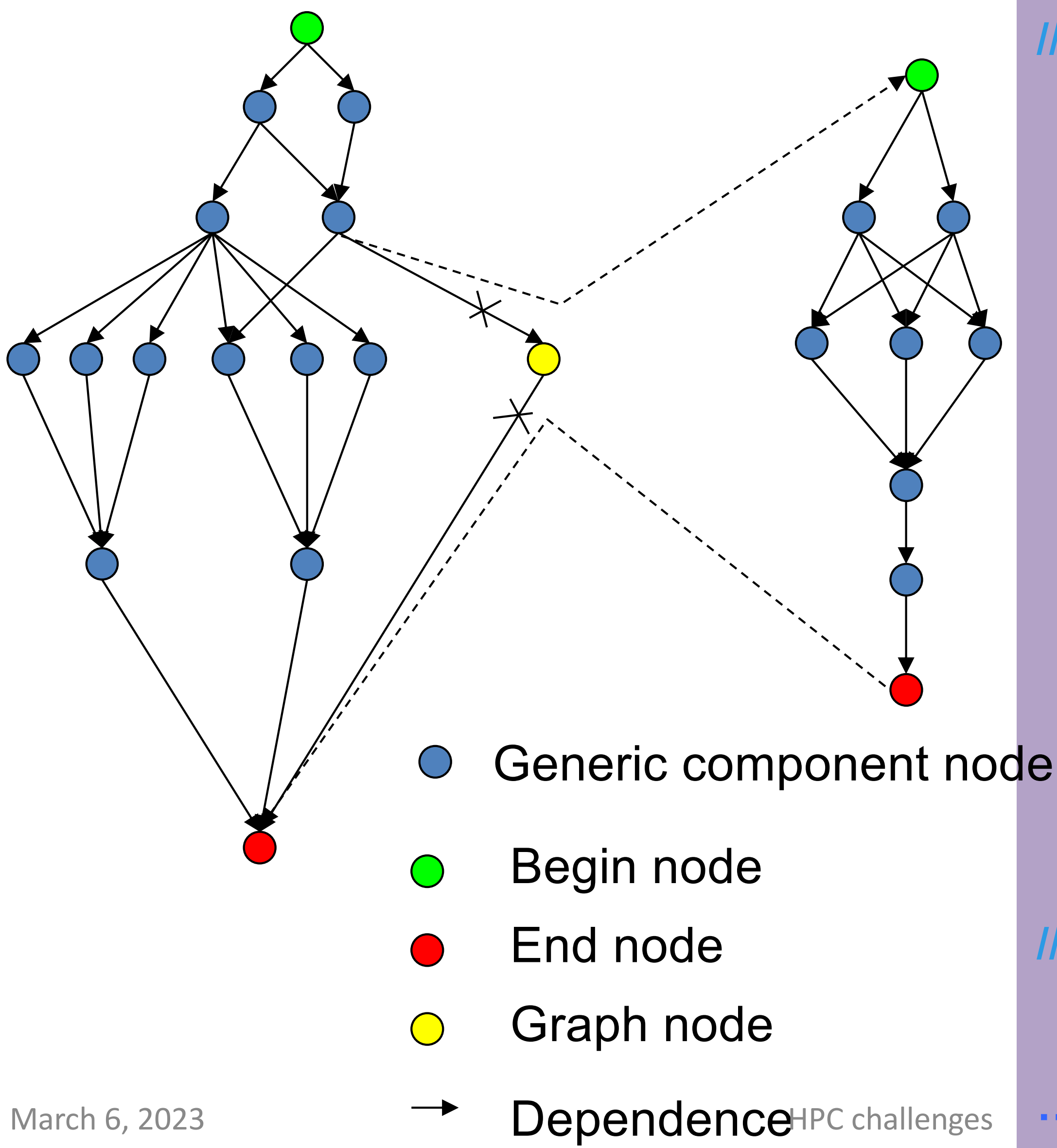
yml.prism.uvsq.fr

Main properties :

- High level graph description language (*coordination/control language) – LL(1) grammar*
- Independent of Middleware, hardware and libraries
- A backend for each systems or middleware (*then platforms or supercomputers/hypercomputers*) : Xtremweb(P2P), OmniRPC, Xtremweb-OmniRPC
- Expertise may be proposed by end-users
- May use existing components / thought eventualy libraries

Deployed in France, Belgium, Ireland, Japan (K, T2K-Tsukuba, FX10-AICS)
China (Hohai, Najing), Tunisia, USA (Hooper-LBNL, TOTAL-Houston).
Experiment on P2P or GRID platforms : Grid (Gird5000) and P2P (100 PCs in Lille, 100 PC in France, and 4 clusters in Japan, launch from a SC INRIA booth

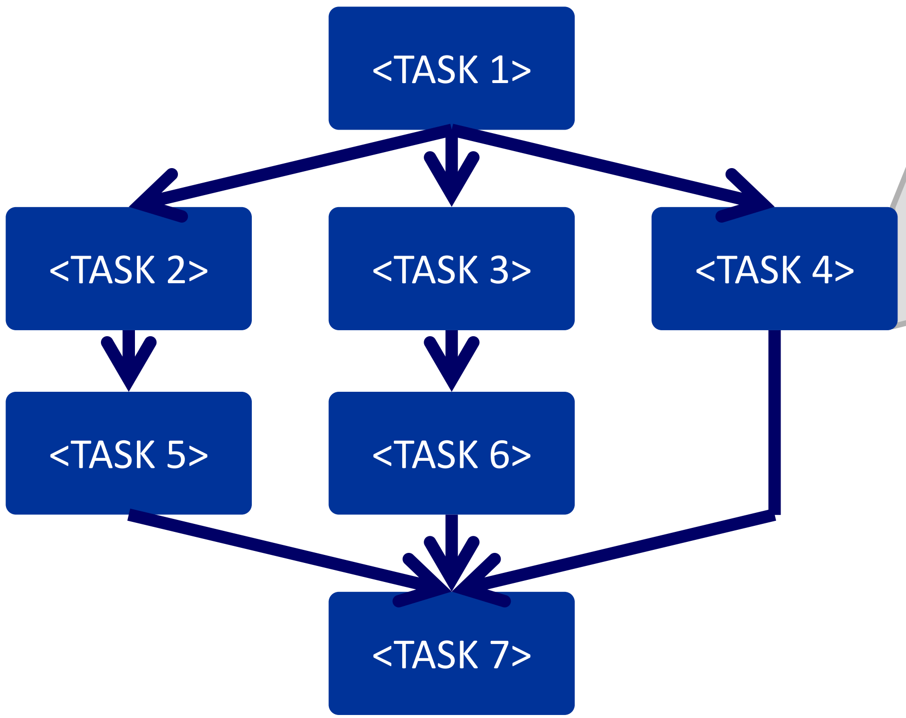
Graph (n dimensions) of components/tasksYML



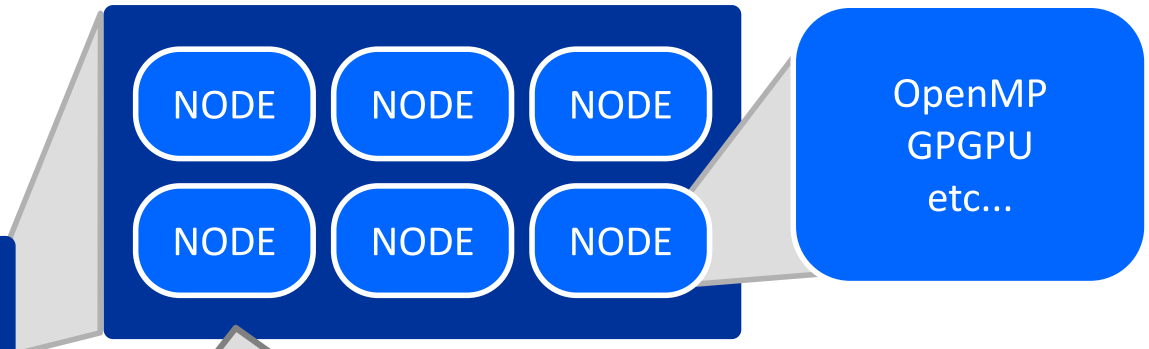

```
par
  compute tache1(..);
  notify(e1);
//
  compute tache2(..); migrate matrix(..);
  notify(e2);
//
  wait(e1 and e2);
  Par (i :=1;n) do
    par
        compute tache3(..);
    notify(e3(i));
    //
    if(I < n)then
      wait(e3(i+1));
      compute tache4(..);
      notify(e4);
     endif;
    //
    compute tache5(..); control robot(..);
    notify(e5); visualize mesh(…) ;
    end par
  end do par
//
  wait(e3(2:n) and e4 and e5);
  compute tache6(..);
…/…
end par
```

Multi-Level Parallelism Integration: YML-XMP

N dimension graphs available



YML provides a workflow programming environment and high level graph description language called YvetteML

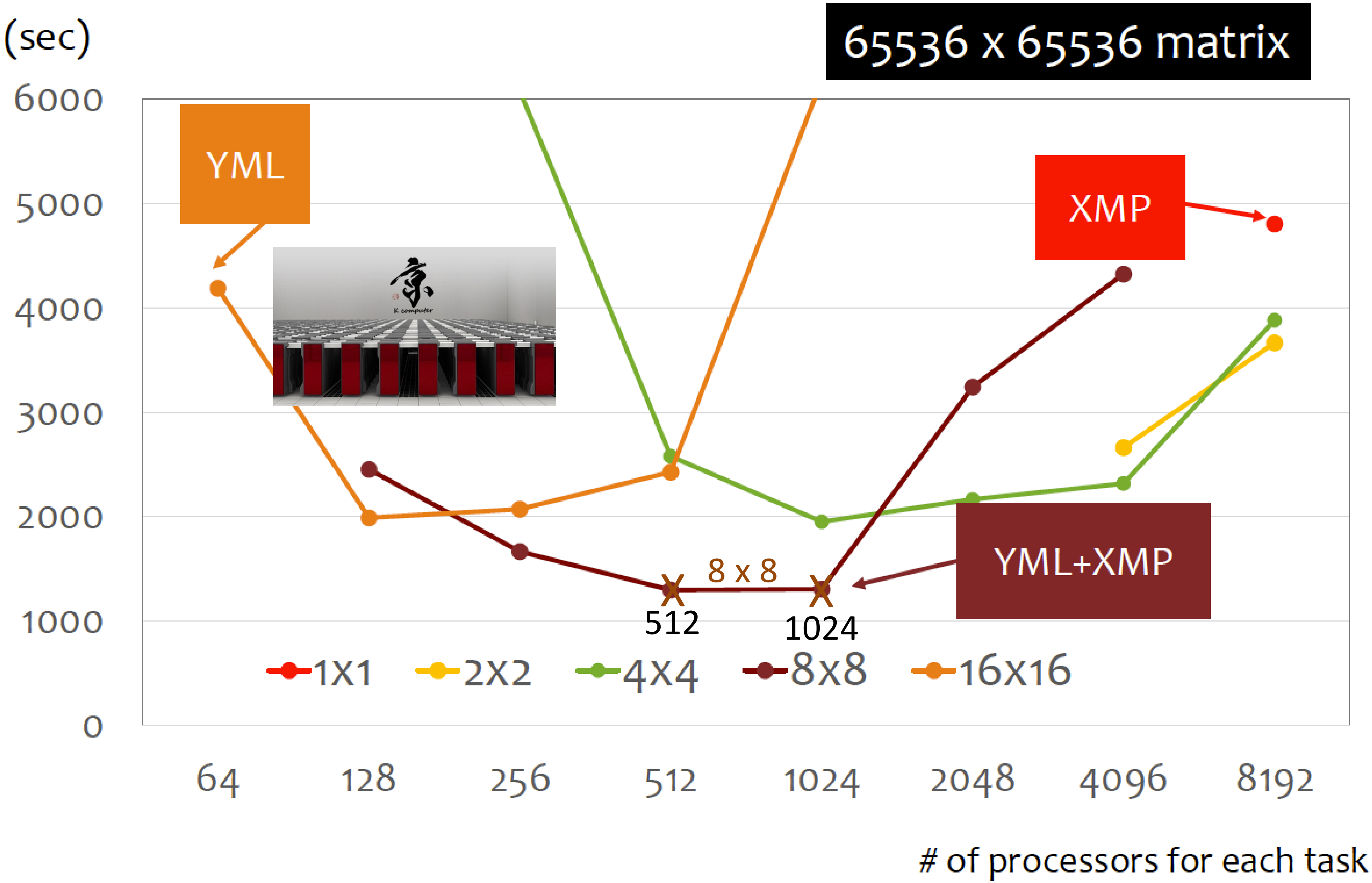



```
for(i=0;i<n;i++){
    for(j=0;j<n;j++){
        tmp[i][j]=0.0;
#pragma xmp loop (k) on t(k)
          for(k=0;k<n;k++){
              tmp[i][j]+=(m1[i][k]*m2[k][j]);
}}}
#pragma xmp reduction (+:tmp)
```

Each task is a parallel program over several nodes.
XMP language can be used to descript parallel program easily!

YML/XMP/StarPu expriments on T2K in Japan, French-Japanese project FP3C

Experiments (2) BGJ on K-Computer



Slide written by Miwako TSUJI, RIKEN/AICS

Outline

- Introduction
- New levels of programming (Graphs of Tasks, Network on chip)
- **New methods and algorithms (Unite&Conquer, Stochastic Matrix,..)**
- HPC and Machine Learning (GCN, Transformer,..)
- Generators of Data Sets and matrices for brain-scale applicatons
- What post-exascale plateforms and programin paradigms

1 - Minimizing the number of operations

Efficient Parallel PageRank Algorithm for Network Analysis

Maxence Vandromme*, Serge G. Petiton*
*Univ. Lille, UMR 9189 - CRIStAL, CNRS
Lille, France

ParSoc22, proceedings of IPDPS22

***Abstract*—We propose an efficient version of the PageRank algorithm for adjacency matrices, that reduces the complexity by a factor two. This method computes the $A^T x$ operation on the transpose matrix A^T without having to explicitly normalize and transpose the matrix. We implement the method using standard row-major and column-major matrix storage formats. We perform experiments with parallel implementations in OpenMP, on synthetic data as well as on matrices extracted from large-scale graphs. The experiments are done on two different Intel processors from recent generations. The column-major storage format version of our method shows good scaling and outperforms the standard PageRank in a majority of cases, even when not considering the preprocessing burden in the latter.**

Based on the optimisation of the numer of operations for stochastic matrix by a vector products

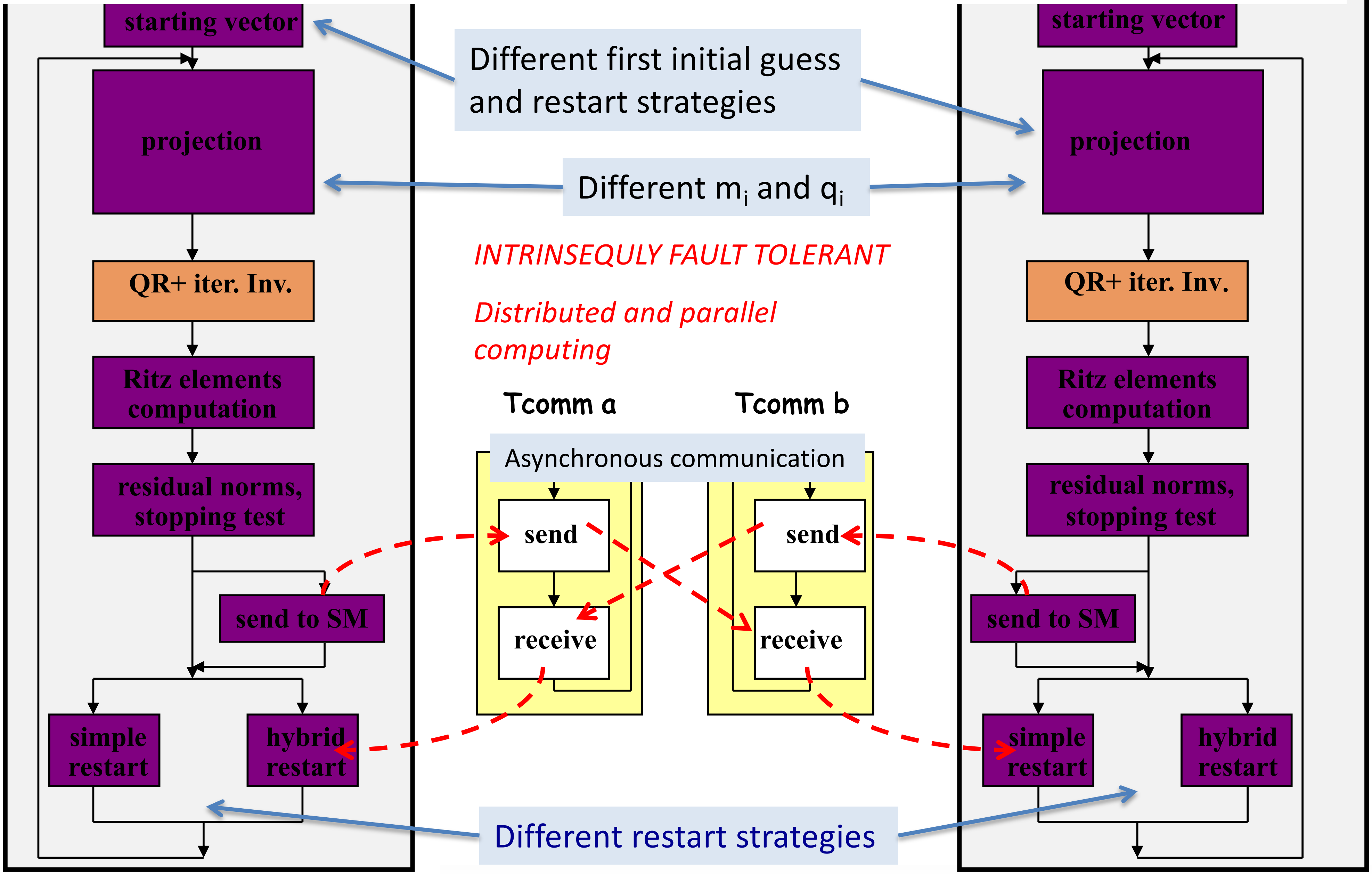
TABLE II: Median runtime (in ms) of all three applications on synth-3, for both storage formats and both variants of SpMV (o = original, n = new) on Ruche

	SpMV				$A(Ax+x)$				PageRank			
threads	CSR-o	CSR-n	CSC-o	CSC-n	CSR-o	CSR-n	CSC-o	CSC-n	CSR-o	CSR-n	CSC-o	CSC-n
1	259.9	**197.1**	302.6	200.8	533.8	405.5	607.8	**398.3**	1611	1253	1788	**1186**
2	131.1	112.6	151.3	**100.5**	264.3	225.3	310.8	**199.7**	807.7	673.6	905.8	**594.7**
4	67.2	56.3	77.9	**50.6**	135.2	113.1	158.0	**100.1**	413.7	339.2	461.4	**298.8**
8	35.1	28.5	41.3	**25.5**	70.1	57.4	81.2	**50.6**	215.7	172.4	266.8	**151.3**
12	24.2	19.4	28.5	**17.2**	48.4	39.1	56.0	**34.1**	149.3	117.2	172.5	**102.3**
16	18.7	14.8	22.3	**13.0**	37.6	29.7	44.0	**25.8**	116.1	90.7	133.3	**77.7**
20	15.5	12.1	19.1	**10.5**	31.1	24.6	37.9	**20.9**	96.2	74.6	115.4	**63.2**
30	14.7	10.2	17.1	**7.9**	29.7	21.2	34.4	**15.5**	92.1	65.8	102.7	**46.8**
40	17.2	9.5	19.5	**6.6**	34.8	22.1	39.0	**13.0**	108.1	67.7	119.3	**39.8**

TABLE III: Median runtime (in ms) of all three applications on synth-3, for both storage formats and both variants of SpMV (o = original, n = new) on Ice Lake

	SpMV				$A(Ax+x)$				PageRank			
threads	CSR-o	CSR-n	CSC-o	CSC-n	CSR-o	CSR-n	CSC-o	CSC-n	CSR-o	CSR-n	CSC-o	CSC-n
1	138.2	**109.9**	142.9	119.7	278.3	**220.9**	284.4	239.7	857.4	**674.5**	900.0	718.7
2	75.2	**57.1**	79.5	60.6	152.3	**114.6**	158.1	121.7	484.6	**339.5**	483.1	366.2
4	38.2	**28.3**	40.4	30.4	80.3	**57.8**	81.6	61.4	254.2	**168.3**	260.6	186.6
8	19.7	**14.3**	23.2	15.2	40.1	**30.1**	41.9	31.1	126.1	**90.0**	133.4	92.1
12	13.4	**9.9**	18.0	10.2	28.1	20.9	32.7	**20.8**	84.9	**60.8**	106.9	62.7
20	11.4	6.7	11.2	**6.2**	18.6	13.8	21.3	**12.6**	57.2	41.7	66.1	**38.9**
28	8.5	5.7	9.1	**4.7**	15.8	11.5	18.7	**9.4**	47.2	34.2	53.9	**28.9**
38	7.5	5.6	9.2	**3.6**	14.6	10.9	18.4	**7.2**	42.0	33.2	47.3	**22.9**
57	7.4	6.1	9.7	**2.7**	15.2	12.6	19.7	**5.5**	51.9	37.8	53.5	**17.7**
76	10.8	6.7	16.1	**3.2**	21.9	14.3	25.1	**6.6**	41.8	46.7	65.7	**22.2**

2 - Unite and Conquer (asynchronous computation to minimize the number of iterations)

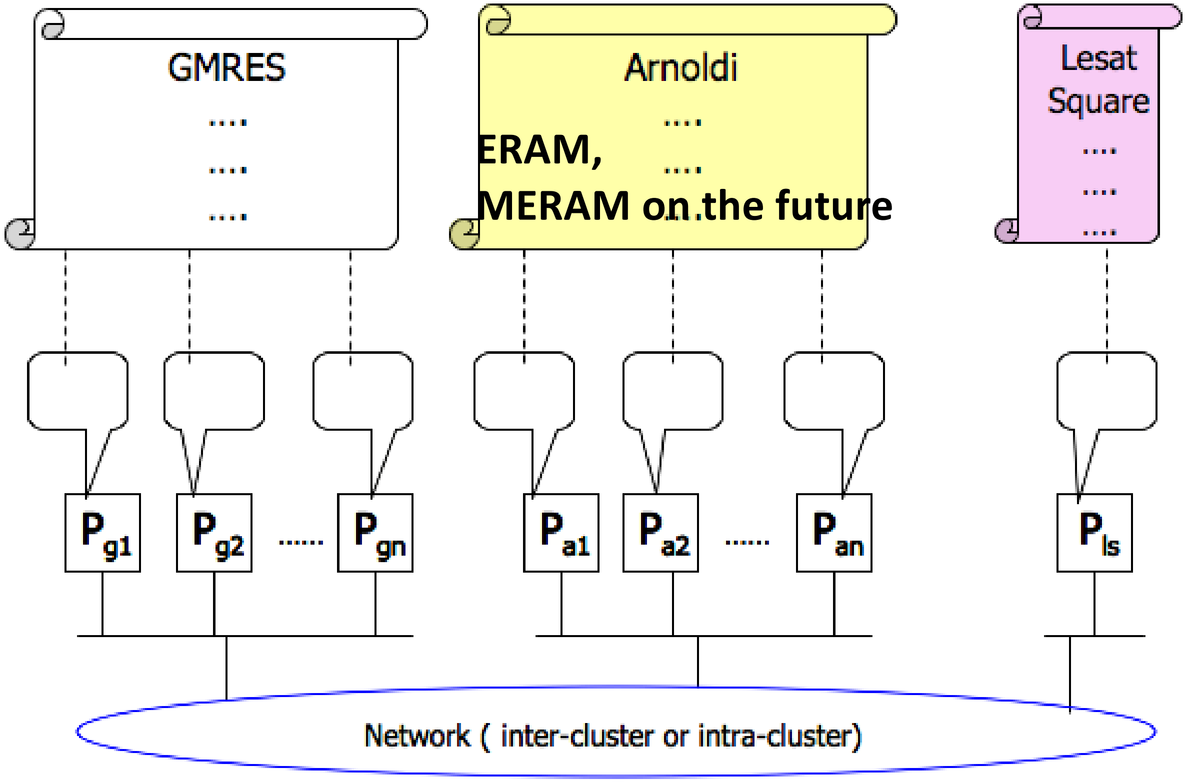


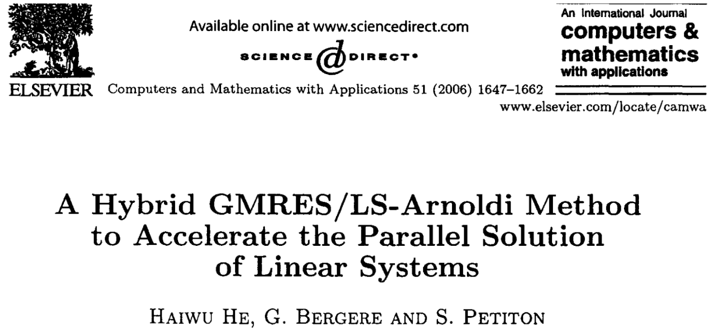
SIAM Journal on Scientific Computing → Vol. 27, Iss. 1 (2005) → 10.1137/S1064827500366082

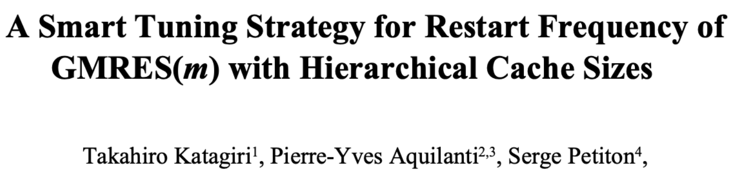
Multiple Explicitly Restarted Arnoldi Method for Solving Large Eigenproblems **Nahid Emad, Serge Petiton,** and **Guy Edjlali**

Asynchronous Iterative Restarted Methods

Collaboration with He Haiwu and Guy Bergère (U. Lille 1, CNRS)
and Ye Zhang (Hohai Univ. Nanjing) , Salim Nahi (Maison de la simulation),
and Pierre-Yves Aquilenti (TOTAL)







- Haiwu He, Guy Bergère, and Serge Petiton, Computational Math. Appl., 2006
- Ye Zhang, Guy Bergère, and Serge Petiton, LNCS, Springer Verlag, 2008
- .../...

Outline

- Introduction
- New levels of programming (Graphs of Tasks, Network on chip)
- New methods and algorithms (Unite&Conquer, Stochastic Matrix,..)
- **HPC and Machine Learning (GCN, Transformer,..)**
- Generators of Data Sets and matrices for brain-scale applicatons
- What post-exascale plateforms and programin paradigms

Parallel Jaccard and Related Graph Clustering Techniques

Alexandre Fender
Nvidia Corp., Maison de la Simulation
LI-PaRAD - University of Paris-Saclay
afender@nvidia.com

Nahid Emad
Maison de la Simulation
LI-PaRAD - University of Paris-Saclay
nahid.emad@uvsq.fr

Serge Petiton
Maison de la Simulation
University of Lille I, Sci. & Tech.
serge.petiton@univ-lille1.fr

Joe Eaton
Nvidia Corporation
featon@nvidia.com

Maxim Naumov
Nvidia Corporation
mnaumov@nvidia.com

ScalA17, SC17

6 GRAPH CLUSTERING

In graph clustering a vertex set V is often partitioned into p disjoint sets S_k, such that $V = S_1 \cup S_2 \ldots \cup S_p$ and $S_i \cap S_j = \{\emptyset\}$ for $i \neq j$ [16, 21]. Notice that instead of the original graph $G = (V, E)$ we can use the modified graph $G^{(*)} = (V^{(*)}, E^{(*)})$, with vertex $v_i^{(*)}$ and edge $w_{ij}^{(*)}$ weights computed based on PageRank and Jaccard or related schemes discussed in earlier sections.

6.1 Jaccard Spectral Clustering

Notice that we can define the Laplacian as

$$L^{(*)} = D^{(*)} - A^{(*)} \tag{32}$$

where $D^{(*)} = \text{diag}(A^{(*)}\mathbf{e})$ is the diagonal matrix.

Then, we would minimize the normalized balanced cut

$$\begin{aligned} \tilde{\eta}(S_1, \ldots, S_p) &= \min_{S_1,\ldots,S_p} \sum_{k=1}^{p} \frac{\text{vol}(\partial(S_k))}{\text{vol}(S_k)} \\ &= \min_{U^T D^{(*)} U = I} \text{Tr}(U^T L^{(*)} U) \end{aligned} \tag{33}$$

where Tr(.) is the trace of a matrix, boundary edges

$$\partial S = \{(i,j) \mid i \in S \wedge j \notin S\} \tag{34}$$

and volume

$$\text{vol}(S) = \sum_{i \in S} w_{ij}^{(*)} \tag{35}$$

$$\text{vol}(\partial S) = \sum_{(i,j) \in \partial(S)} w_{ij}^{(*)} = \sum_{(i,j) \in \partial(S)} w_{ij}^{(O)} \left(1 + \frac{w_{ij}^{(I)}}{w_{ij}^{(U)}}\right)$$

by finding its smallest eigenpairs and transforming them into assignment of nodes into clusters [22]. Notice that Jaccard weights correspond to the last term in the above formula, and are related to the sum of ratios of the intersection and union of nodes on the boundary of clusters.

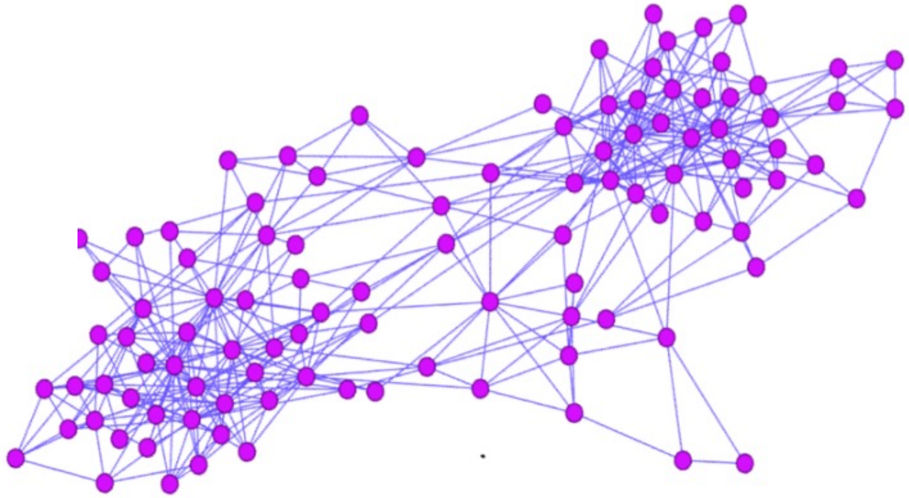


Figure 1: Amazon book co-purchasing original graph

Figure 2: Amazon book co-purchasing graph with Jaccard

Graph Convolutional Network (GCN)

$$H^{(\ell+1)} = \sigma\left(\tilde{A} H^{(\ell)} W^{(\ell)}\right); \ell = 0, L$$

2022 IEEE International Conference on Big Data (Big Data)

Enhancing Graph Convolutional Networks by Topology Sampling

Quentin R. Petit
Huawei Paris Research Center
& Université Paris-Saclay
Boulogne-Billancourt, France
quentin.petit2@huawei.com

Chong Li
Distributed and Parallel Software Lab
Huawei Paris Research Center
Boulogne-Billancourt, France
ch.l@huawei.com

Serge G. Petiton
Université de Lille
Lille, France
serge.petiton@univ-lille.fr

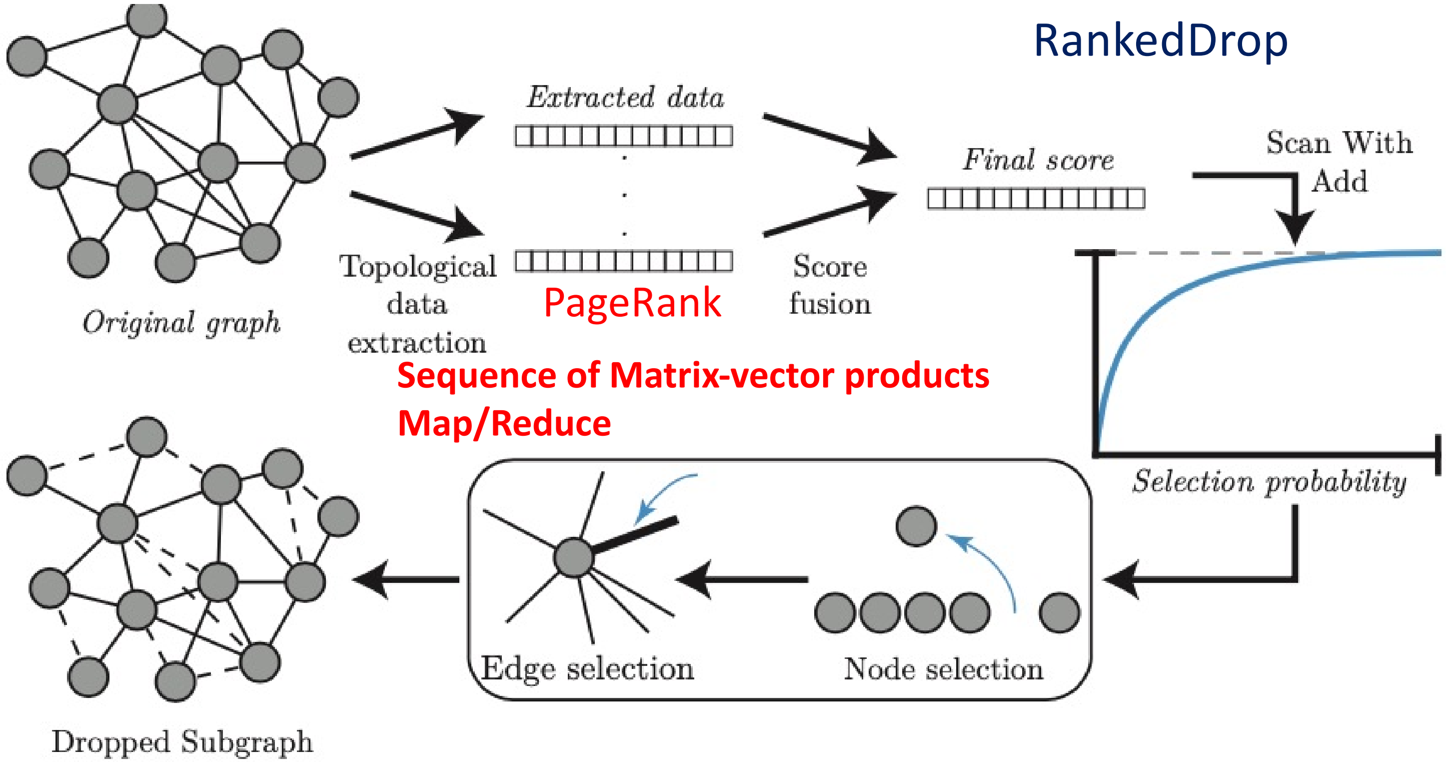
Kelun Chai
Huawei Paris Research Center
& Université Paris-Saclay
Boulogne-Billancourt, France
kelunchai@gmail.com

Nahid Emad
Maison de la Simulation & LI-PaRAD
Université Versailles Saint-Quentin
Saclay, France
nahid.emad@uvsq.fr

Normalized Laplacian (Sparse non symetrical) Matrices (computed from the adjacency matrix)

Sequence of sparse by dense by dense products

Edge and/or node droping to limit the over-smoothing

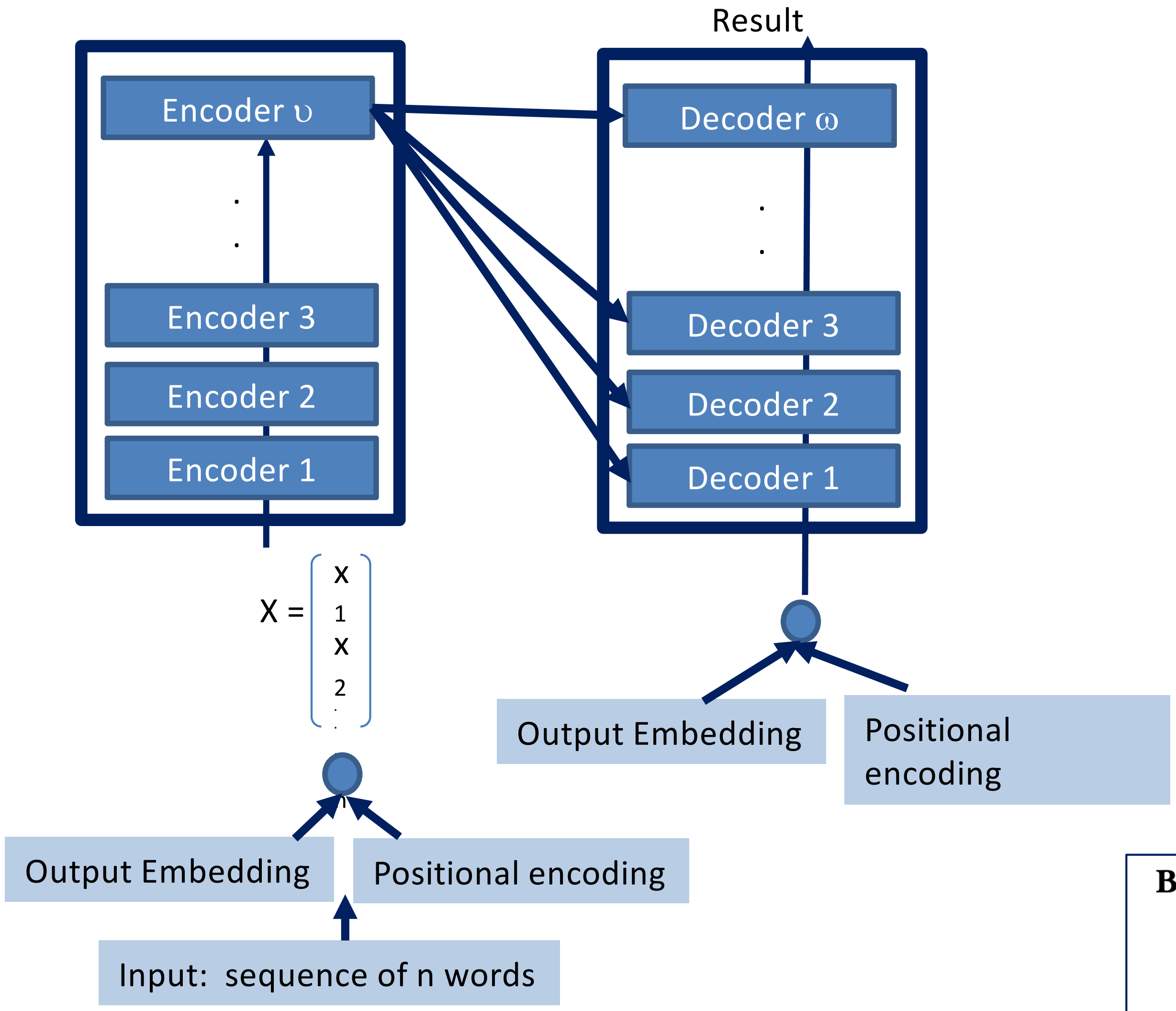


Benchmark, Data sets :

- Cora,
- Pubmed
- …

Small ones

Transformer method general structure



BERT, GPT,..

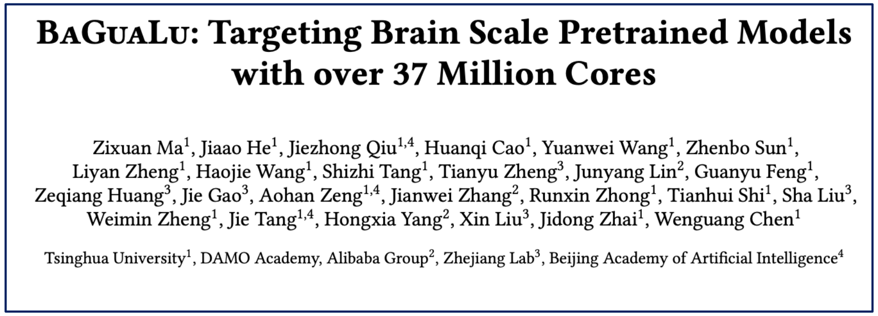
Attention :

$$\left(\sigma \frac{QK^T}{\sqrt{d_K}}\right) V$$

Matmul

Sequence of dense by dense rectangular matrix products

Brain scale experiment :
BaGuaLu (China, 1 exascale, mixed arithmetic, Sunway – processor,w ith a network on chip)
Proceedings of PPoPP 22

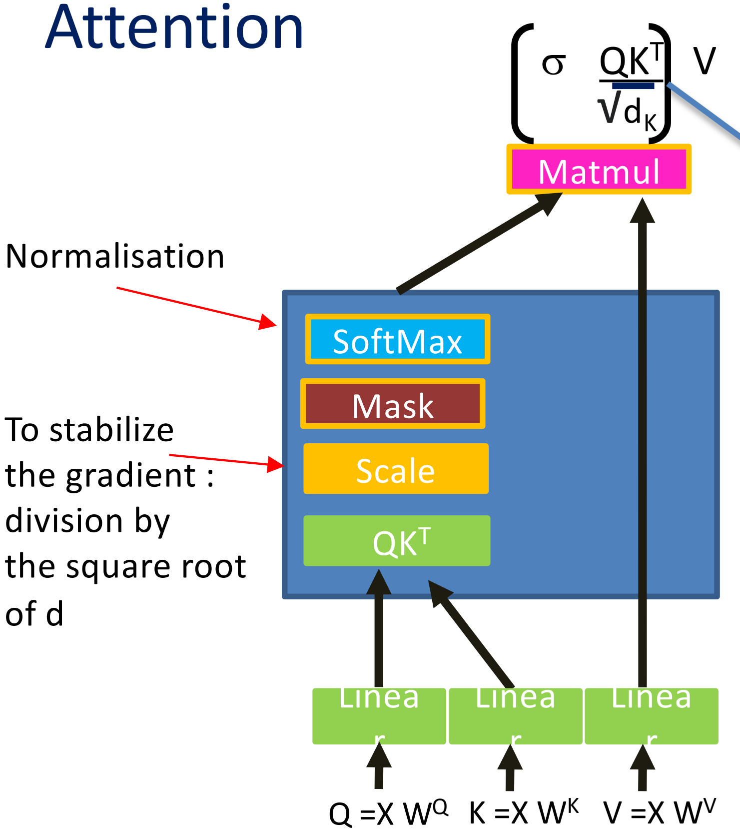


BaGuaLu: Targeting Brain Scale Pretrained Models with over 37 Million Cores

Zixuan Ma[1], Jiaao He[1], Jiezhong Qiu[1,4], Huanqi Cao[1], Yuanwei Wang[1], Zhenbo Sun[1], Liyan Zheng[1], Haojie Wang[1], Shizhi Tang[1], Tianyu Zheng[3], Junyang Lin[2], Guanyu Feng[1], Zeqiang Huang[3], Jie Gao[3], Aohan Zeng[1,4], Jianwei Zhang[2], Runxin Zhong[1], Tianhui Shi[1], Sha Liu[3], Weimin Zheng[1], Jie Tang[1,4], Hongxia Yang[2], Xin Liu[3], Jidong Zhai[1], Wenguang Chen[1]

Tsinghua University[1], DAMO Academy, Alibaba Group[2], Zhejiang Lab[3], Beijing Academy of Artificial Intelligence[4]

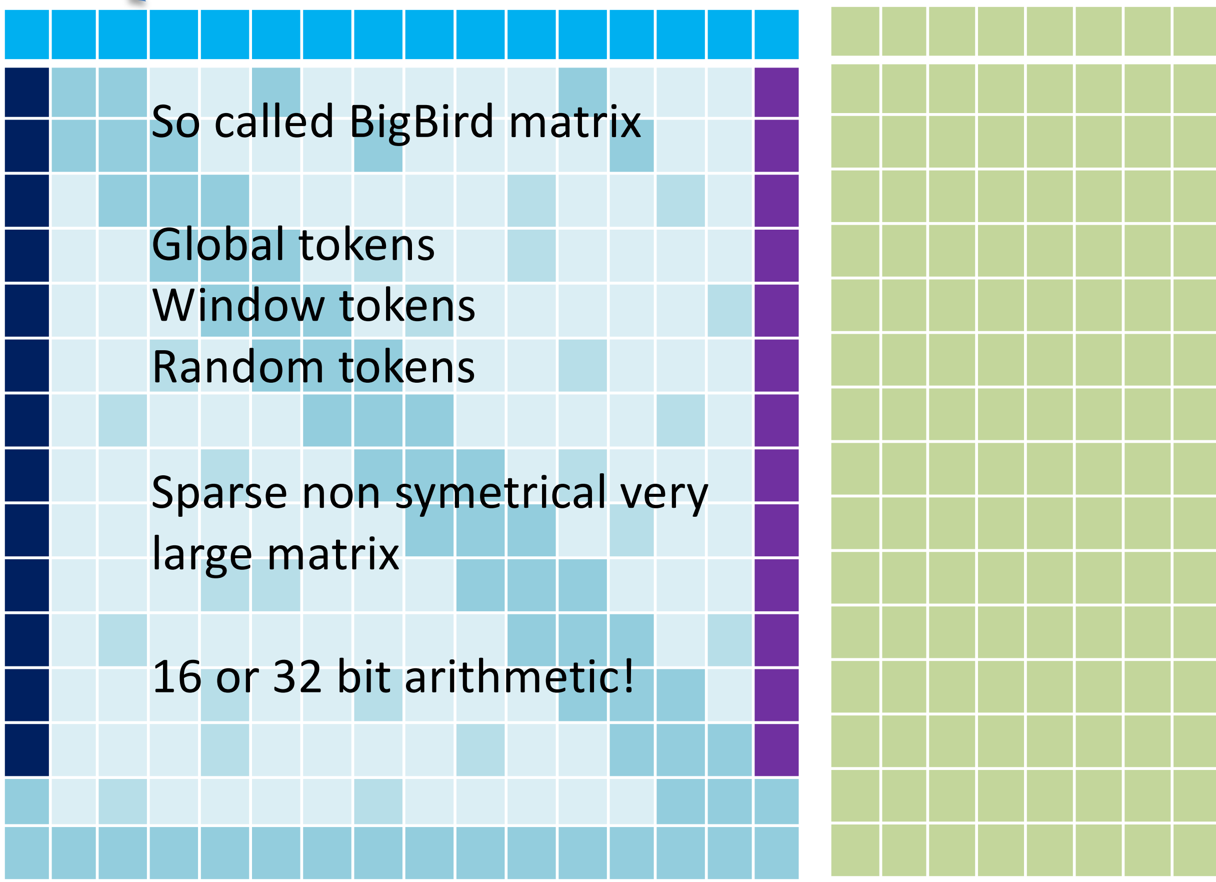
Attention



Dense by Dense rectangular matrix products (32-16 bits)

If the sequence is large ==> very large dense matrices

Then, we "sparsify" the matrices



The initial W are with randome values

Sequence of such computation

The choice of the random blocks are diferent for each "iteration" : dynamic structures

Each block is dense : depending of the hierarchy of the machine, we may use vectorial hardware at the lower level.


SWINBERT: End-to-End Transformers with Sparse Attention for Video Captioning

Kevin Lin*, Linjie Li*, Chung-Ching Lin*, Faisal Ahmed, Zhe Gan, Zicheng Liu, Yumao Lu, Lijuan Wang
Microsoft
{keli, lindsey.li, chungching.lin, fiahmed, zhe.gan, zliu, yumaolu, lijuanw}@microsoft.com

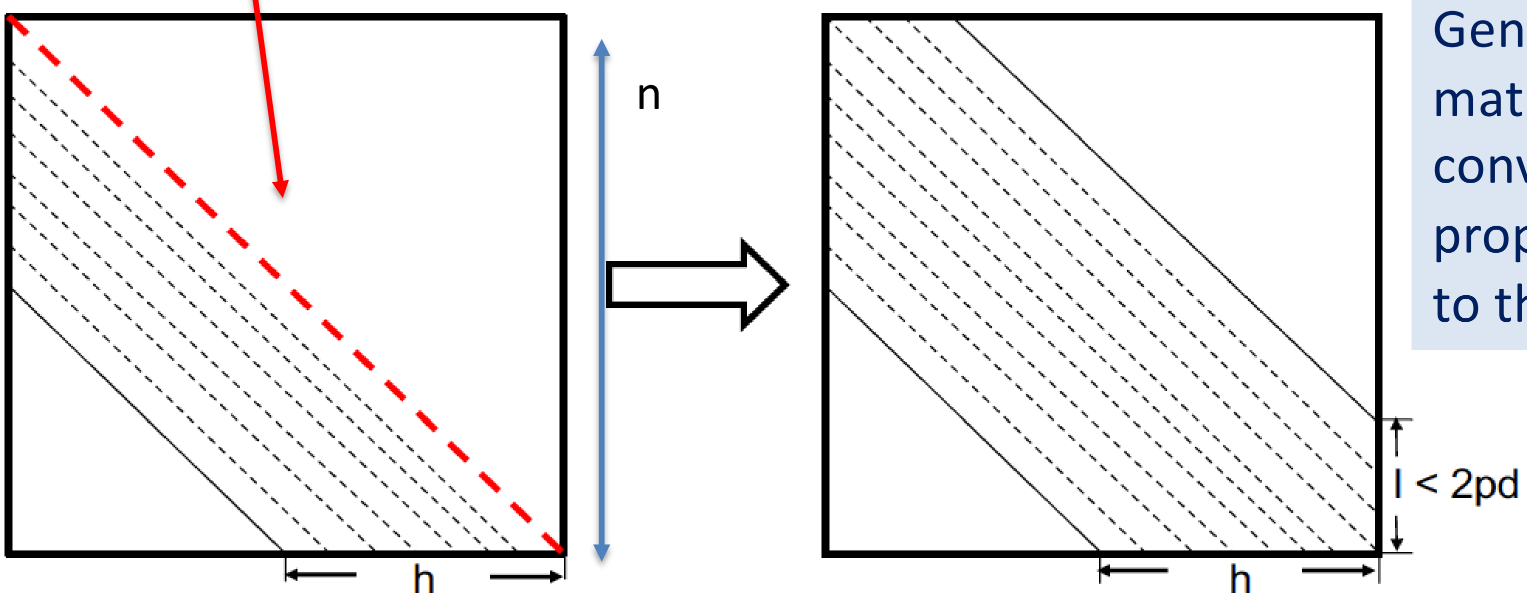
Outline

- Introduction
- New levels of programming (Graphs of Tasks, Network on chip)
- New methods and algorithms (Unite&Conquer, Stochastic Matrix,..)
- HPC and Machine Learning (GCN, Transformer,..)
- **Generators of Data Sets and matrices for brain-scale applicatons**
- What post-exascale plateforms and programin paradigms

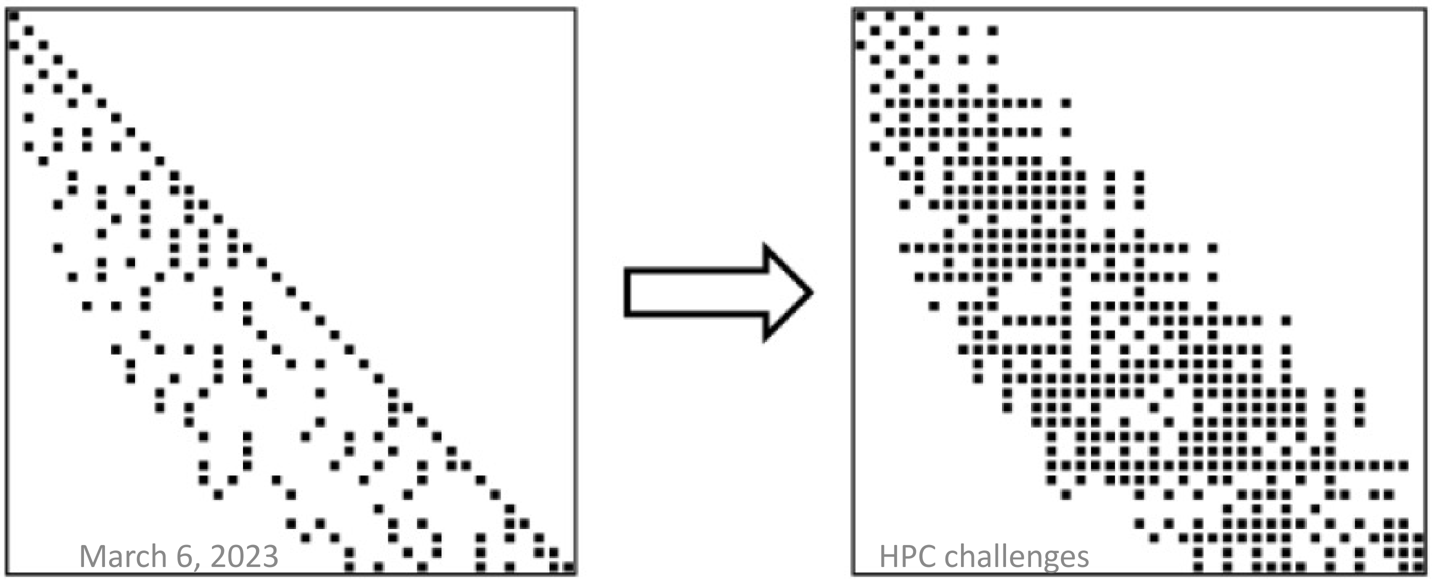
1 - Generator of non-Hermitian matrices, from given spectrum

given spectrum

Generated matrix



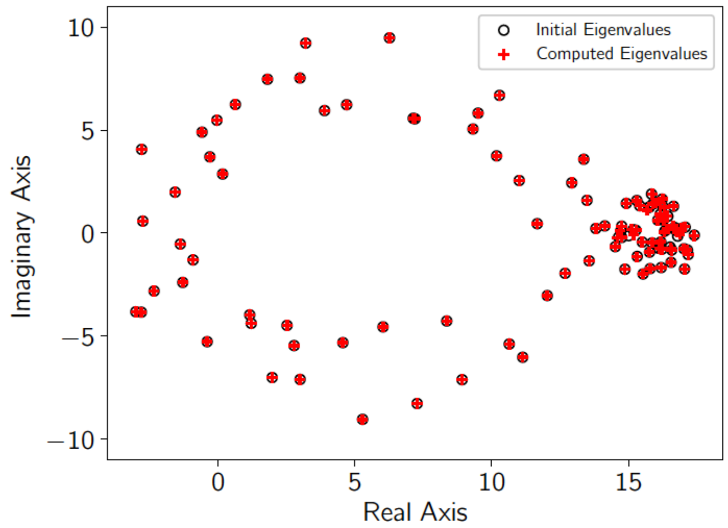
Generate sparse matrices to evalaute convergence and other properties with respcet to the spectrum



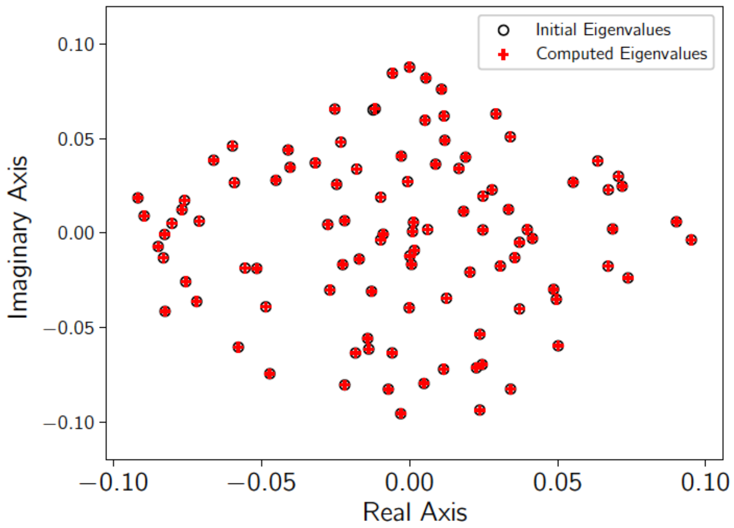
Directly compute in parallel.

Using only the memory size to store the final matrice! CSR or others formats

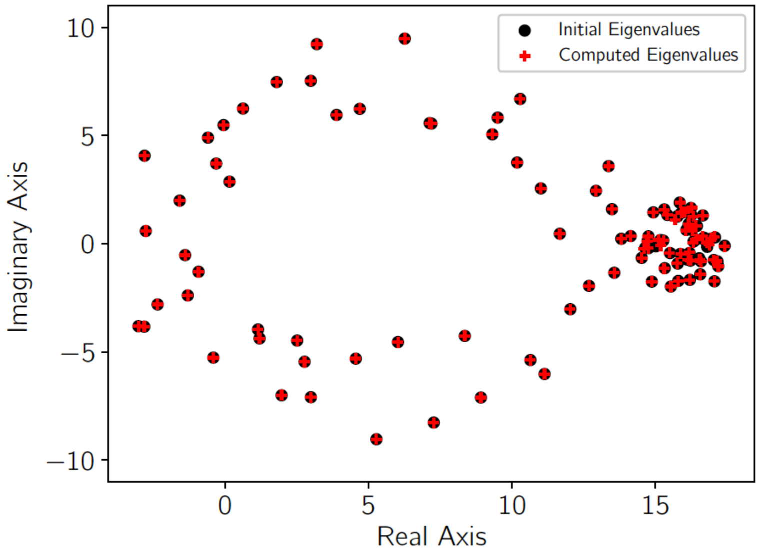
Example



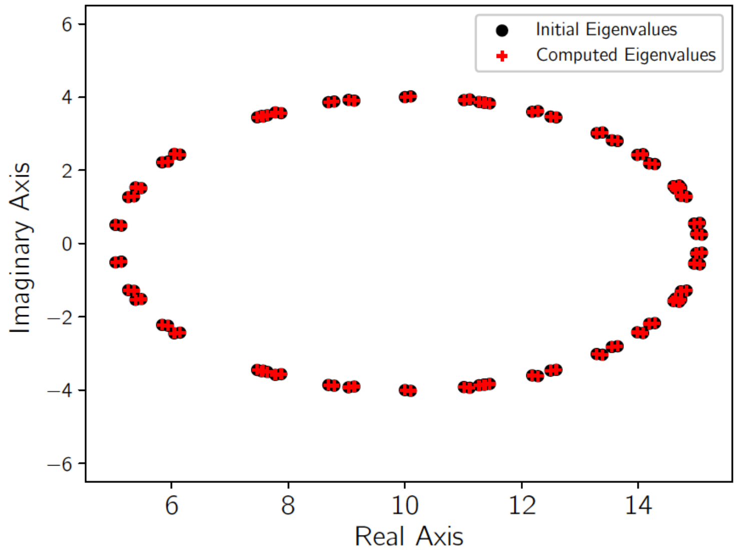
(a) Dominant Clustered Eigenvalues: acceptance = 94%, max *error* = 3×10^{-2}



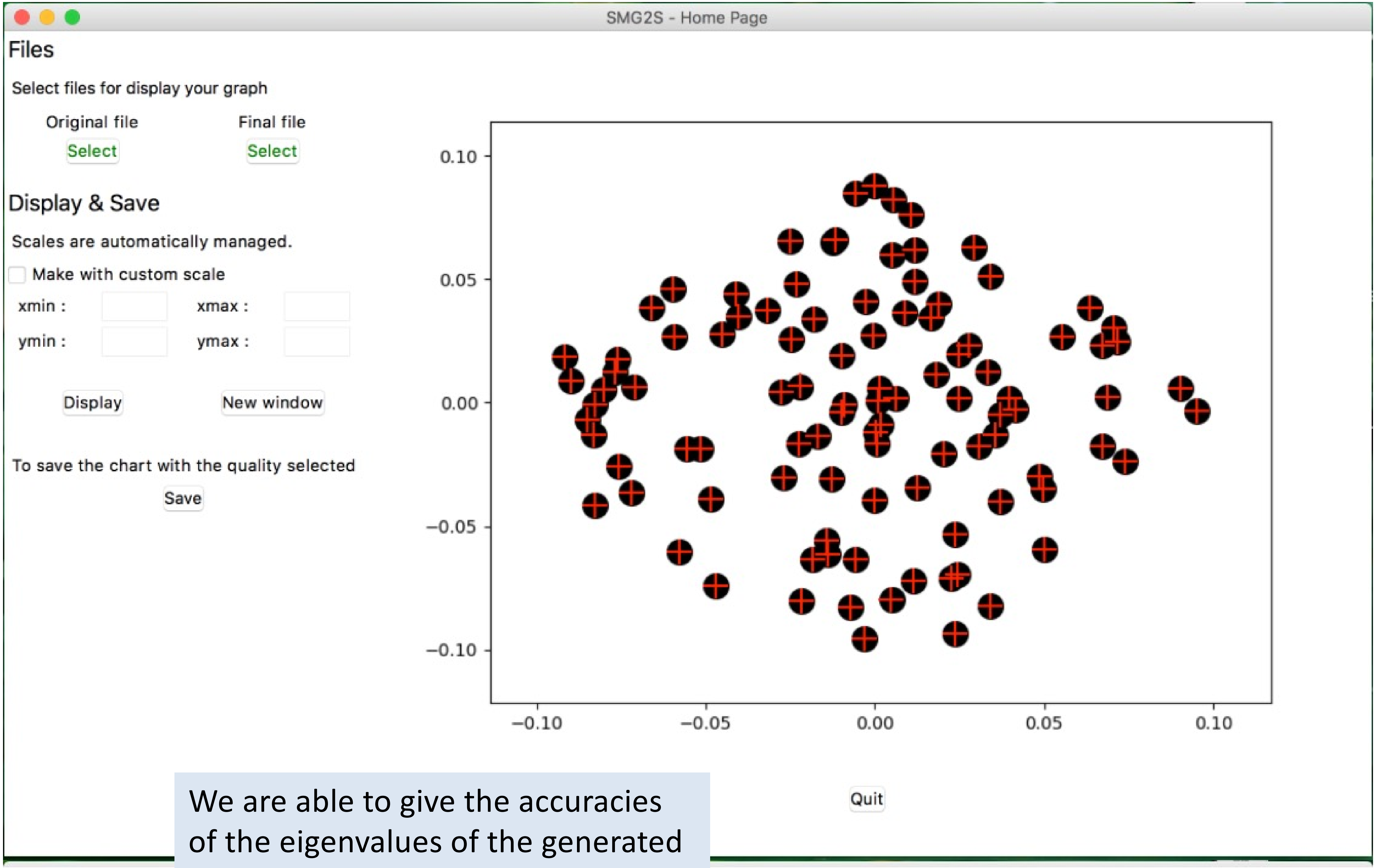
(b) Clustered Eigenvalues: acceptance = 100%, max *error* = 7×10^{-5}



(j) Dominant Clustered Eigenvalues: acceptance = 94%, max *error* = 3×10^{-2}

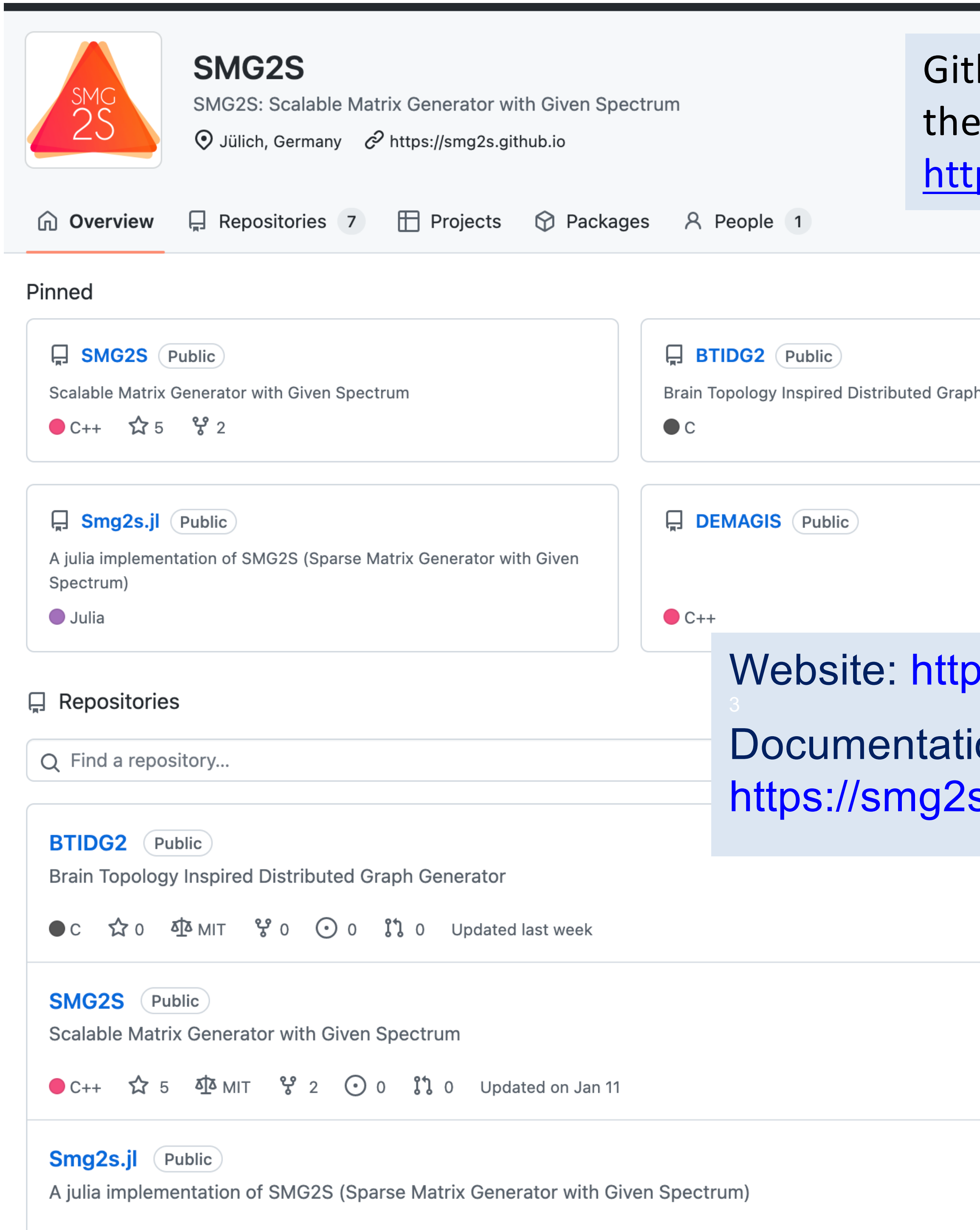


(k) Conjugate and Closest Eigenvalues: acceptance = 100%, max *error* = 3×10^{-7}



We are able to give the accuracies of the eigenvalues of the generated matrix compare to the given ones

Github Organisation for hosting all the generators: SMG2S and BTIDG2 https://github.com/SMG2S



Website: https://smg2s.github.io

Documentation: https://smg2s.github.io/files/smg2s-manual.pdf

2 – Distributed and Parallel Generator of brain-scale graph-matrices

As we are speaking about "brain scale", we generate a graph as close as possible from the brain structure (to be updated with data from several researches – Neurospin/CEA)

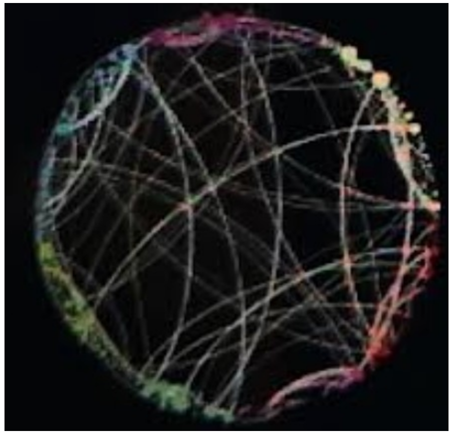
Graphs, insipered form the topolgy of the humain brain

- Aprox. 10^{11} Neurons,
- Up to 10 000 (different) conections,
- Several parts (left, rigth, entiric,…),
- Several feature per neuron.

It is also a possible to generate such sparse matrices for other kind of experiments

Graphs (sparse matrices) : several densities of nodes-neurons on each part, and several densities of connection from one part to another one (to be set in the future from data coming from RMI brain topology researches).

If we add some features to each neuron, we have a data set which may be adapted for **Graph Convolutional Network analysis, and others approaches**



Size of the data set : 10^{11} x 10^4 + 10^{11} x number of features

It would be very expensive to upload the data (I/O), to experiment with several hypothesis : we have to generate the data directly in Parallel without I/O, using sparse adjaceny non-symetric matrices)

Brain

Parts of the brain

- part name
- % of connection to the opposite side
- % of connection to other parts
- number of neuron types in each part
- total number of neurons

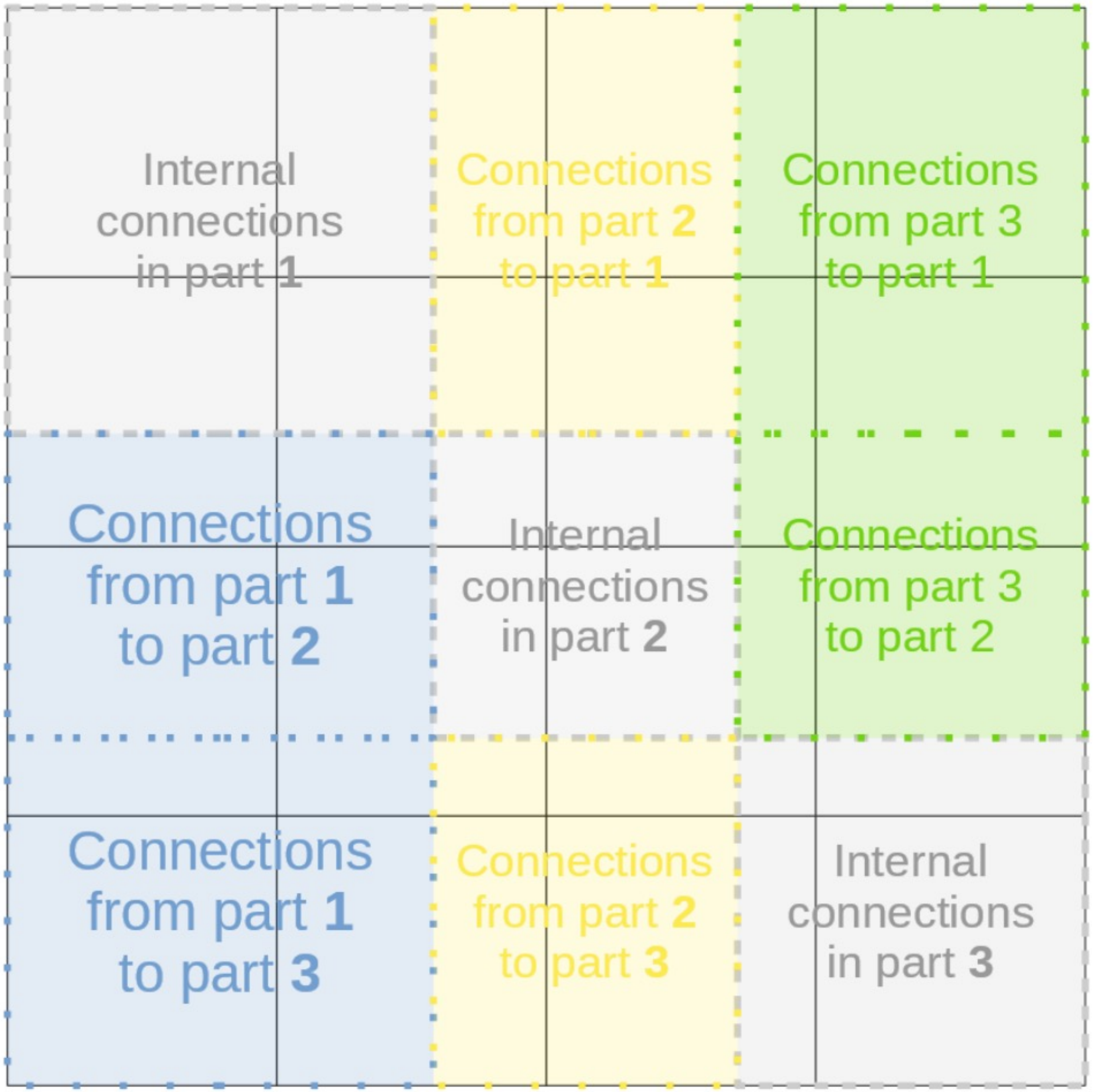
- Neurons

- neuron name
- number of neurons for each type of neuron
- number of connections for these neurons

+ features

Very sparse and large non symetrical matrices

The connection inside each part, and betwwen parts are parmatrized and randomly set, for the moment, waiting more data.



+ dense rectangular matrix for the features (n * number features)

PageRank? : ranking of the neurons (personalized PageRank would analyse the topological impacts of on set of neurons with the others)

Data set for Graph Convolutional Networks : to avoid oversmoothing, we drop edge and/or nodes. The ranking of the nodes (topologie – random walk based – **Pagerank**) may be use to optimize the dropping (method RankedDrop, IEEE Big Data 2022, Tokyo)

We first expriment on not too large graphs with such structures, the generation of the graphs and the ranking of the nodes, on different plateforms and supercomputers.

The % of connections between neurons-nodes are for the moment not based on the state-or-art, we are just evaluating our algorithms.

We don't compare with version with I/O as it is not relevant as it would be too expensive, Even if the matrices are not to large . We don't want to consume such energy.

On the future, we would like to run on exascale supercomputer to generate really brain-scale graphs, and run (personalized) PageRank methods as examples.

How do neurons operate on sparse distributed representations? A mathematical theory of sparsity, neurons and active dendrites[1]

Subutai Ahmad[1]*, Jeff Hawkins[1]

[1]Numenta, Inc., Redwood City, CA, USA

From the on line documentations

What is BTIDG2?

BTIDG2, short for Brain Topology Inspired Distributed Graph Generator, is able to generate very large graphs which is inspired by the topology of humain brain. This matrix generated is implemented based C, and parallised based on MPI.

As a software, **BTIDG2** ensures that:

- the generator must be able to work for large sparse matrices without any I/Os
- the generator must work in a distributed fashion
- support multiple (COO and CSR) distributed sparse data formats
- the generator must be as configurable as possible
- a ***converter***, allowing to convert a data file containing the brain information as we found it on the internet into an input file for our matrix generator.

BTIDG2's Documentation

INTRODUCTION

USER DOCUMENTATION

API

Indices and tables

BTIDG2 github repository:
https://github.com/SMG2S/BTIDG2

BTIDG2 website:
https://smg2s.github.io/BTIDG2/

License

Brain Structure

Structs

struct **`BrainPart`**

Structure containing the information of a part of the brain.

It contains :

a. The number of neuron types that can be encountered in the part
b. the cumulative distribution of neurons in the neuron types
c. the probability of connection (for each type) to other parts of the brain.

This structure depends on the brain to which it belongs.

Public Members

int **`nbTypeNeuron`**

the number of neuron types that can be encountered in the part

double ***`repartitionNeuronCumulee`**

the cumulative distribution of neurons in the neuron types

double ***`probaConnection`**

the probability of connection (for each type) to other parts of the brain

M

struct **Brain**

Structure containing the information of a brain.

It contains :

a. the total number of neurons
b. the number of parts in the brain
c. the indices (neurons) at which parts start
d. the brain parts (see BrainPart).

Public Members

long long **dimension**

the total number of neurons

int **nb_part**

the number of parts in the brain

long long ***parties_cerveau**

the indices (neurons) at which parts start

BrainPart ***brainPart**

the brain parts (see BrainPart).

void **brainAdjMatrixCSR**(csr **M_CSR*, MatrixDist*BlockInfo*, Brain **brain*, int **neuron_types*, BrainMatrixInfo **debugInfo*)

Generates a CSR square adjacency matrix of dimension (*brain).dimension, corresponding to the brain passed as a parameter, in a two-dimensional process grid. A row/column of the matrix corresponding to a neuron, adjacency matrix means that it is the row-neuron that connect to the column-neuron

Condition : BlockInfo must have been filled with information suitable for the brain (Otherwise, the generation will not necessarily fail, but the generated matrix will not necessarily correspond to the brain)

Parameters

- `[in] BlockInfo` : structure containing information about the local mpi process ("block")
- `[in] brain` : Pointer to the brain, basis for the generation of the matrix
- `[in] neuron_types` : vector containing the types chosen for the neurons of the brain passed as a parameter
- `[out] M_CSR` : Pointer to a structure corresponding to a CSR matrix. At the end of the generation, contains the generated matrix.
- `[out] debugInfo` : **OPTIONAL** - Pointer to a debug structure or NULL. If not NULL, at the end of the generation, contains debug information such as the number of connections made per neuron (<=> number of 1 on each row), the total number of connections, etc.

int **get_nb_neuron_brain_part**(Brain **brain*, int *part*)

Function that returns the number of neurons in a specific brain part.

Function that returns the number of neurons in the brain part if index `part`, in the Brain `brain`

Return

number of neurons in the brain part

Parameters

- `[in] ind` : index of the neuron
- `[in] brain` : pointer to a brain

void **printf_recap_brain**(Brain **brain*)

Brain print.

Function that displays a summary of the brain passed as a parameter (useful for debugging a brain)

Parameters

- `[in] brain` : pointer to a brain

3 - The convergence depend of the spectrum. we may analyze the efficiency of the different preconditoners with respect to the spectra, fro very large sparse matrices

Unite and Conquer method

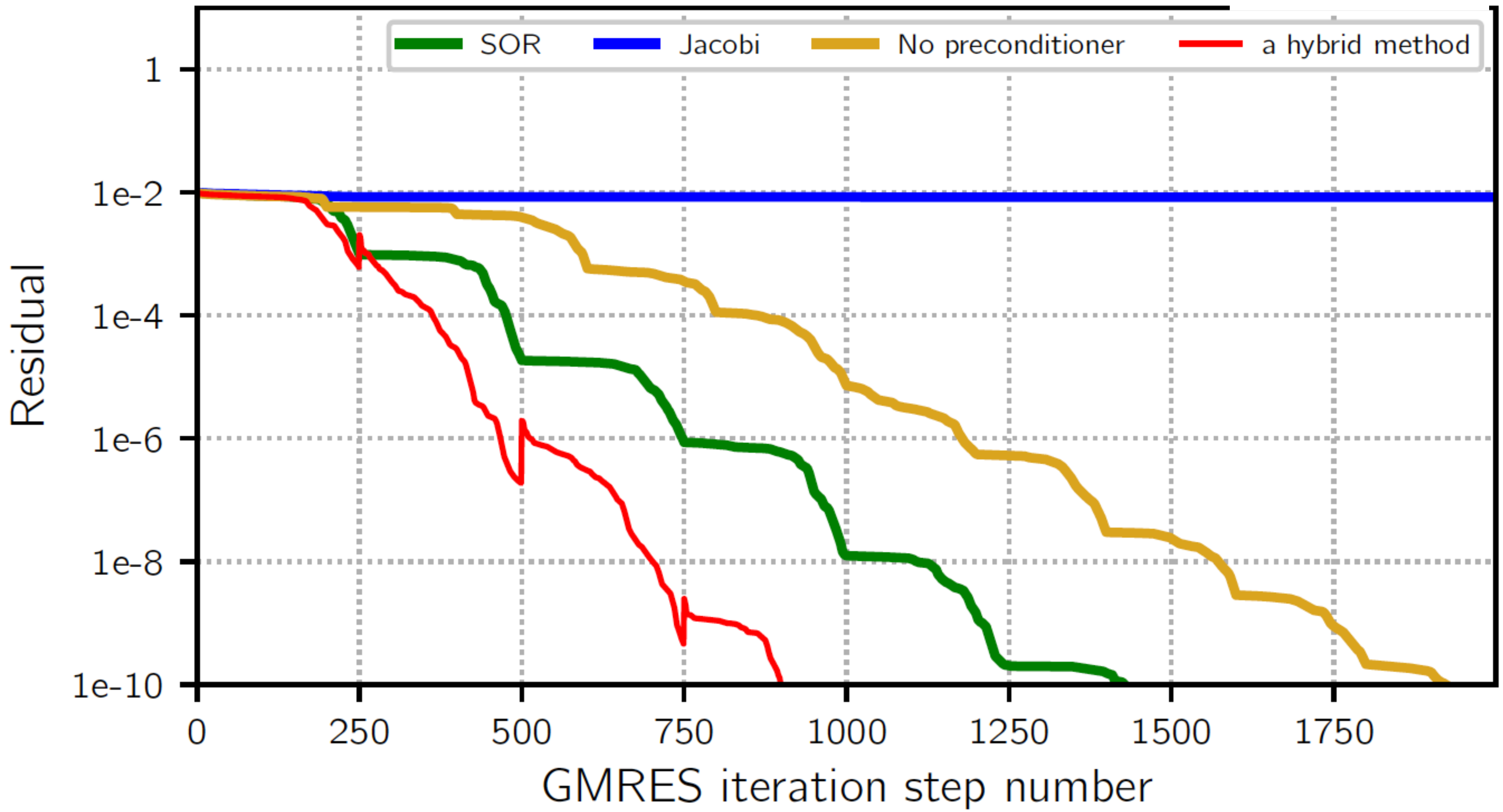
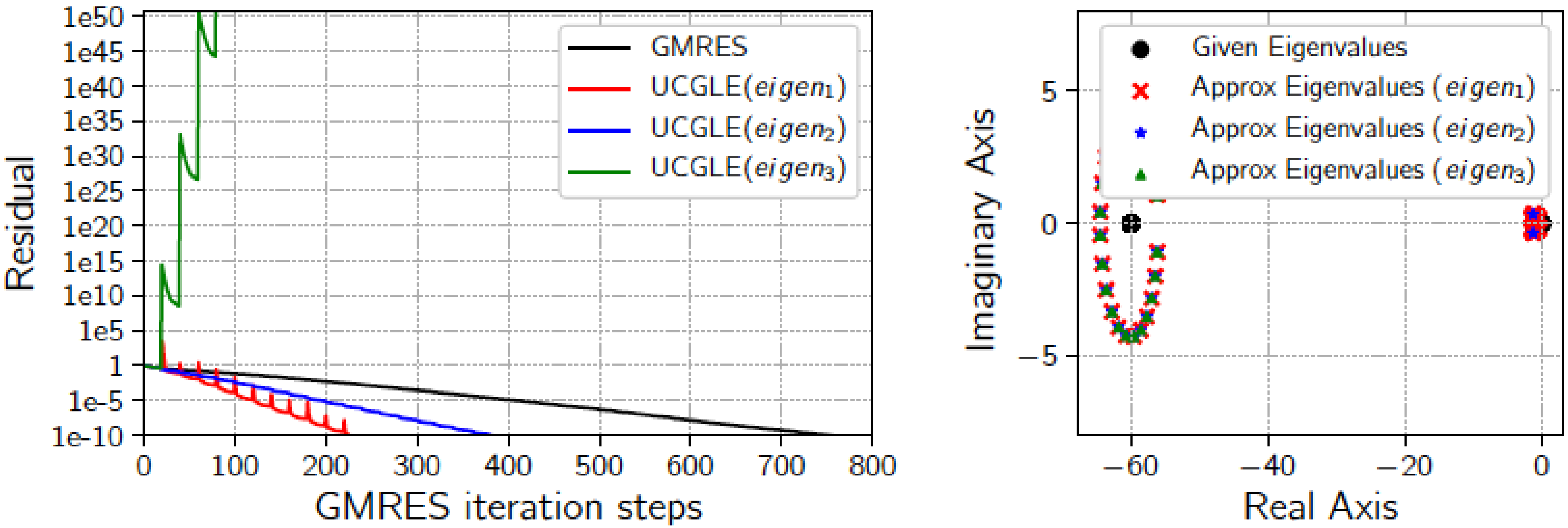


Figure: Convergence Comparison using a matrix generated by SMG2S.

Spectrum with several clusters
eigen1 : 1 cluster
eigen2 : 2 clusters
eigen3 : 3 clusters

Thianhe 2A

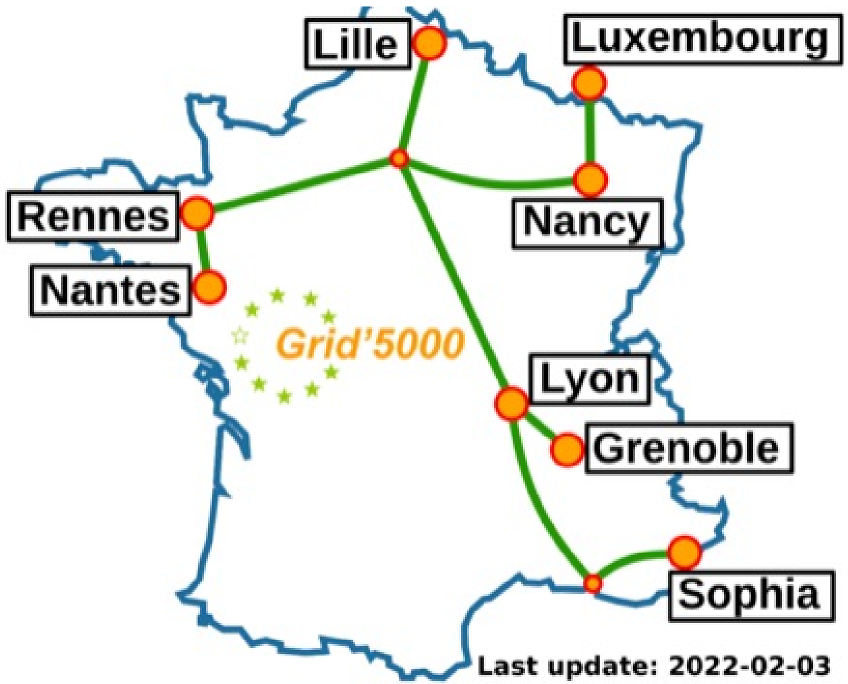


(d) Spectral Distribution IV: matrix size = 2000, m_g = 20, d = 10. UCGLE($eigen_1$) has 3× speedup, and UCGLE($eigen_2$) has 2× speedup.

Xinzhe Wu, Serge G. Petiton, Yutong Lu:
A parallel generator of non-Hermitian matrices computed from given spectra. Concurr. Comput. Pract. Exp. 32(20) (2020)

Expriments on GRID5000 (France), JEREDA (Julich), and Fugaku (Kobe)

Using only 256 cores , to generate not too large sparse matrices



Hardware Configuration of the JURECA DC Module (Phase 2: as of May 2021)

- **480 standard compute nodes**
  - 2× AMD EPYC 7742, 2× 64 cores, 2.25 GHz
  - 512 (16× 32) GB DDR4, 3200 MHz
  - InfiniBand HDR100 (NVIDIA Mellanox Connect-X6)
  - diskless
- **96 large-memory compute nodes**
  - 2× AMD EPYC 7742, 2× 64 cores, 2.25 GHz
  - 1024 (16× 64) GB DDR4, 3200 MHz
  - InfiniBand HDR100 (NVIDIA Mellanox Connect-X6)
  - diskless
- **192 accelerated compute nodes**
  - 2× AMD EPYC 7742, 2× 64 cores, 2.25 GHz
  - 512 (16× 32) GB DDR4, 3200 MHz
  - 4× NVIDIA A100 GPU, 4× 40 GB HBM2e
  - 2× InfiniBand HDR (NVIDIA Mellanox Connect-X6)
  - diskless

ATOS-BULL

We expriment with respect to several paramters :

- The matrix size
- The number of core
- Grid size and others pameters for the graph-matrices

Block CSR
NNZ = 3,5 % of N^2
Densities
- 0.5% inside each parts
- 5% between parts

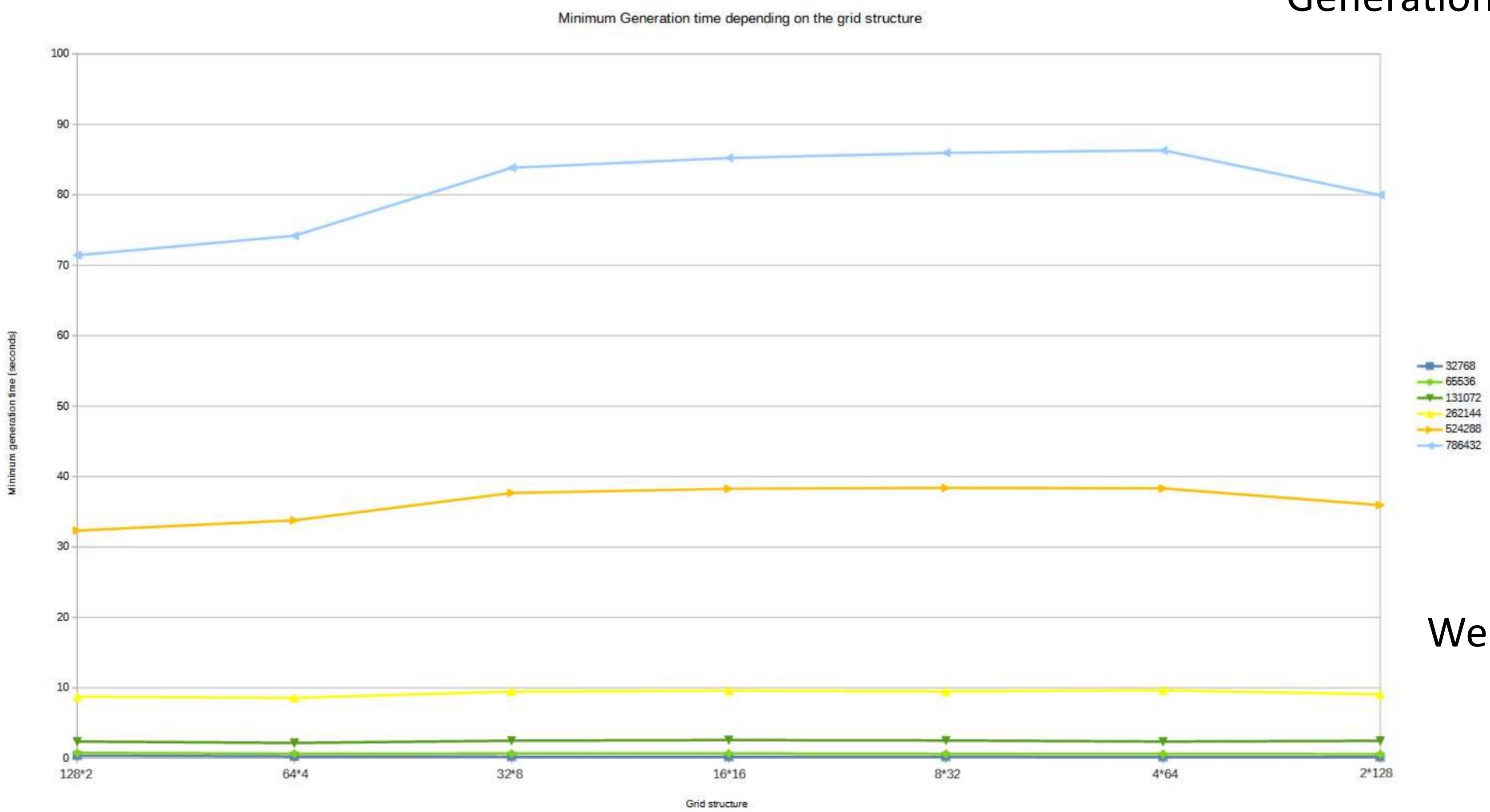
Diferent numbers of blocks

JEREDA DC, Julich

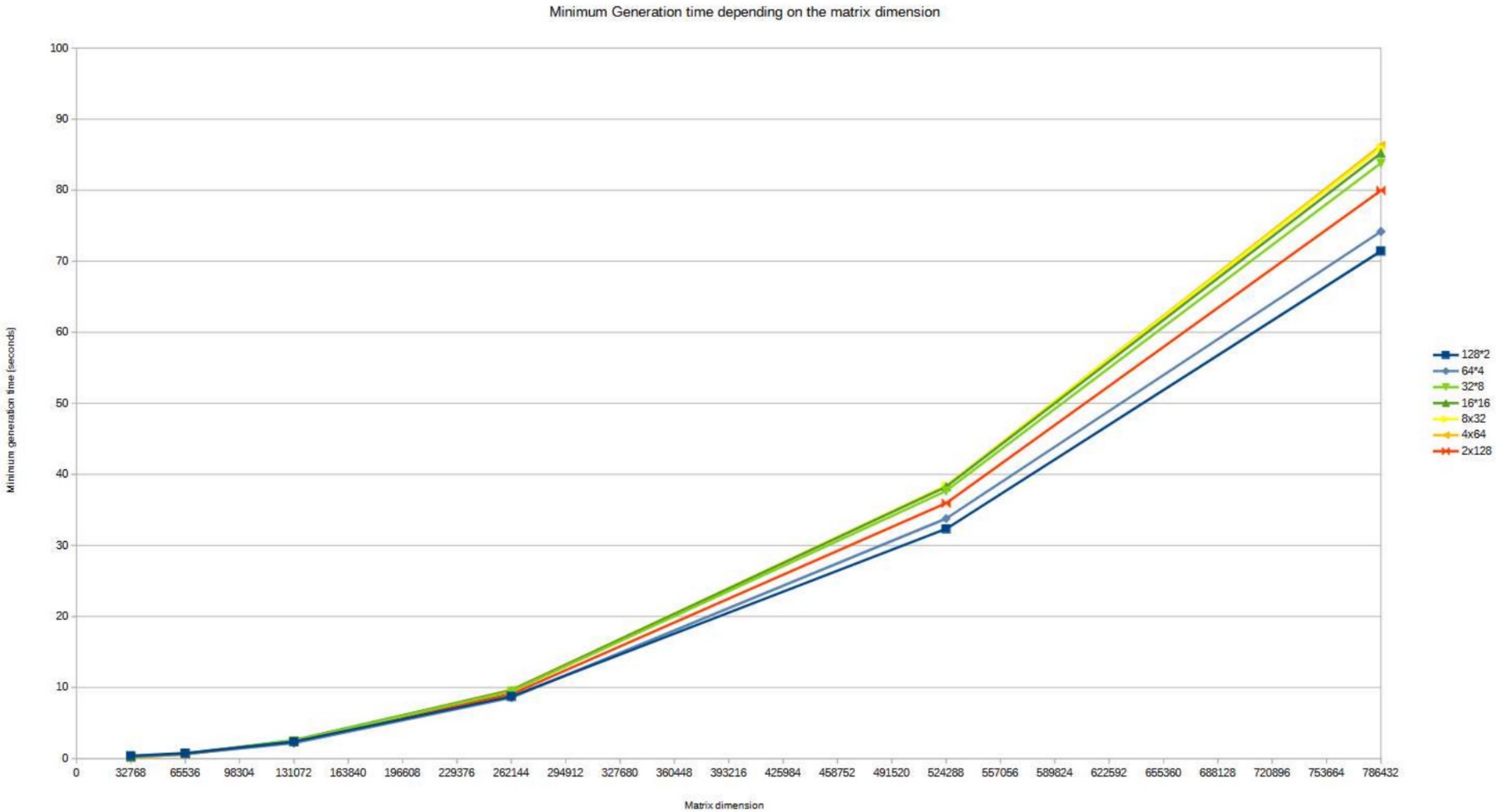
Process Grid tests

We applied the following tests to matrices whose dimension vary from 30k to 80k, while keeping the same number of allocated resources (256 cores)

Generation



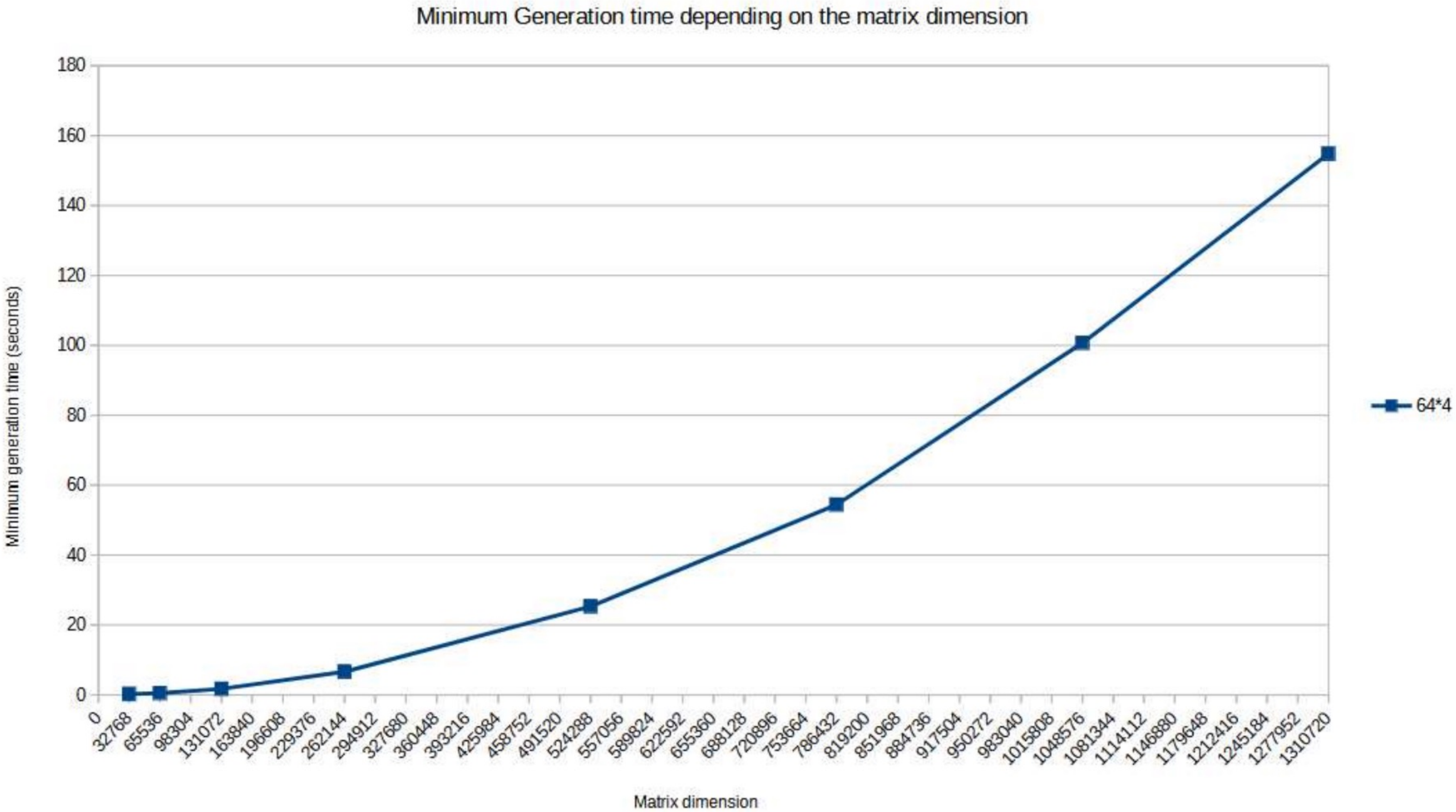
Weak scaling



Minimum Generation time depending on the matrix dimension

Matrix size tests

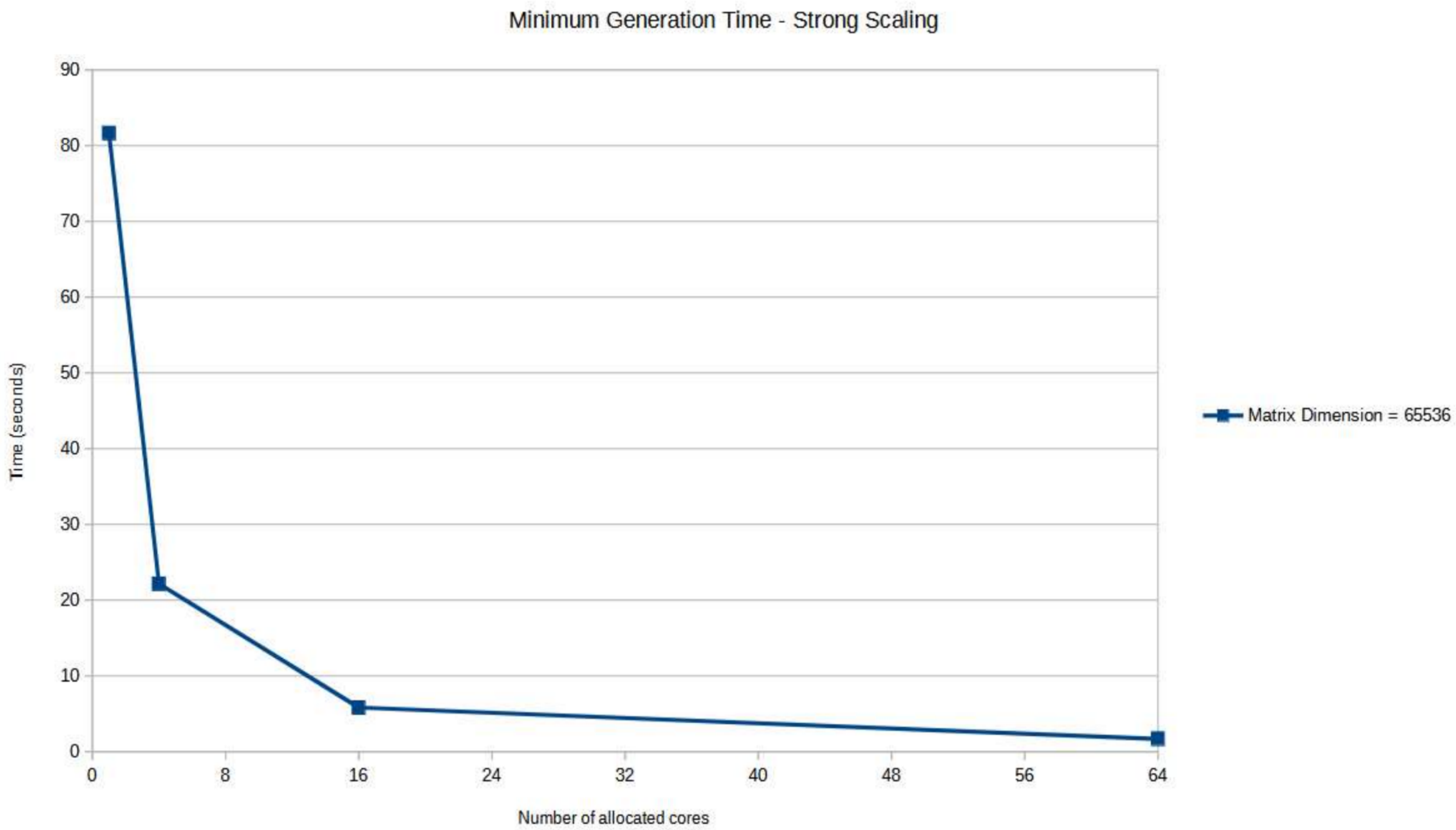
The following test were carried out by varying the size (dimension) of the matrix, while working with the same allocated resources (256 cores)



Minimum generation time depending on the matrix dimension

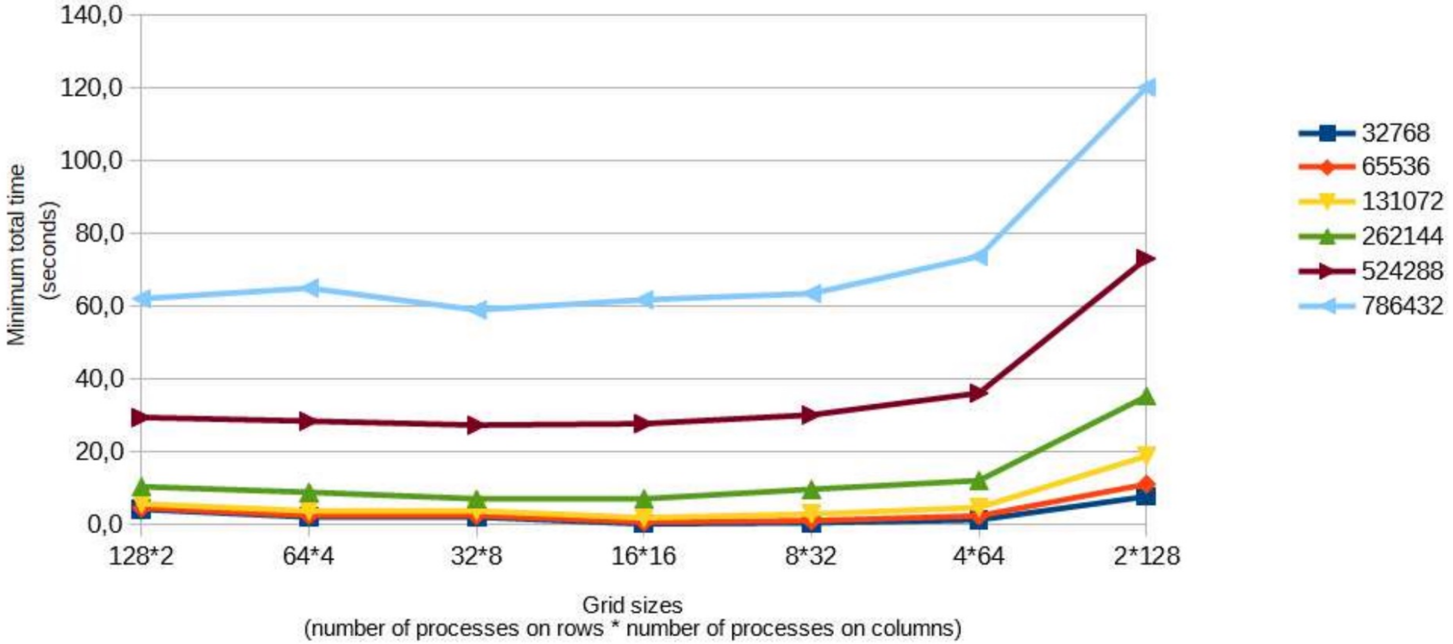
Strong scaling experiments

In this experiment, we increase the number of allocated core while keeping the same problem size (a matrix with a dimension equal to 65536)



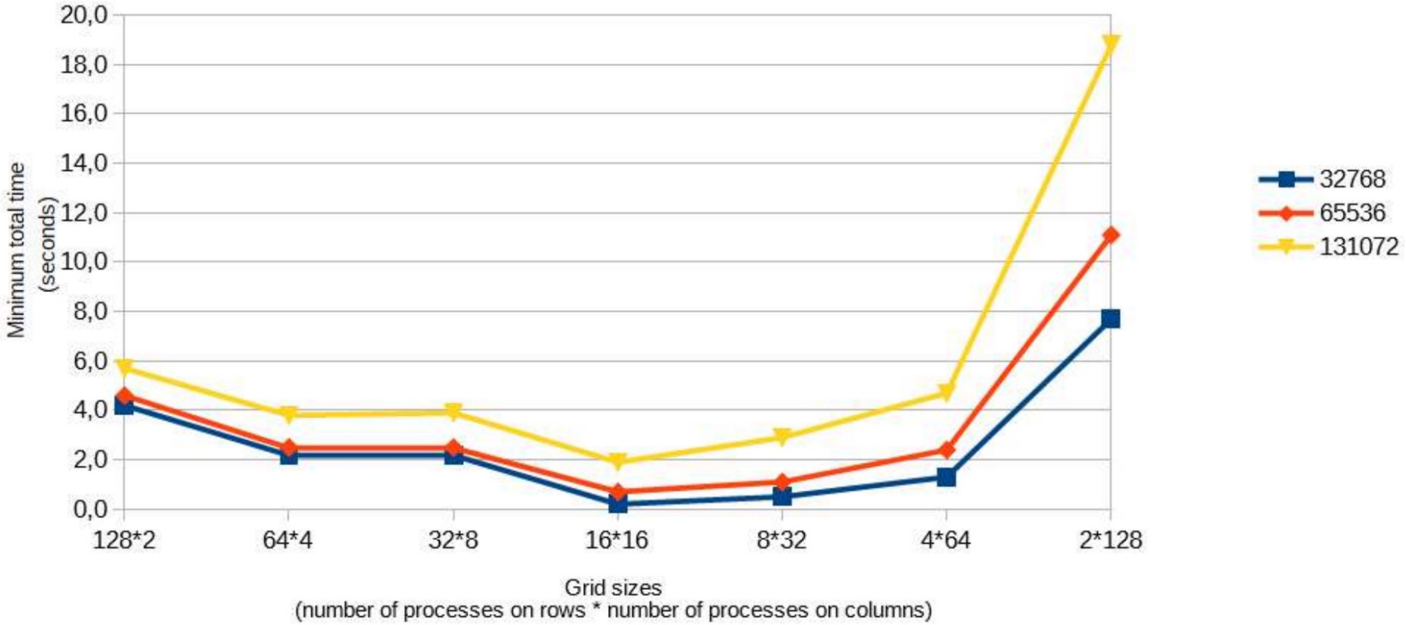
Minimum generation time depending on number of allocated core

GRID5000



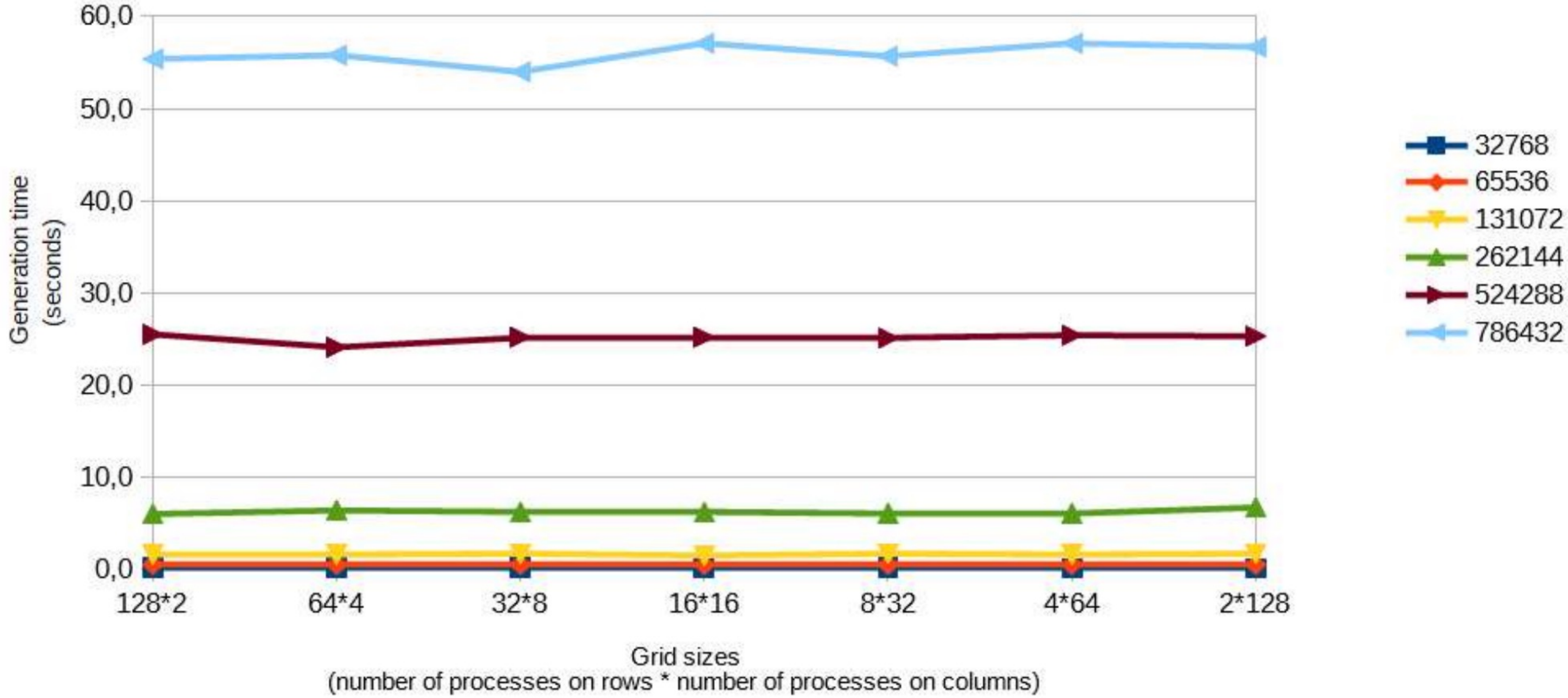
Minimum total time (generation and pagerank) depending on the grid structure

Pagerank : seqence of sparse matrix-vector products + reduction_with_add



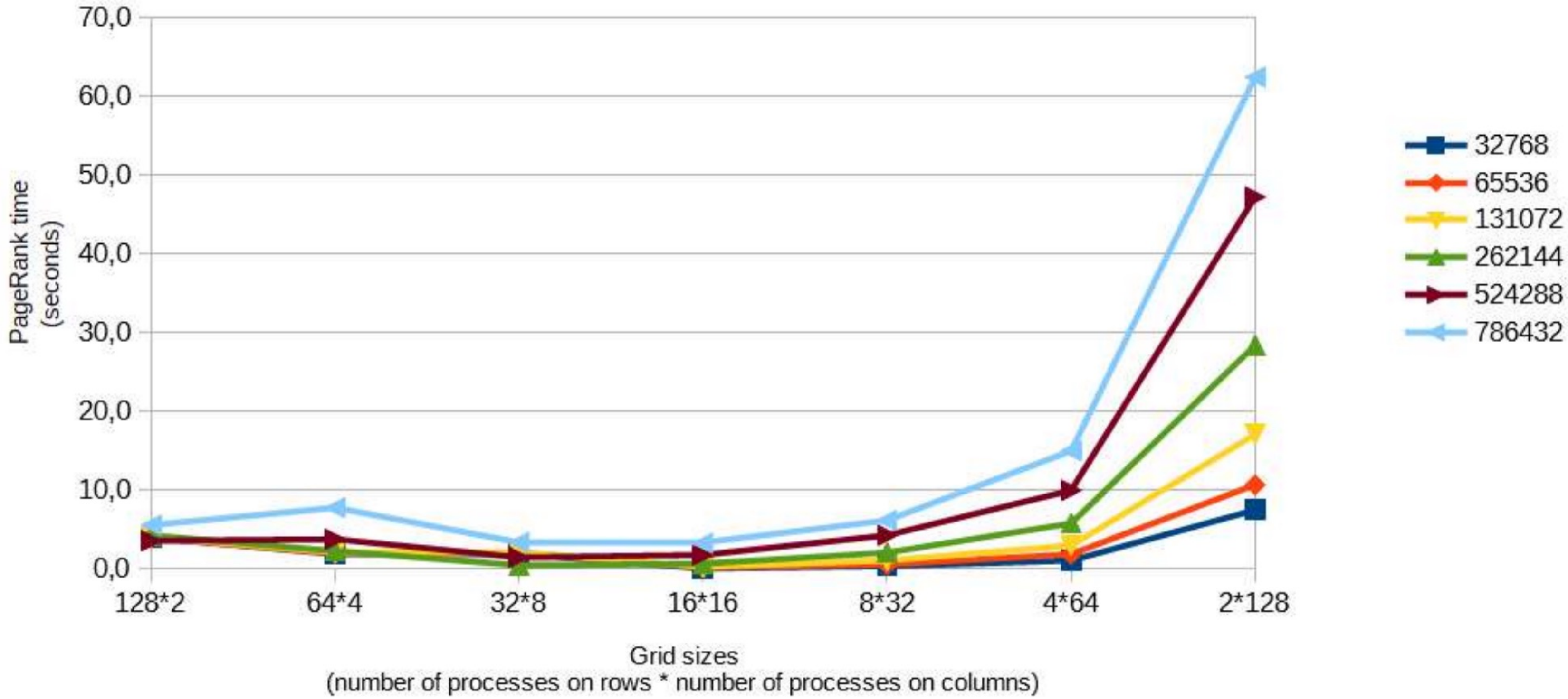
Minimum total time (generation and pagerank) depending on the grid structure (zoom on "little matrices")

Generation time :

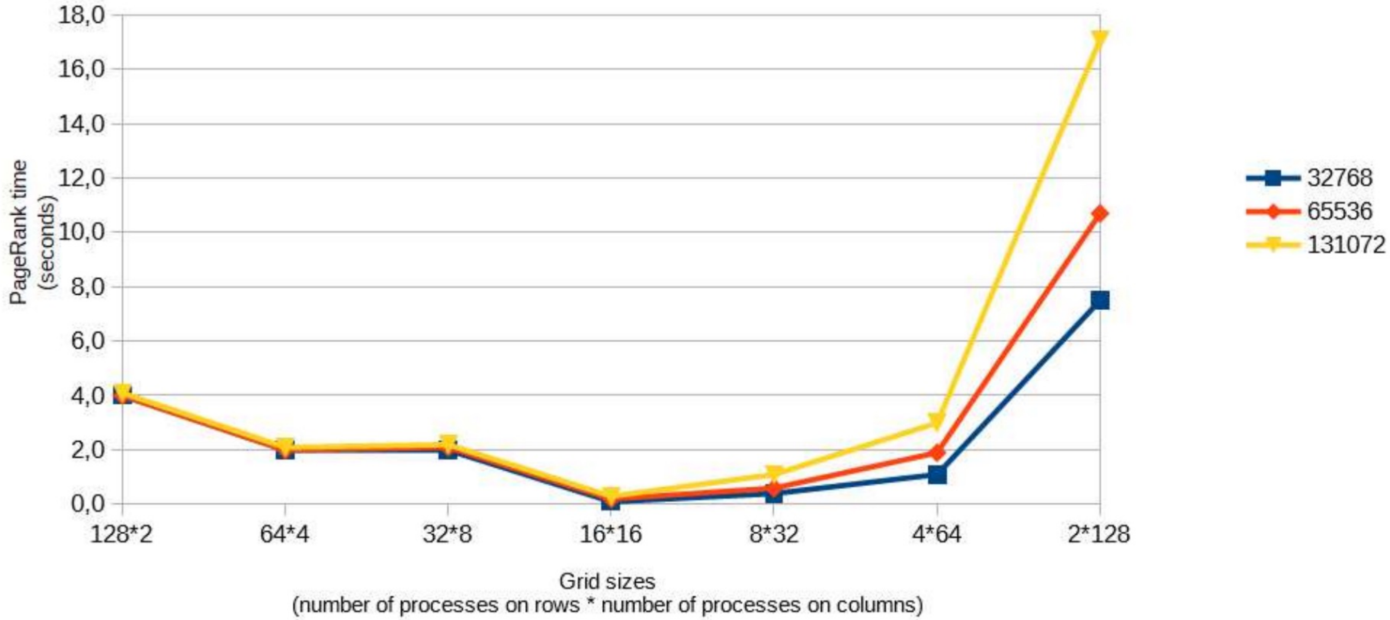


Minimum generation time depending on the grid structure

PageRank time :

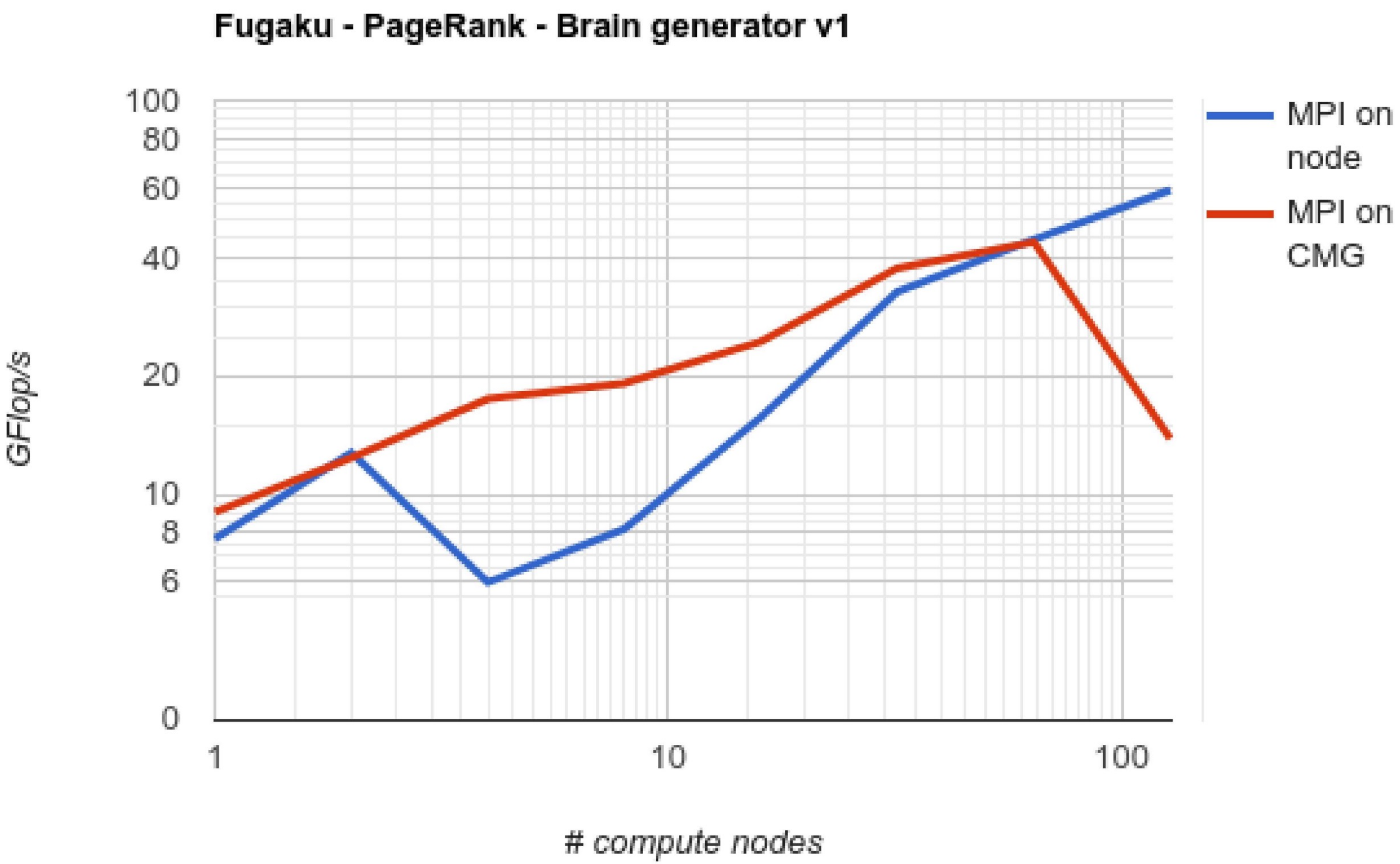


Minimum PageRank time depending on the grid structure



Minimum PageRank time depending on the grid structure (zoom on "little matrices")

First experiments done by Maxence Vandromme, part of a collaboration with Mitsuhisa Sato a and Miwako T



Outline

- Introduction
- New levels of programming (Graphs of Tasks, Network on chip)
- New methods and algorithms (Unite&Conquer, Stochastic Matrix,..)
- HPC and Machine Learning (GCN, Transformer,..)
- Generators of Data Sets and matrices for brain-scale applicatons
- **What post-exascale plateforms and programing paradigms**

Conclusion

We propose two generators of data to be able to expriment “brain-scale” applications without expensive I/O, both for CSE and Machine Learning ; directly compute in parallel.

These generators may also be used to preconditione or pre-train (transfert learning) methods, while we upload other data, if we have enough knownledge of the spectrum or of the topology of the targeted graphs-matrices

HPC challenge and new computing frontiers

- Arithmetic : mixed, new normalisations?
- New methods : optimisation of operations, of iterations-epochs (unite&conquer or Neural-Network ensembles), minimization of communications,
- Hierarchical architecture : cluster-cloud-distributed + parallelism + NOC chips, (accelerated) set of cores,
- Progamming paradigms : graph of task, PGAS-data parallelism, vectorial
- (Non-Hermitian) Sparse linear algebra : sequence of matrix-vector products, sequence of (dynamic) sparse matrix products, eigenvalues
- New applications : “brain scale” bigbird transformer, AI, human brain, …