Université de Lille

CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

cnrs dépasser les frontières

Challenges for Extreme Scale Computational Science and Machine Learning

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

Outline

1. **Introduction**
2. Programming Paradigms
3. Methods and Algorithms (Unite&Conquer, PageRank)
4. HPC and Machine Learning (GCN, Transformer,..)
5. Generators of Data Sets and matrices for brain-scale applicatons
6. Conclusion

Introduction

HPC and BigData convergence towards machine learning and AI : new scientific frontiers, and challenges

Computational science : the “classical” quite predict evolution.

In a 2005 interview, Moore himself admitted that "...the fact that materials are made of atoms is the fundamental limitation and it's not that far away...We're pushing up against some fairly fundamental limits so one of these days we're going to have to stop making things smaller." [7]

Probably on the 2020s

- End of the Moore law
- Network on chip : all cores of one processor don’t share memories
- More acellerators *(gpu, amd, matrix2000, ….)*
- High hierachical architecture
- Cluster-Cloud for HPC
- More complex pogramming paradigms, resilience (fault tolerence), energy consumption
- CSR, ELLPACK compressed format, C++, 64IEEE bit-arithmetic, Ax for iterative methods.

BigData (Data science)

- Data centers and clouds are deployed worldwide and become more importants than supercomputer centers
- Data are so larges that we may do science without any modelization, for some applications
- MapReduce is also used by linear scientific methods: matrices, mesches and others, saw as sets of data
- Energy consumption, security
- COO, Python, Spark, HDFS, Integers, pixels, other data, 16-32 bit-arithmetic, AB for linear algebra

Machine-Learning, AI

- Tensors (multi-dimensional arrays), Graphs
- Pyhton, Pytorch, TensorFlow, API, 16-32 bit-arithmetic.
- Result of the convergence of computational and data worlds
- Ecosystem still to be defined
- New hardware for AI (Tensor Processing Unit (TPU),….)
- Scientific problems saw as a “game” or an “learning process”.

An Exascale Heterogenous World?

TOP500 "June 2023" list, LINPACK-LU

64 bits arithmetic.

Rank	System	Cores	(PFlop/s)	(PFlop/s)	(kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	304.47	7,404
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
6	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94.64	125.71	7,438
7	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93.01	125.44	15,371
8	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70.87	93.75	2,589

x >2

AMD accelerator

ARM SVE, no accelerators
6D hypercube network

x approx. 4

x approx. 5

Nvidia accelerator

China, former #1, "old" data

Frontier Remains As Sole Exaflop Machine And Retains Top Spot, Improving Upon Its Previous HPL Score

May 22, 2023

The 61st edition of the TOP500 reveals that the Frontier system out of Oak Ridge National Laboratory (ORNL) remains the only true exascale machine on the list.

On the list as China don't give anymore information, but they probably have at least 2 exscale machines.

Cerebras propose an exascale machine (but wihout 64 bit arithmetic, not TOP500-LU benchmarck).

Up to approximatly 9 millions of cores, distributed along many racks : distributed and paralel programming, graph of tasks to be scheduled (with I/O and communciation optilization)

HPCG, “June” 2023

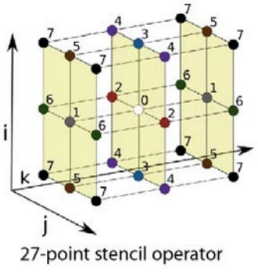
HPCG Benchmark Technical Specification

Michael A. Heroux
Scalable Algorithm Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS 1320

Jack Dongarra
Piotr Luszczek
Electrical Engineering and Computer Science Department
1122 Volunteer Blvd University of Tennessee
Knoxville, TN 37996-3450

2013
First list: 2017

Iterative méthod. Preconditionned Conjugate Gradient
64 bit arithmetic.
27 Band matrices



27-point stencil operator

Stil structured

It is a more realistic benchmark, with “sparse-regular” matrices
Having reduction operation (scalar products)

#1 : Fugaku, since several semesters. “Just” 16 Petaflops
If we have sparse matrices with irregular patterns, the performance would be around 1 Petaflops (1000 time less that the top500 #1), reaching just 1 Petaflop (see slides on experients on Fugaku)

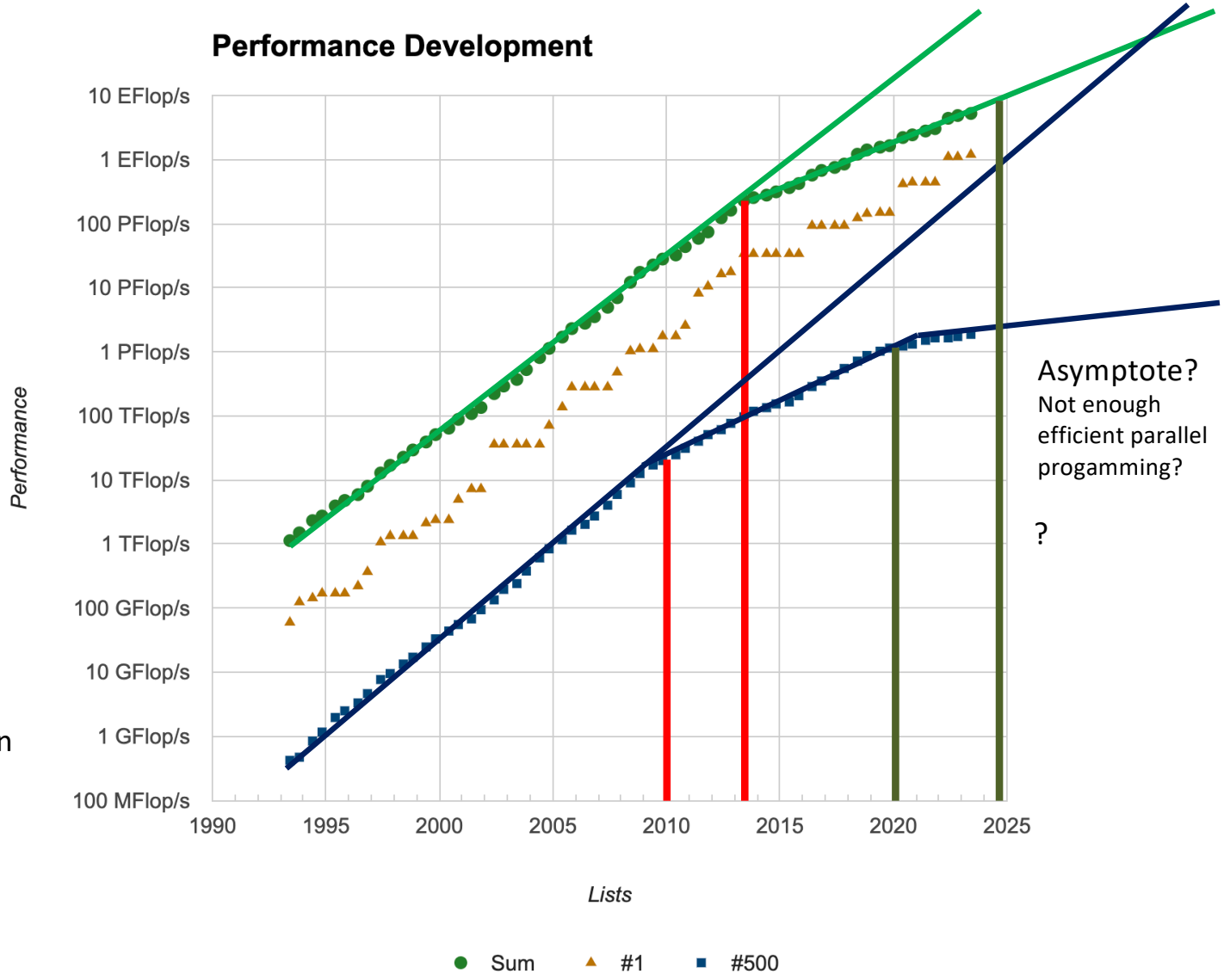
Thianhe would have simular performance than Fugaku, but sunway would have smaller efficiencies (extrapolated from former experiments).

Rank	TOP500 Rank	System	Cores	Rmax (PFlop/s)	HPCG (TFlop/s)
1	2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	16004.50
					0,016 Exa
2	1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	14054.00
					0,014 Exa
3	3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	3408.47
					0,003 Exa
4	4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	3113.94
5	5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	2925.75
6	8	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70.87	1905.44
7	6	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94.64	1795.67
8	9	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation	555,520	63.46	1622.51

TOP500 over the years

We have another level of programming, inside the ship. As Network on chip lead to distributed and parallel programming.

Example : hybrid programming MPI and OpenMP for several interconnected sets of cores wihout unique shared memories.



The slope of the curb was higher than the Moore law one, because the parallelism added an extra speedup.

Now we have ”just” this acceleration as the processor frequencies don’t increase anymore.

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. A priori, 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 the data are stored (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.

New programming paradigms have to be proposed and evaluated

Outline

1. Introduction
2. **Programming Paradigms**
3. Methods and Algorithms (Unite&Conquer, PageRank)
4. HPC and Machine Learning (GCN, Transformer,..)
5. Generators of Data Sets and matrices for brain-scale applicatons
6. Conclusion

1 - Network on Chip

and irregular “local” communications

Larger number of nodes
(task programming)

Network on chip :
Distributed and data parallelism

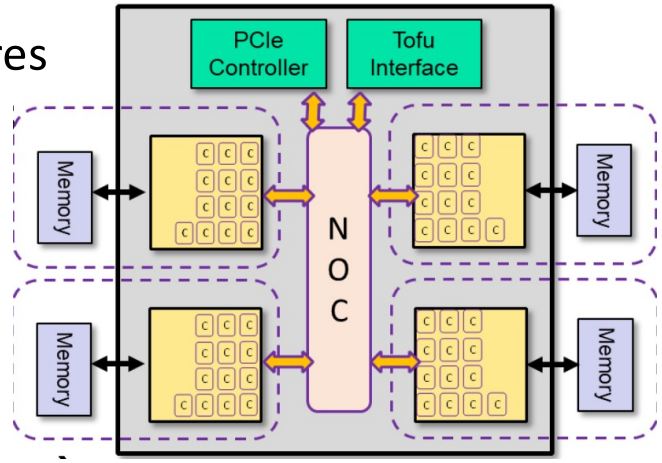
High hiearachy of execution models, which lead to several programming paradims for a given method : graph of tasks, PGAS, data parallel

Experiments on Fugaku :

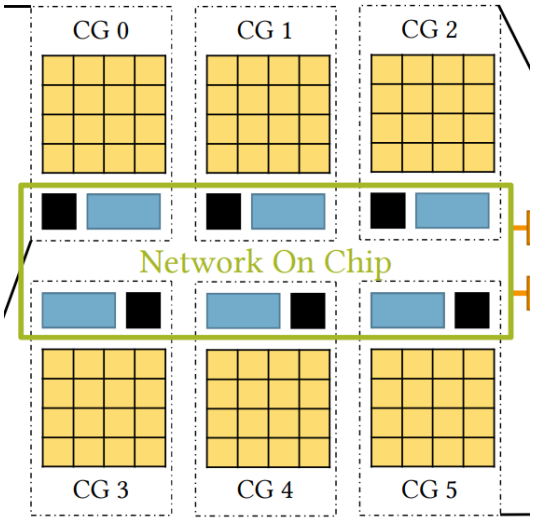
- 1 MPI “task” per chip (1 openMP per chip + SVE)
- 4 MPI “tasks” per chip (1 openMP per CMG + SVE)

For sparse matrices

4 x 12 cores



4 CMGs, SVE



Sunway

Sparse matrix : q-Perturbed, or from MatrixMarket

Starting from a C diagonal matrix (C ”main” diagonals), we randomly perturb the pattern

We randomly select some non zero elements of the diagonals with a probability q, and we ”send’ the selected elements to a random column of their row

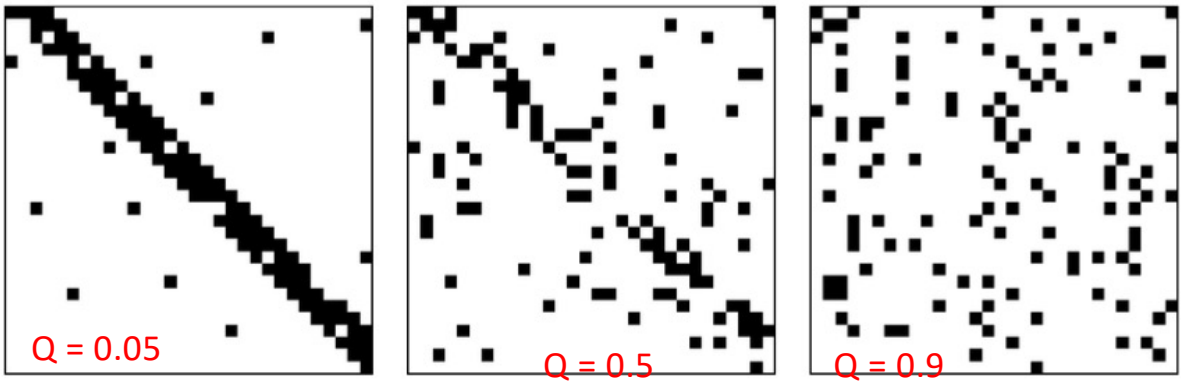


Fig. 1. C-diagonal Q-perturbed sparse matrix with C=4 and Q=0.05 on the left, Q=0.5 on the middle and Q=0.9 on the right

Remark : the distance to the diagonal is important for the distance of the communciations between cores

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

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

Collaboration avec RIKEN-Kobe

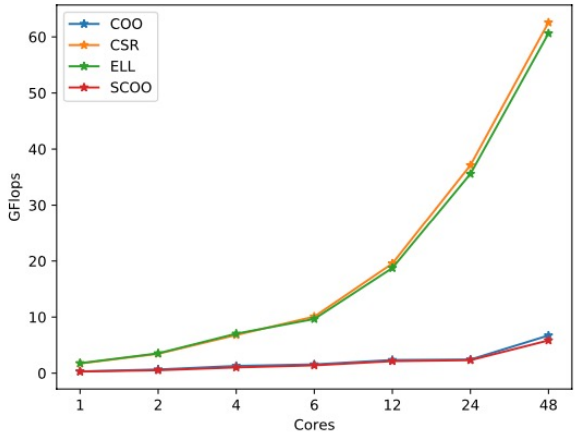


Fig. 4. $A(Ax + x) + x$ for COO, CSR, ELL and SCOO with nlpkkt120 as example

1 Fugaku node

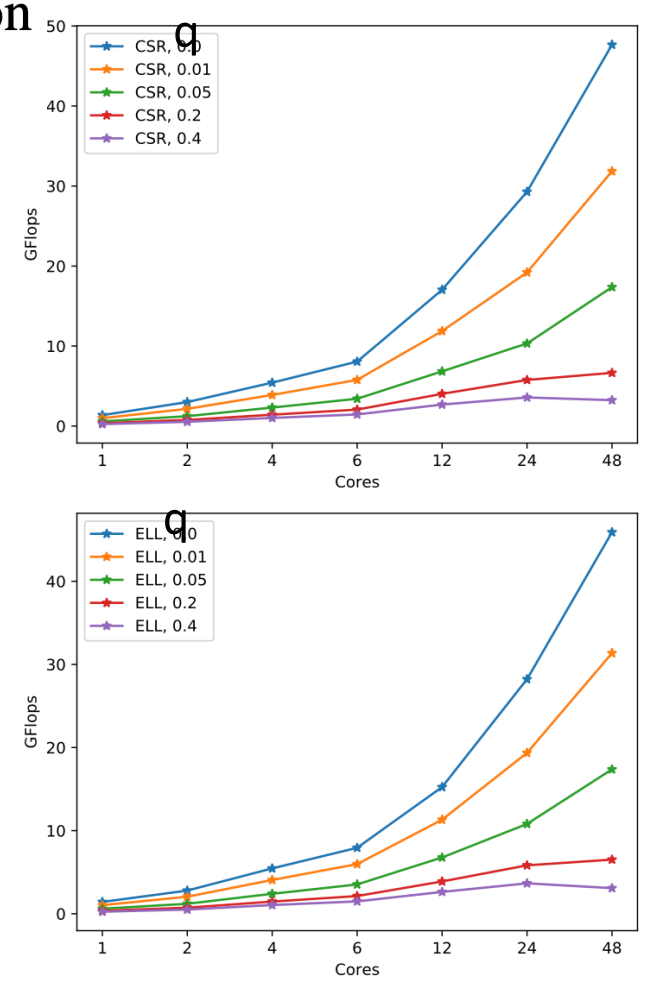


Fig. 8. $A(Ax + x) + x$ with OpenMP for C-diagonal Q-perturbed matrices with C = 16

1 or 2 Fugaku nodes

Ax, matrix NLPKKT120 (N = 3 542 400, NNZ = 96 845 792)

threads	COO		CSR		ELL		SCOO	
	1	2	1	2	1	2	1	2
1	7.9	10.0	75.6	127.2	81.7	145.5	6.0	11.9
2	3.8	4.4	75.0	136.4	72.4	135.0	6.0	11.9
4	5.4	4.0	73.6	142.2	80.3	135.0	5.9	11.7
6	5.9	2.6	73.8	130.0	64.2	131.7	5.8	11.4
12	3.1	3.1	72.5	131.5	69.9	129.4	5.5	11.0
24	3.0	2.4	71.9	130.9	69.1	124.5	3.2	5.4
48	1.6	2.6	69.8	127.9	68.8	123.2	2.4	3.3
MPI	14.5	27.4	78.1	119.1	84.4	121.2	6.8	13.0

TABLE II

MPI AND MPI+OPENMP PERFORMANCE (IN GFLOP/S) FOR Ax WITH NLPKKT120 MATRIX ON 1 AND 2 NODES FOR COO, CSR, ELL AND SCOO STORAGE FORMATS WITH DIFFERENT NUMBER OF THREADS PER MPI PROCESS, KEEPING 48 THREADS TOTAL ON EACH NODE

OpenMP 2 CMG

OpenMPs Inside each CMG

MPI 2 nodes

SCOO : COO, but elements of a given row are together on the same node

Best result : ELLPACK, 145.5 Gflops, 1 OpenMP per core
CSR , 142.2 Gflops, 1 OpenMp per 4 cores
SCOO, 13 Gflops, MPI 2 nodes
COO, 27.4 Gflops, MPI 2 nodes

A(Ax+x), matrix NLPKKT120

threads	COO		CSR		ELL		SCOO	
	1	2	1	2	1	2	1	2
1	2.6	1.1	8.9	10.4	8.3	10.6	3.8	5.9
2	2.8	1.7	8.2	9.2	8.1	9.1	3.7	5.9
4	3.2	3.1	13.1	15.0	13.9	14.8	4.3	7.2
6	2.3	1.9	16.7	17.6	16.0	17.9	4.6	7.5
12	2.1	1.9	18.1	20.8	17.2	22.5	4.6	8.0
24	3.6	1.8	27.5	25.9	29.6	25.1	3.6	5.6
48	1.6	3.4	34.1	35.8	33.4	36.1	3.3	4.3
MPI	3.2	1.2	9.0	9.9	8.1	9.9	3.6	5.6

TABLE III

MPI AND MPI+OPENMP PERFORMANCE (IN GFLOP/S) FOR $A(Ax + x) + x$ WITH NLPKKT120 MATRIX ON 1 AND 2 NODES FOR COO, CSR, ELL AND SCOO STORAGE FORMATS WITH DIFFERENT NUMBER OF THREADS PER MPI PROCESS, KEEPING 48 THREADS ON EACH NODE

Best result : ELLPACK, 36,1 Gflops, 1 OpenMp per node
CSR , 35.8 Gflops, 1 OpenMp per node
SCOO, 8 Gflops, 1 openMP per CMG
COO, 3.4 Gflops, 1 openMP per node

Ax

ELLPACK

When the pattern is highly irreguar, 1 openMP per CMG is better

threads \ q	0.0		0.4		0.8	
	1	2	1	2	1	2
1	133.3	265.2	12.2	24.4	7.3	14.5
2	132.1	260.0	12.2	24.4	7.3	14.5
4	131.0	261.4	12.3	25.0	7.4	14.9
6	130.8	254.4	12.4	25.3	7.6	15.3
12	131.0	253.9	12.9	26.3	9.0	17.1
24	126.5	255.5	8.6	16.8	7.5	14.1
48	64.5	241.5	3.3	11.0	1.3	5.5
MPI	140.7	279.6	12.0	24.1	7.2	14.4

TABLE VII

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

N = 8 OOO OOO
C =100

A(Ax+x)

threads \ q	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

Scaling the PageRank Algorithm for Very Large Graphs on the Fugaku Supercomputer

Maxence Vandromme[1], Jérôme Gurhem[2], Miwako Tsuji[3], Serge Petiton[1,2], and Mitsuhisa Sato[3]

IEEE ICCS 2022

PageRank on Fugaju

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 nnz per row, from a base of $nnz = 50$

N = 4000 000 (random pattern!!!)
The number of non zero elements per core is the same, as the size of the matrix. The NNZ of each row of the matrices increases with respect to the number of nodes.

4 x MPI per chip (1 per CMG) and OpenMP on CMG
versus
1 x MPI per chip and OpenMP on each node

NNZ per row , on each node : 50

The compiler did’nt success to optimize such computation for the VSE

CSR and ELLPACK better than SCOO

When the matrix pattern is really iregular and/or random, we have to consider distributed computing inside each node

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 nnz per row, from a base of $nnz = 100$

The number of non-zero elements on each node is larger

NNZ per row , on each node : 100

The ratio 1 MPI per node / 1 MPI per CMG increases, for a random pattern (i.e. really irregular communications between the CMG)

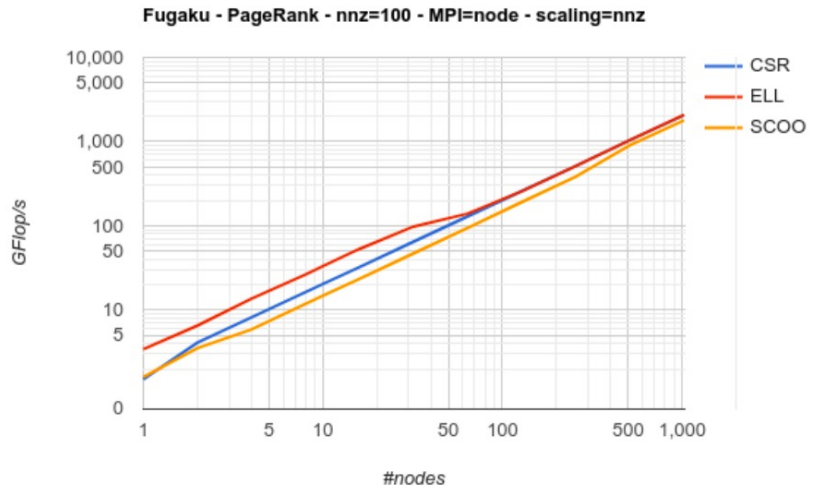
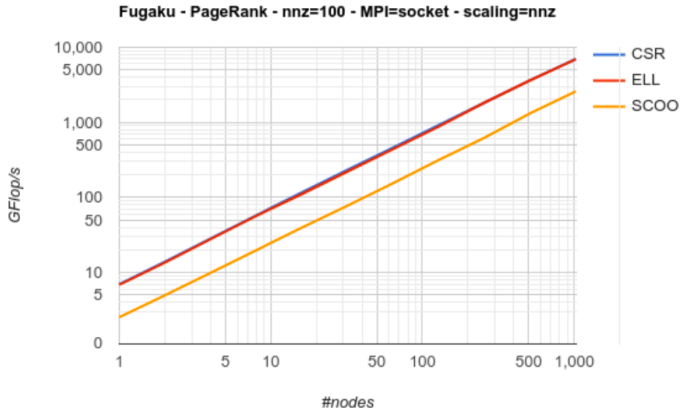


Fig. 3: Median performance for the PageRank, scaling the number of nonzero elements, from a base of $nnz = 100$, with 1 MPI per node



Extrapolation : close to 1 Pflop on Fugaku?

Compare to 16 Pflops for HPCG (diagonals)

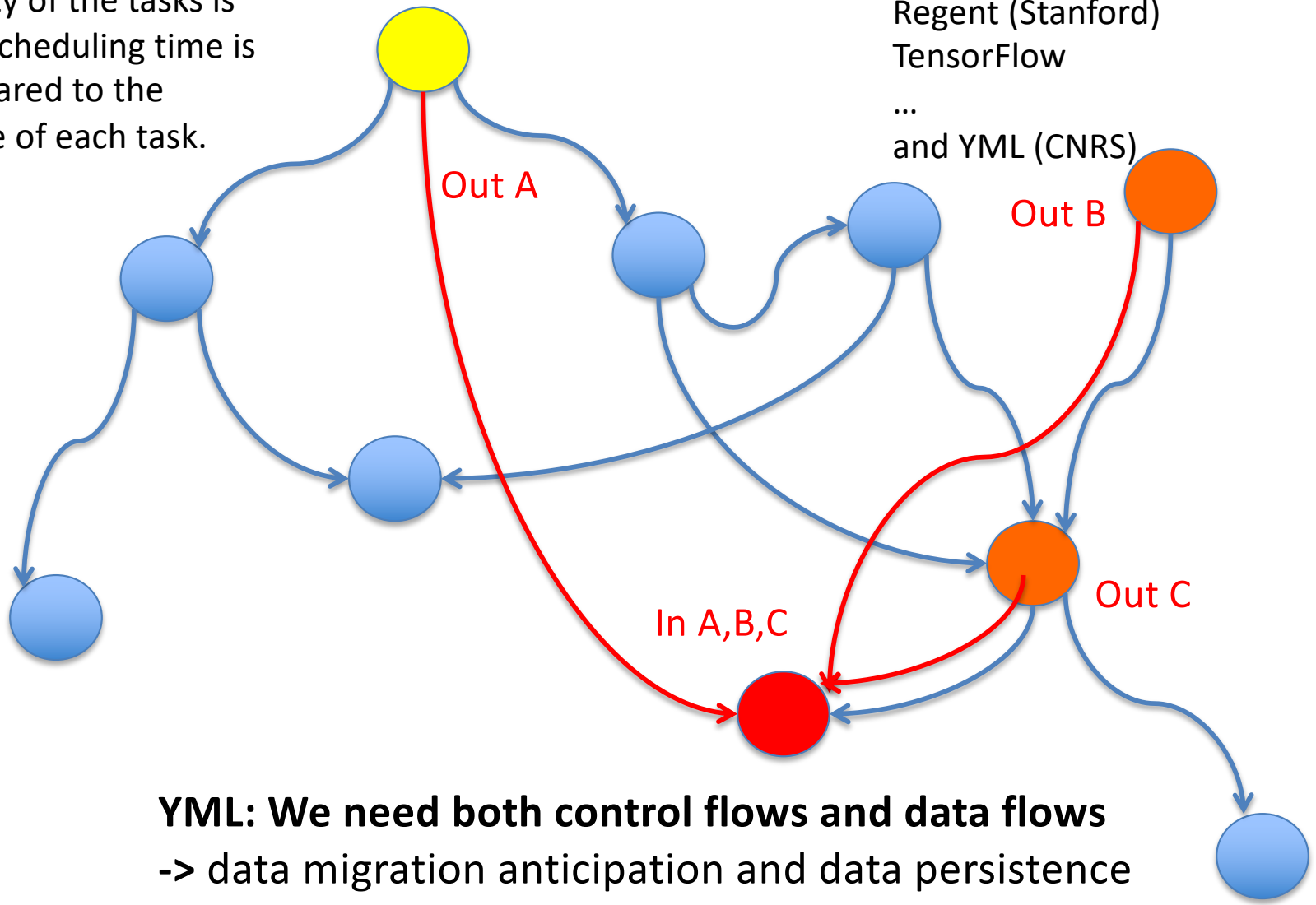
Fig. 4: Median performance for the PageRank, scaling the number of nonzero elements, from a base of $nnz = 100$, with 1 MPI per CMG

2 - Graph of Task programming

Other new level on programming : graph of task programming

If the granularity of the tasks is too small, the scheduling time is too large compared to the computing time of each task.

CONDOR (U. of Wisconsin)
Parsec (ONRL)
Pegasus (USC)
HPX (Louisian U.)
Legion (Stanford)
Regent (Stanford)
TensorFlow
…
and YML (CNRS)



YML: We need both control flows and data flows
-> data migration anticipation and data persistence

YML (since 2000 at CNRS)
(opensource, Cecil Licence)

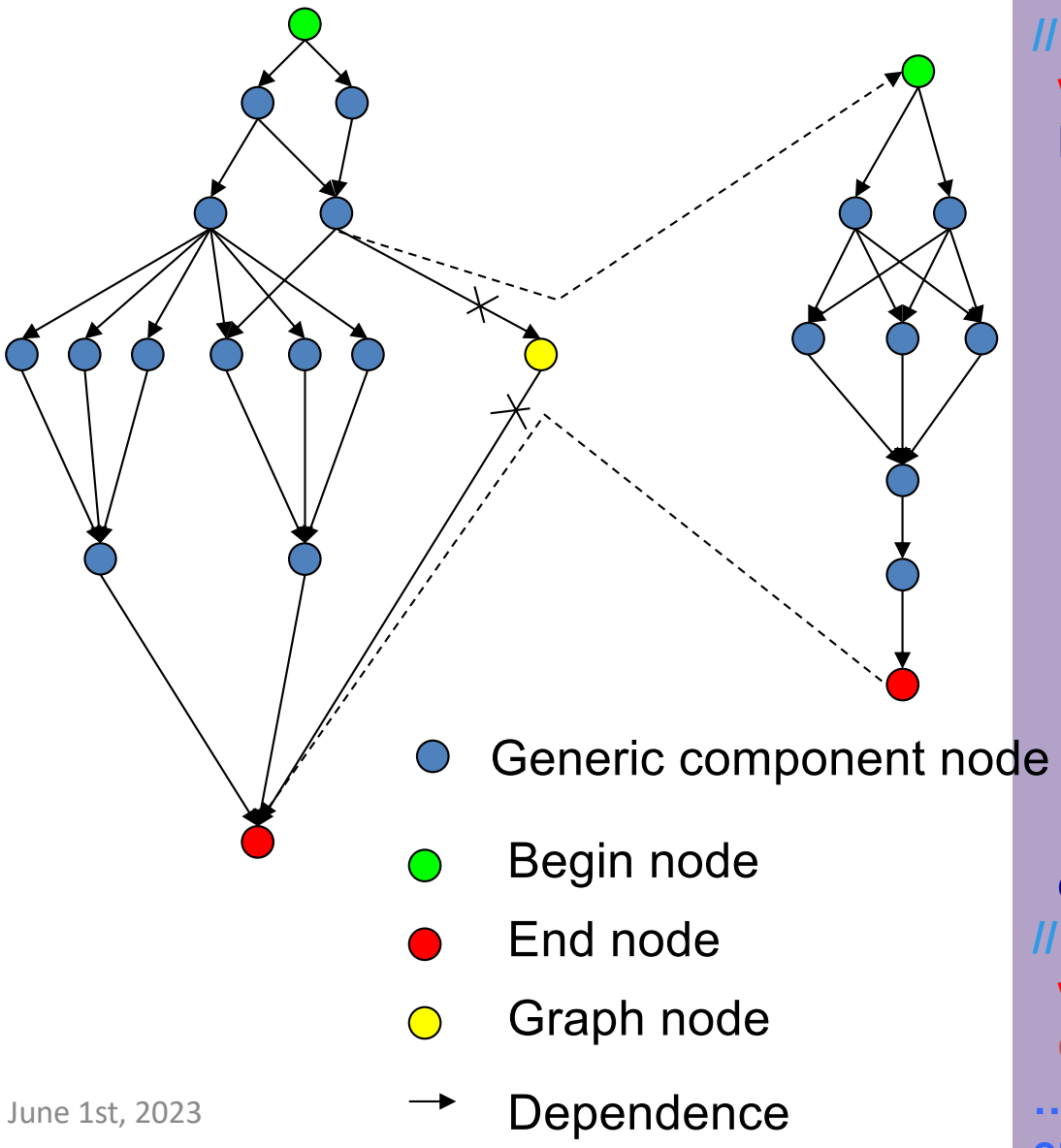
Collaborations with Nahid Emad, Miwako Tsuji, Maxime Hugues,
Laurent Choy, Jérôme Gurhem, Mitsuhisa Sato, et al
experimented on supercomputers, grids and P2P worldwide platforms

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

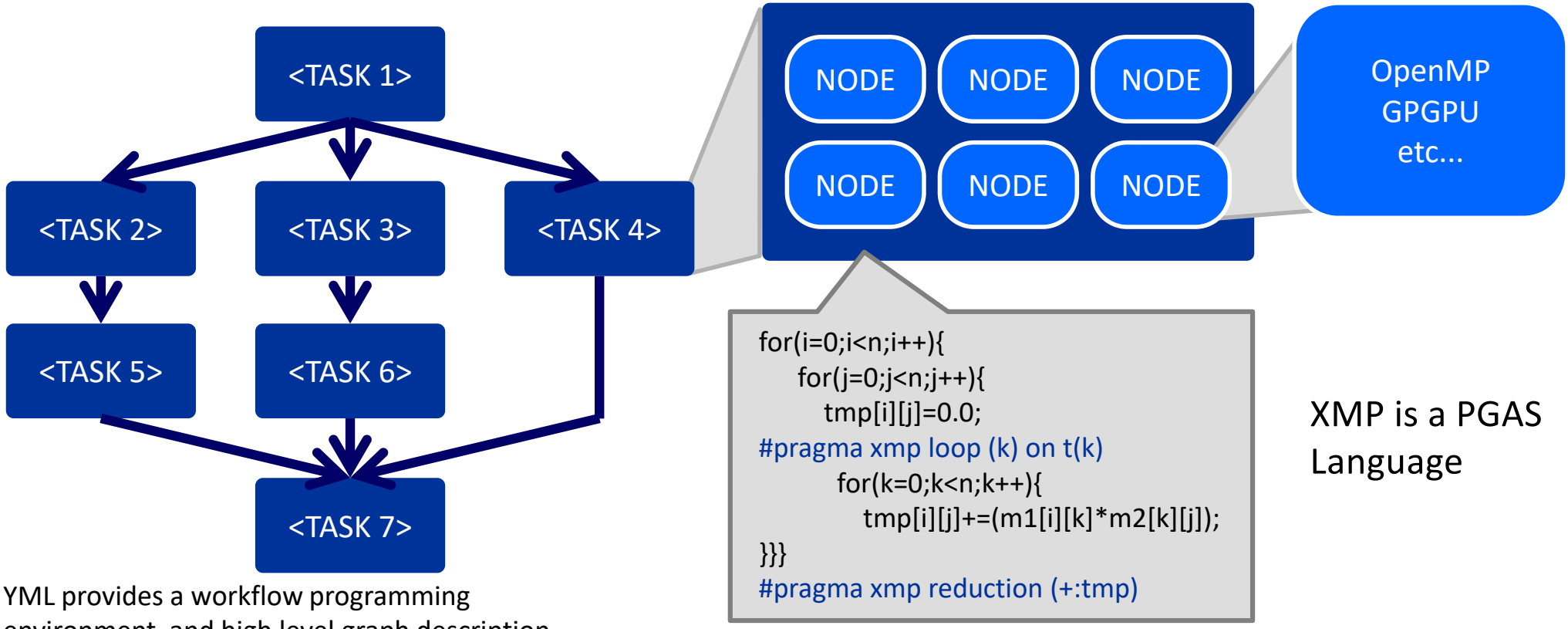
Graph (n dimensions) of components/tasksYML




```
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

XMP is a PGAS Language

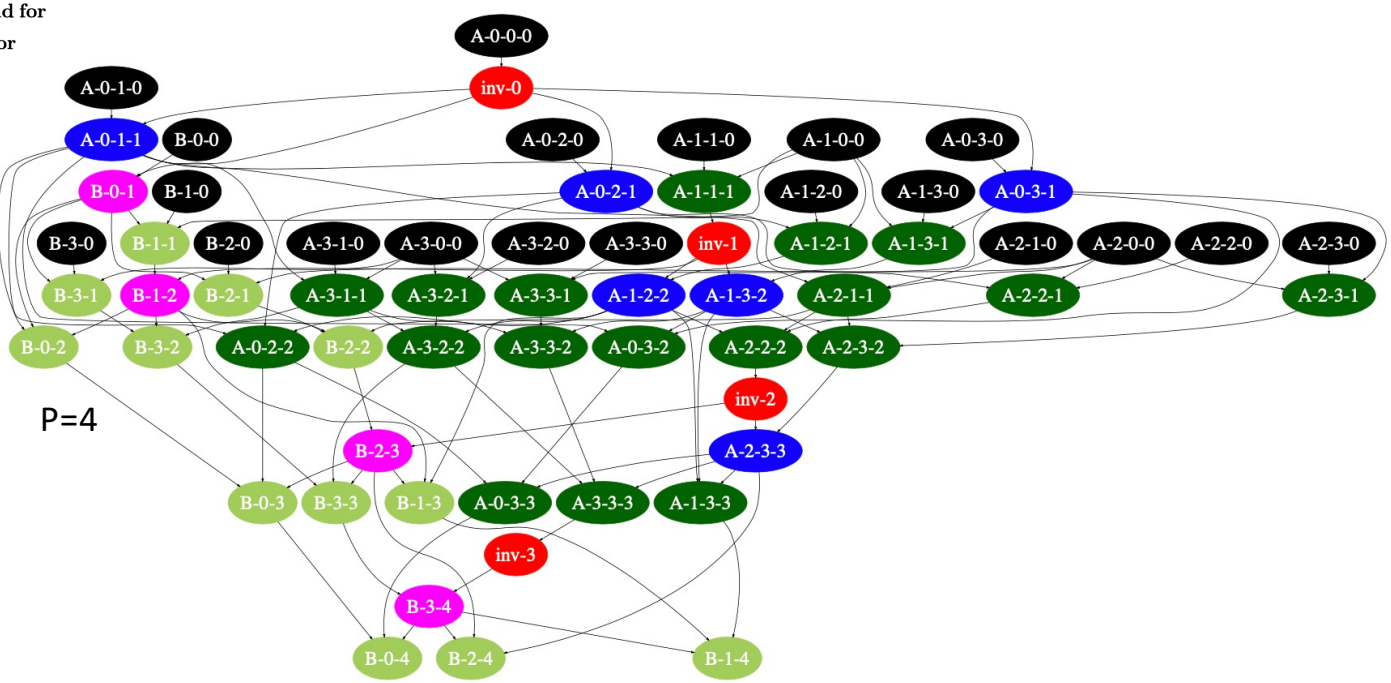
Each task is a parallel program over several nodes.
XMP language can be used to descript parallel program easily!

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

GJ to solve a linear system
Jerome Gurhem's PhD thesis

For k from 0 to $p\text{-}1$ do
- (1) $Inv^{(k)} = [A_{k,k}^{(k)}]^{-1}$
- (2) $b_k^{(k+1)} = Inv^{(k)} \cdot b_k^{(k)}$
- For j from $k\text{+}1$ to $p\text{-}1$ do
  - (3) $A_{k,j}^{(k+1)} = Inv^{(k)} \cdot A_{k,j}^{(k)}$
  - For i from 0 to $p\text{-}1$ do
    - If $k \neq i$ then
      - (4) $A_{i,j}^{(k+1)} = A_{i,j}^{(k)} - A_{i,k}^{(k)} \cdot A_{k,j}^{(k+1)}$
    - End if
  - End for
- End for
- For i from 0 to $p\text{-}1$ do
  - If $k \neq i$ then
    - (5) $b_i^{(k+1)} = b_i^{(k+1)} - A_{i,k}^{(k)} \cdot b_k^{(k+1)}$
  - End if
- End for

End for



Collaboration avec TOTAL-Pau et Houston

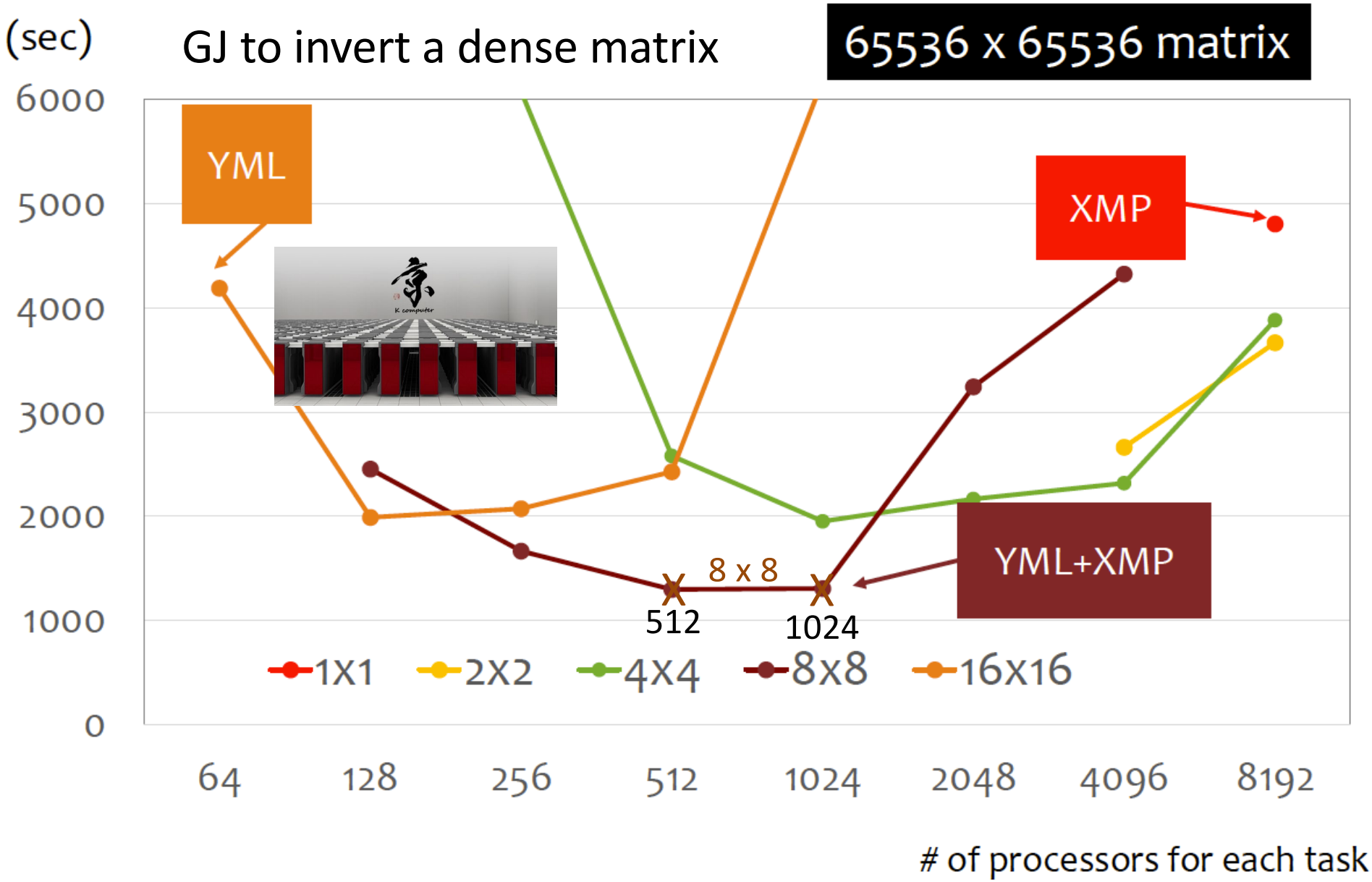
Block Linear Systems (Dense matrices)

We may compare with MPI or others but the most interesting to evaluate YML in these eexperiments is to compare YML-XMP with XMP alone

	Size	YML/XMP			XMP
		blocks	cores/task	best time	
Gaussian elimination	32768	4×4	512	276.8	508.5
	65536	8×8	512	690	2512
Gauss-Jordan elimination	32768	4×4	512	285.12	615.868
	65536	8×8	512	792.737	2970.393
LU factorization	32768	4×4	512	332.35	505.412
	65536	8×8	512	881.032	2306.163

K computer, 8000 nodes

Experiments (2) BGJ on K-Computer



Miwako TSUJI, RIKEN/AICS

Outline

1. Introduction
2. Programming Paradigms
3. **Methods and Algorithms (Unite&Conquer, PageRank)**
4. HPC and Machine Learning (GCN, Transformer,..)
5. Generators of Data Sets and matrices for brain-scale applicatons
6. Conclusion

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 number of operations for stochastic matrix by a vector products

It is important to study the methods. Using API and librairies without understanding the methods are not enough.

==> multidisciplinary teams

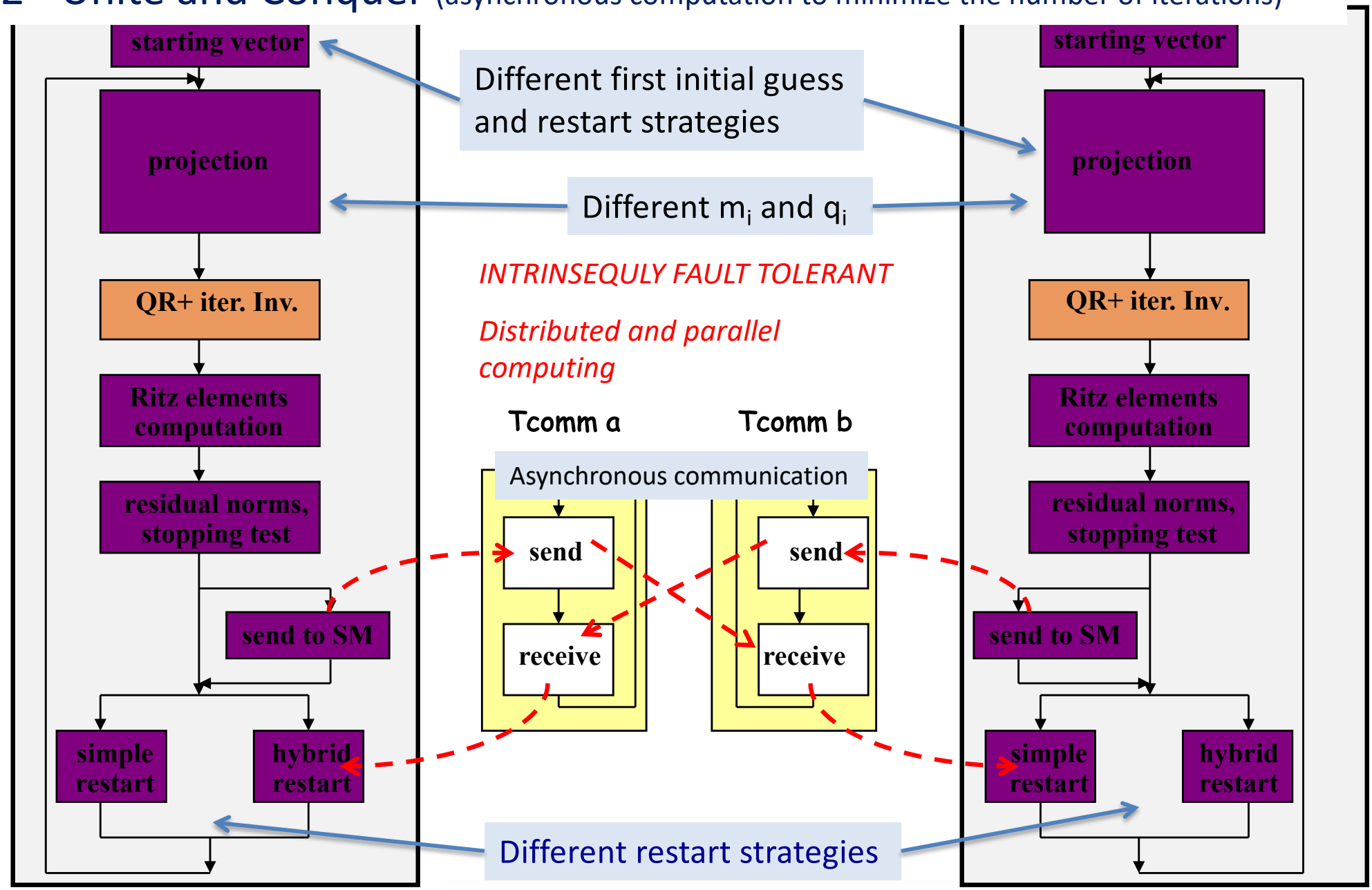
TABLE II: Median runtime (in ms) of all three applications on synth-3, for both storage formats and both variants of SpMV (o = original, n = new) on Ruche

	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**

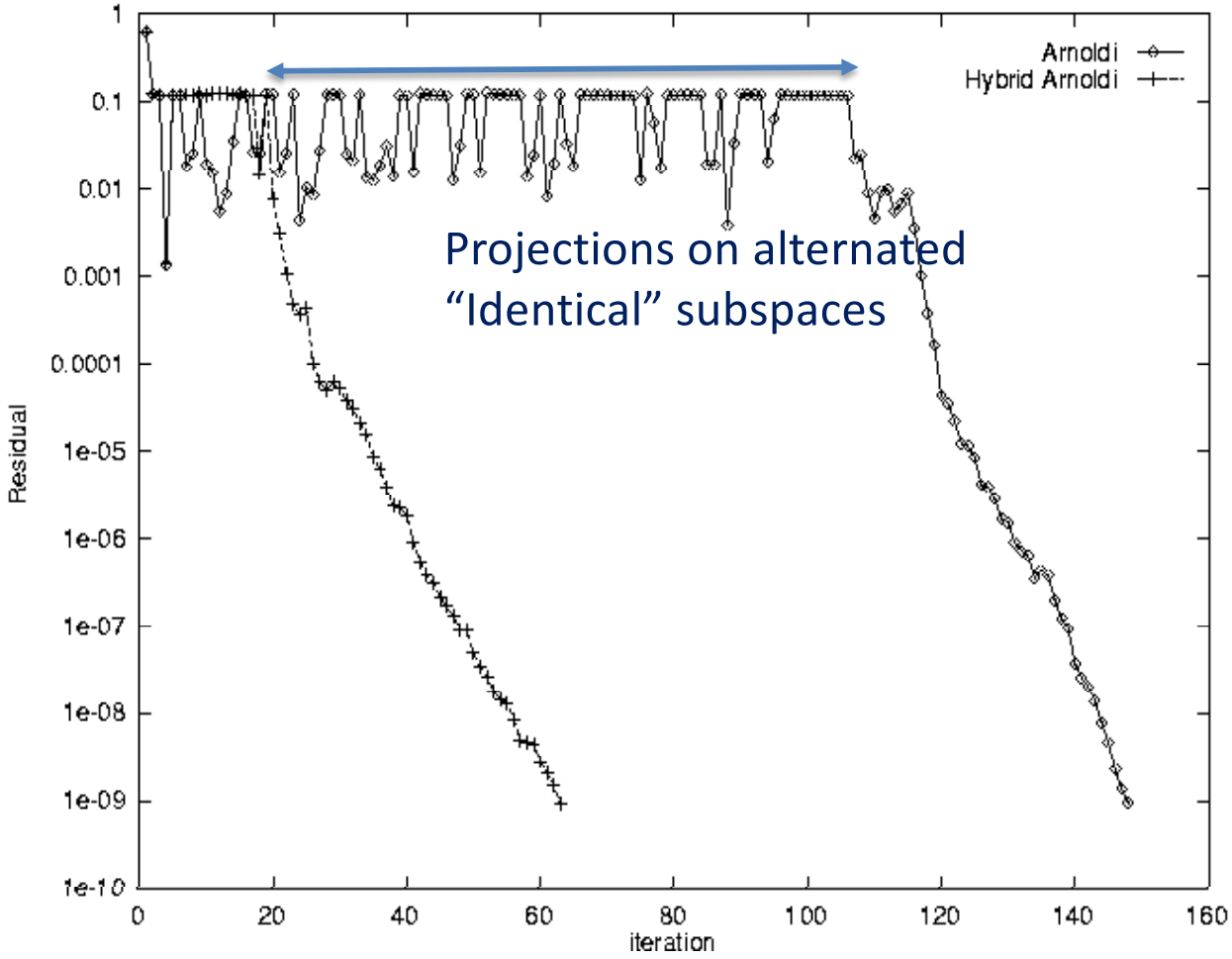
2 - Unite and Conquer (asynchronous computation to minimize the number of iterations)



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**

MERAM, since 1993



U&C Restarded methods
Linear combinaison of
Ritz values and diferent
restart strategies

Guy Edjlali PhD thesis, 1994
France Boillod-Cerneux PhD thesis, 2014

Machine Learning

Ensemble method : diferent
Neural networks in parallel +
optimization

Reiforcement Learning + ensembles for smart grids (with TOTAL).

Pierrick Pochelu PhD thesis, 2022

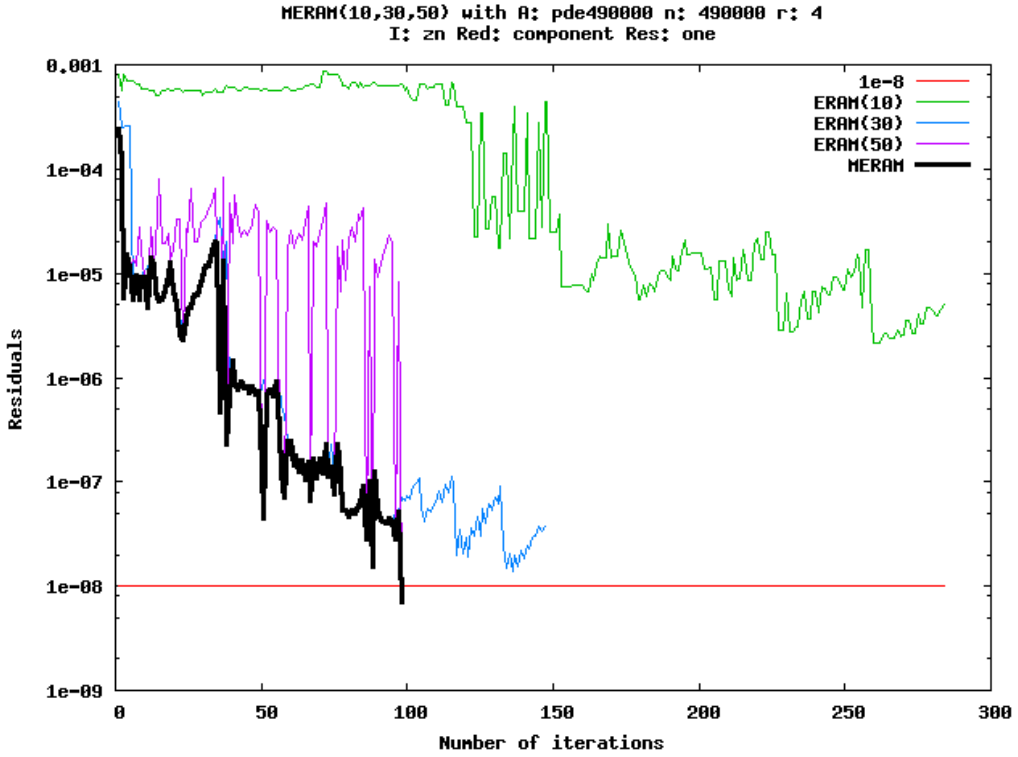
1993 : network of one CM200 and one CM5 and a few SUN workstation

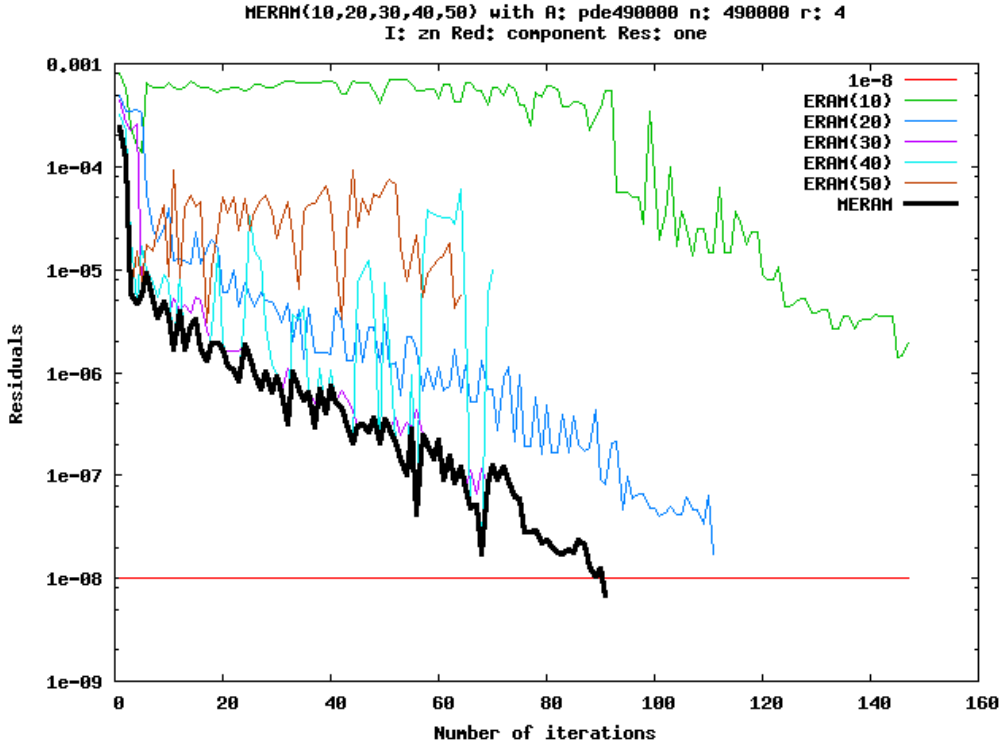
Journal of Computational Science
Volume 14, May 2016, Pages 5-14

Experiments on Grid'5000

Unite and conquer approach for high scale numerical computing 2016

Nahid Emad [a b], Serge Petiton [a c]





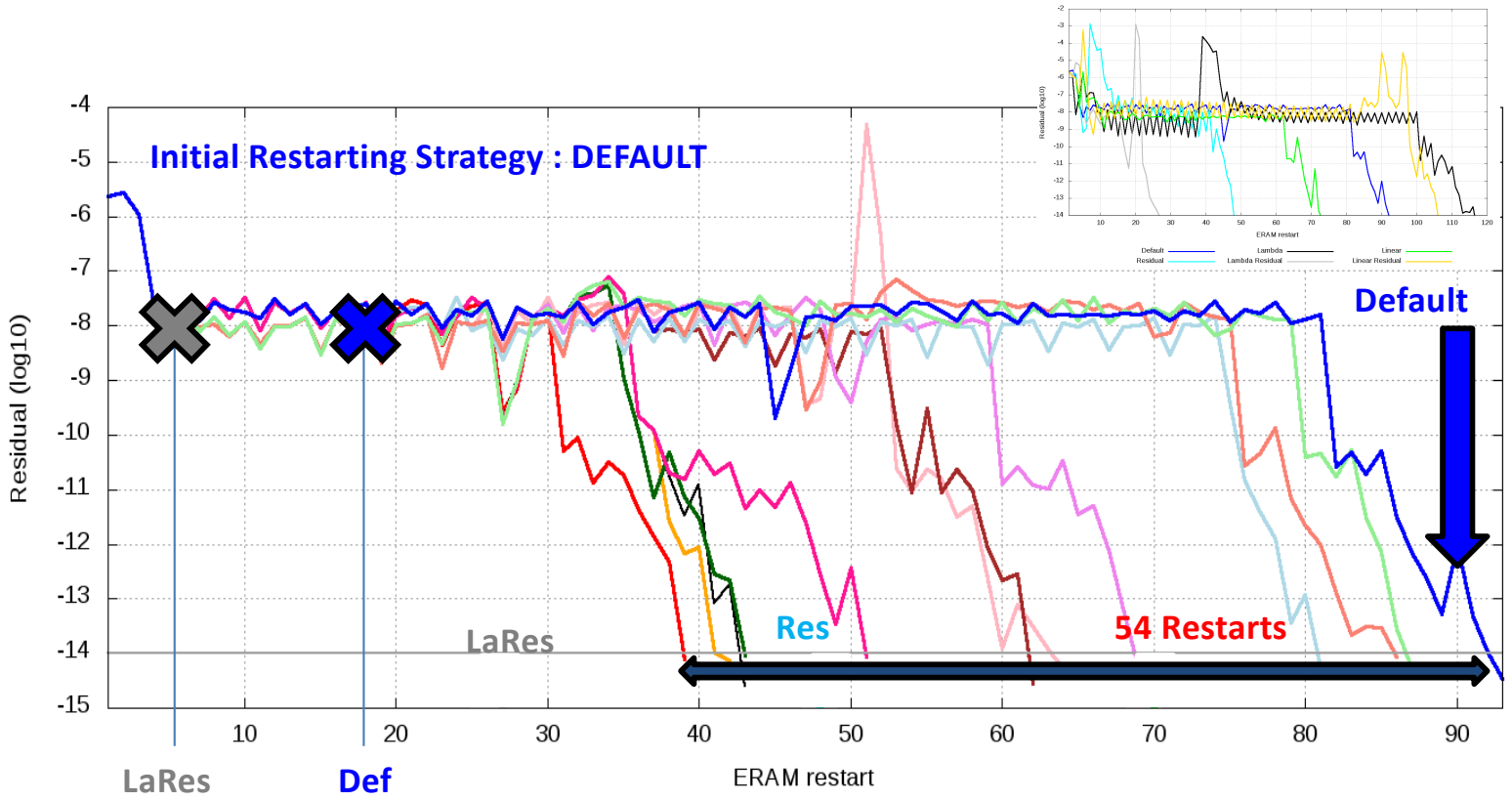
Convergence of MERAM for matrix 490000 with different number of co-methods

Experiments on several supercomputers : Maspar, IBM SP series, Hooper (Berkeley), P2P platforms (using YML),…

U&C MIRAM, CuSparse
Alexandre Feder, collaboration avec Nvida Santa Clara

ERAM restarting strategies mix

France Boillod-Cerneux PhD



4 eigenpairs,
m=15, CGSr
Bayer04 Matrix
- N=20545
- nnz=85537
- 1 Intel i5-2430M

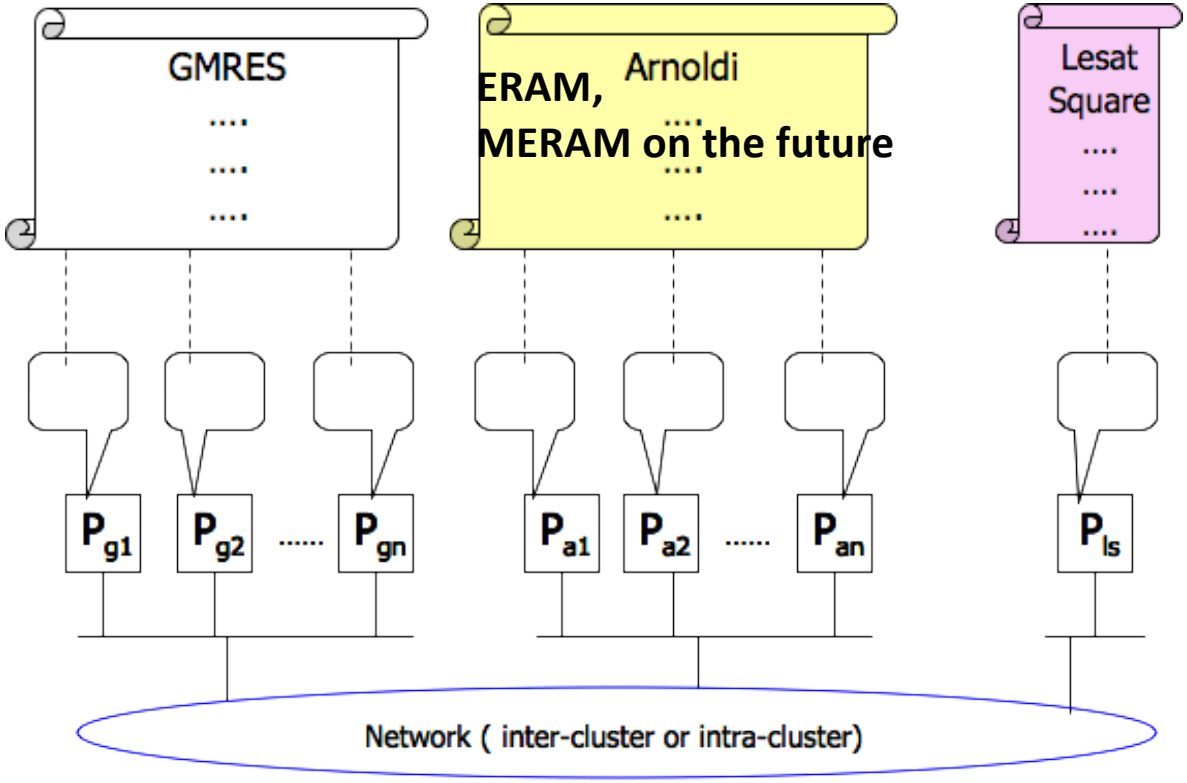
LaRes(5), Def(18)

Li(38)
Res(26)
Res(14), LaRes(24)
LaRes(4), Def(30)
LaRes(5), Def(21)
DEFAULT

Asynchronous Iterative Restarted Methods

Collaboration with He Haiwu and Guy Bergère (U. Lille 1, CNRS)
and Ye Zhang (Hohai Univ. Nanjing, China) , Salim Nahi (Maison de la simulation, Saclay),
and Pierre-Yves Aquilenti (TOTAL Pau and Houston), Takahiro Katagari,5U. Tokyo),
Xinzhe Wu (CNRS, Saclay).

U@C GMRES-LS/ERAM(UCGLE) method



ELSEVIER
Available online at www.sciencedirect.com
SCIENCE DIRECT
Computers and Mathematics with Applications 51 (2006) 1647–1662
www.elsevier.com/locate/camwa
An International Journal computers & mathematics with applications

A Hybrid GMRES/LS-Arnoldi Method to Accelerate the Parallel Solution of Linear Systems

HAIWU HE, G. BERGERE AND S. PETITON

To solve a linear system, we asynchronously compute the eigenvalues to accelerate the convergence, using a least square polynomial acceleration.

A Smart Tuning Strategy for Restart Frequency of GMRES(*m*) with Hierarchical Cache Sizes

Takahiro Katagiri[1], Pierre-Yves Aquilanti[2,3], Serge Petiton[4],

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

It is a fault tolerent method, as U&C MERAM

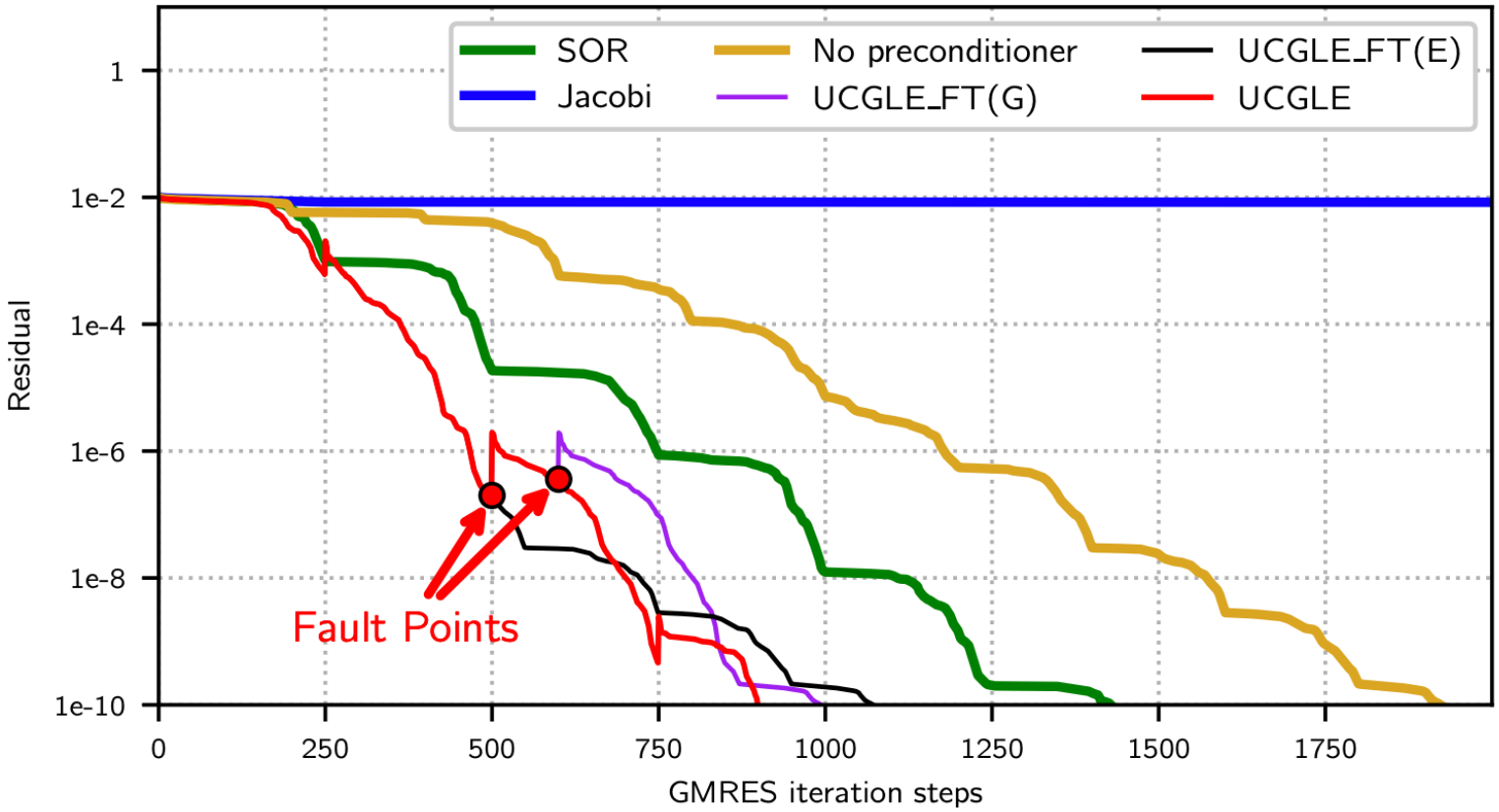
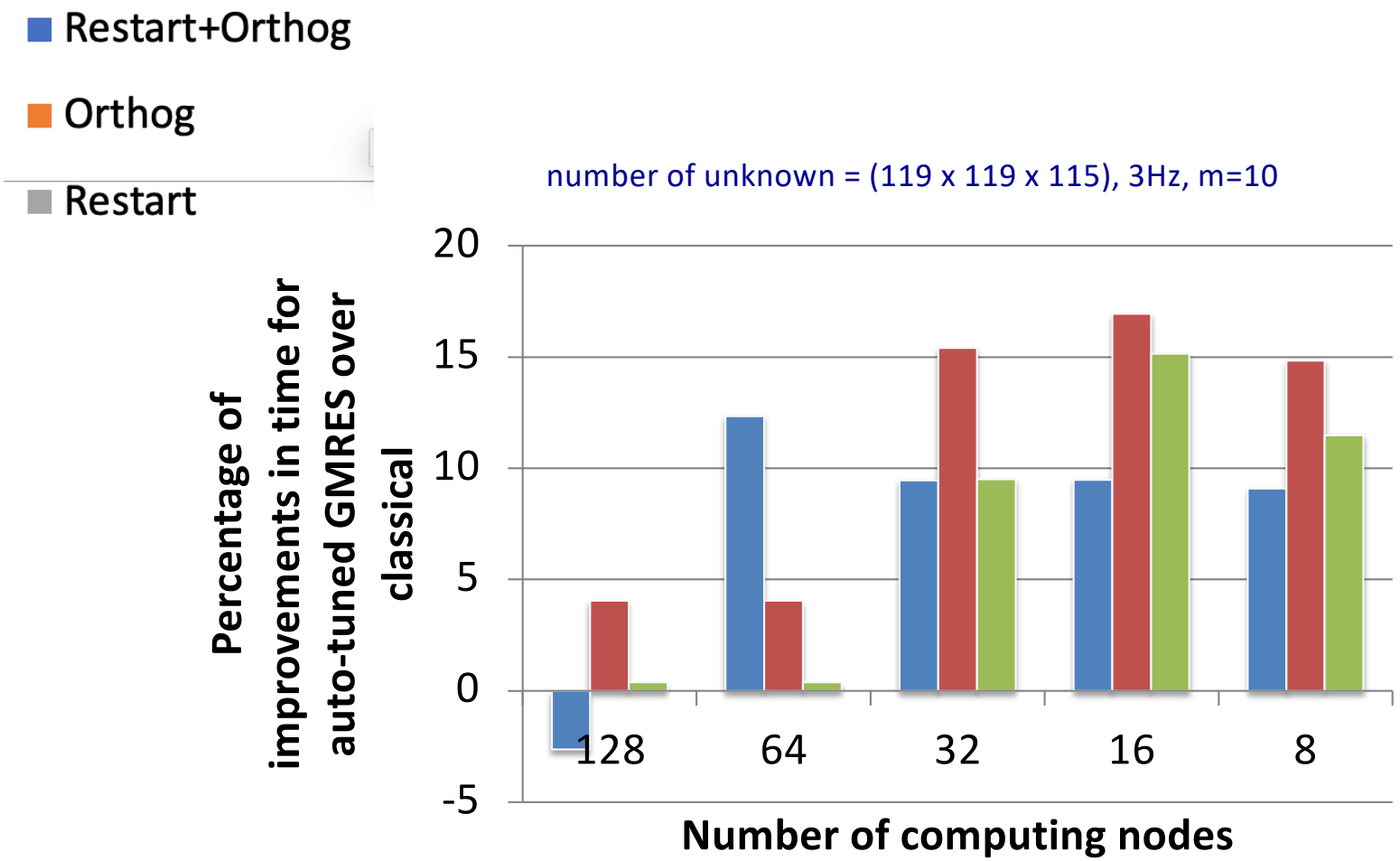


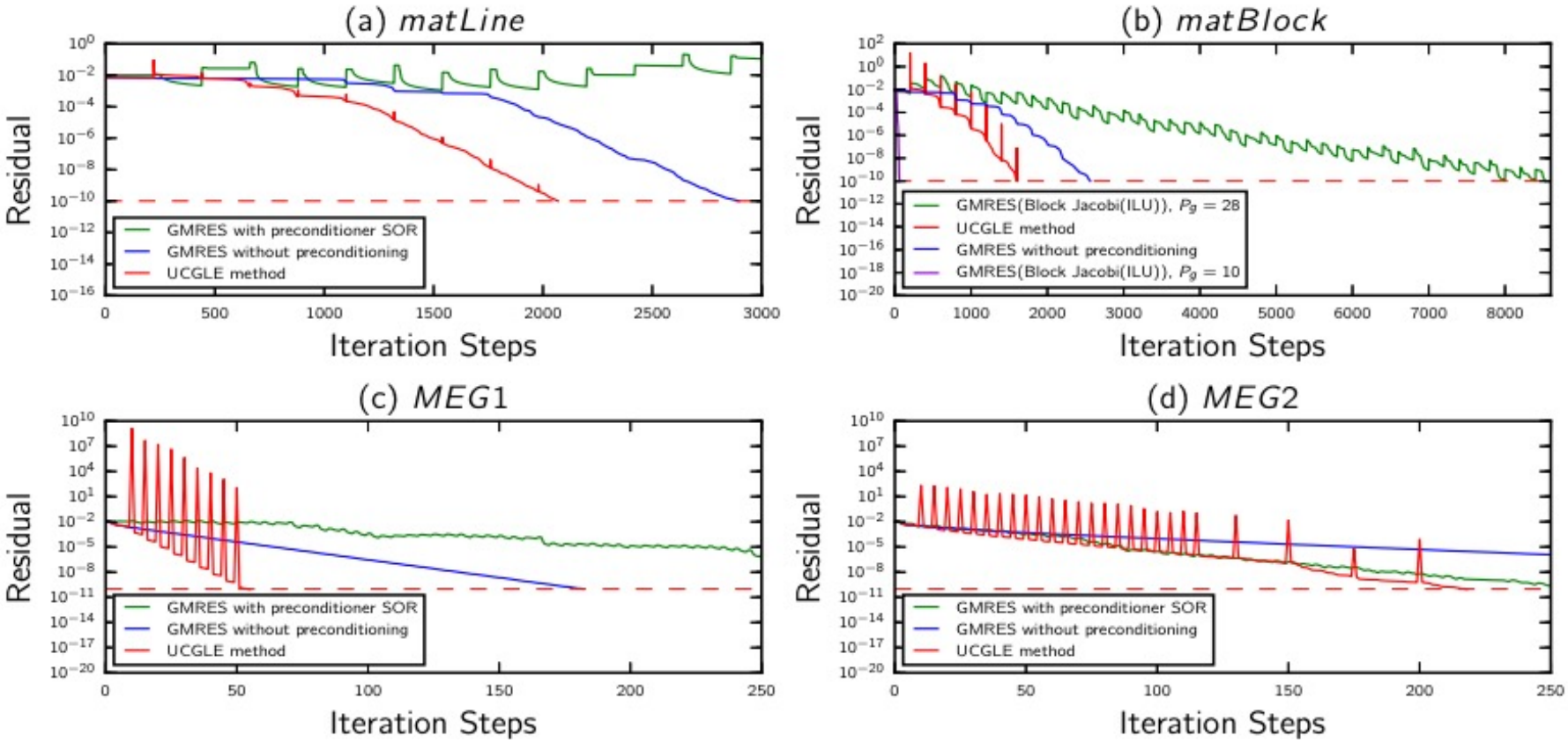
Figure 5.19 – $MEG1$: convergence comparison of UCGLE method vs conventional GMRES

Smart tunning of hyperparameters (subspace size, number or vectors orthogonalized)



Results : Industrial Case
Collaboration with TotalEnergies

On Thianhe 2



U&C and ensemble methods are well-adapted for distributed and parallel computing
They are well-adpated to YML-XMP like programming

Outline

1. Introduction
2. Programming Paradigms
3. Methods and Algorithms (Unite&Conquer, PageRank)
4. **HPC and Machine Learning (GCN, Transformer,..)**
5. Generators of Data Sets and matrices for brain-scale applicatons
6. Conclusion

Parallel Jaccard and Related Graph Clustering Techniques

Collaboration avec
Nvidia Santa Clara

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
Maisc
Universit
serge.p

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 $V = S_1 \cup S_2 ... \cup S_p$ 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

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

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.

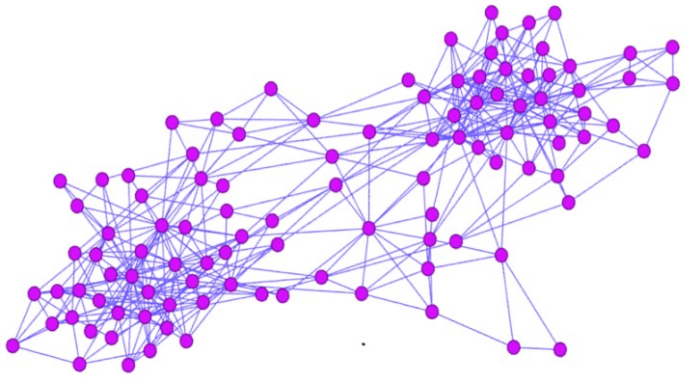


Figure 1: Amazon book co-purchasing original graph

Figure 2: Amazon book co-purchasing graph with Jaccard

Graph Convolutional Network (GCN)

Let A be the **adjacency matrix** of a given graph $(\mathcal{V}, \mathcal{E})$, with N nodes. A is a N x N sparse matrix.

Let $D_{i,i} = \sum_{j=1,N} A_{i,j}$ be the **degree matrix** (D a main diagonal N x N matrix).

The **un-normalized Laplacian matrix** of the graph $(\mathcal{V}, \mathcal{E})$ Graph is L = D – A

Let $D^{1/2}$ (resp. $D^{-1/2}$) be the diagonal matrix with the (resp. inverse of the) square root of the elements of D

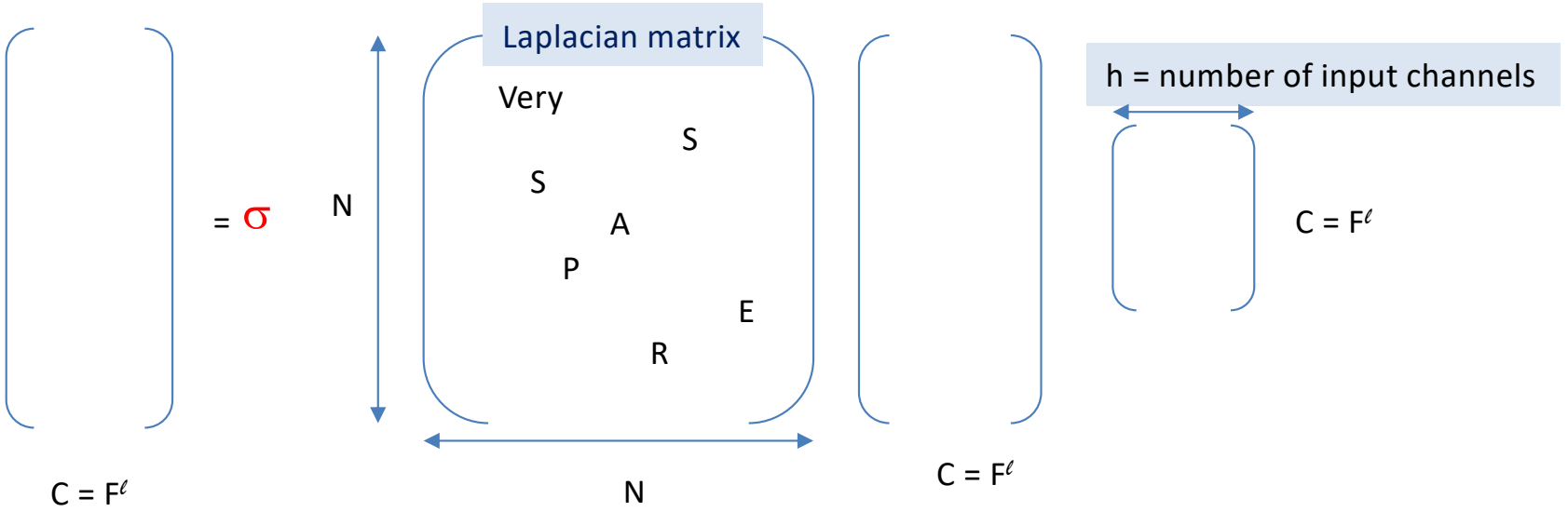
$L_N = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$ is a **normalized Laplacian matrix**

$L_N =$

if i equal j and if the degree of the node i=j is not 0
then **1**.
if i is different of j and if the degrees of the node i and the node j are not zero
then **-1 / sqrt(degree(node i) degree(node j))**
else
0

It exists other normalized Laplacian matrices, such as the Random walk normalized Laplacian matrix :
$L_{RW} = I – P$, where $P = D^{-1}A$ is the transition matrix of a random walker on the graph. This matrix is normalized.

Sequence of Sparse Dense Dense matrix products, plus softmax and other operators



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

Over-smoothing

Over-smoothing : high power calculation of $\tilde{A}$

We have an important over-smoothing when L increase, associated wit the over-smoothing of powers of the Laplacien matrix.

After L steps of $H^{(\ell+1)} = \sigma (\tilde{A} H^{(\ell)} W^{(\ell)})$

We have $H^{(\ell+1)} = \sigma (\tilde{A} (\sigma (\tilde{A} H^{(\ell-1)} W^{(\ell-1)})) W^{(l)})$
$= \sigma (\tilde{A} (\ldots\ldots \sigma \tilde{A} H^{(0)} W^{(0)}) \ldots. W^{(l)})$

We have to control the smoothing.

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

How to limit this smoothing?

DropEdge method : randomly ”drop” edges
DropNode method : randomly drop nodes

We propose to first rank the nodes

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

Collaboration avec Huawei-Paris

Random selection of ranked data.

We first rank the nodes of the graph, using the PageRank method, then, we order the nodes with respect to this ranking, and we compute a scan with add of the resulting vector.

Then, we randomly select a real number between 0 and 1, and we find the node with the closer ranking to this value.

The probabilty to be randomly selected is higher if the PR ranking is higher.

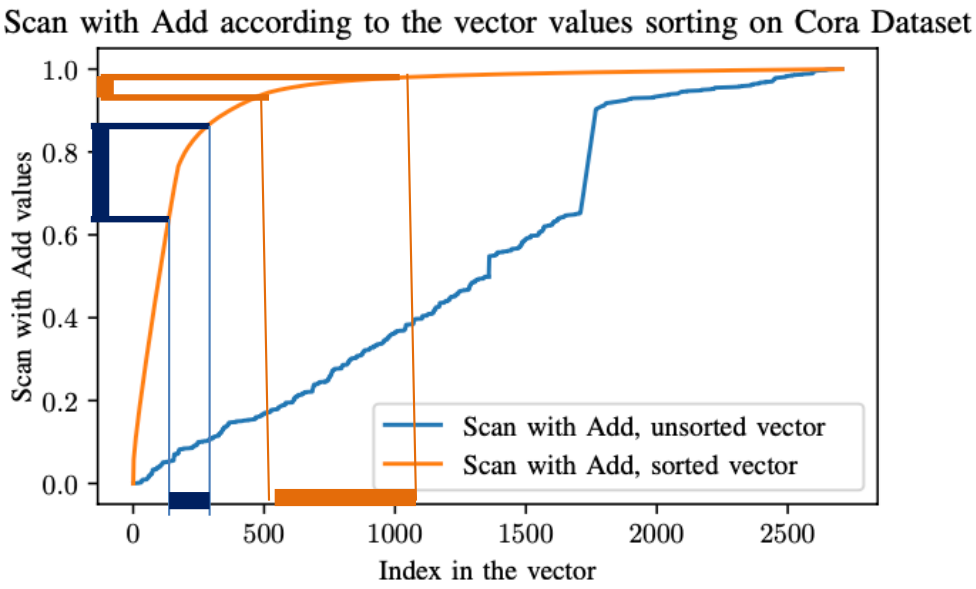


TABLE I: Datasets global information

	Cora	*Citeseer*	*Pubmed*
NB OF NODES	2 707	3 327	19 717
NB OF EDGES	5 429	4 732	44 338
EDGES/NODE (MEAN)	2.01	1.42	2.25
MAX NODE DEGREE	168	99	171
MAX VAL PR	0.0492	0.0401	0.00610

Fig. 2: Scan with Add vector values for the Cora dataset. The orange curve represents the values when the final score values are sorted in descending order before applying the Scan with Add.

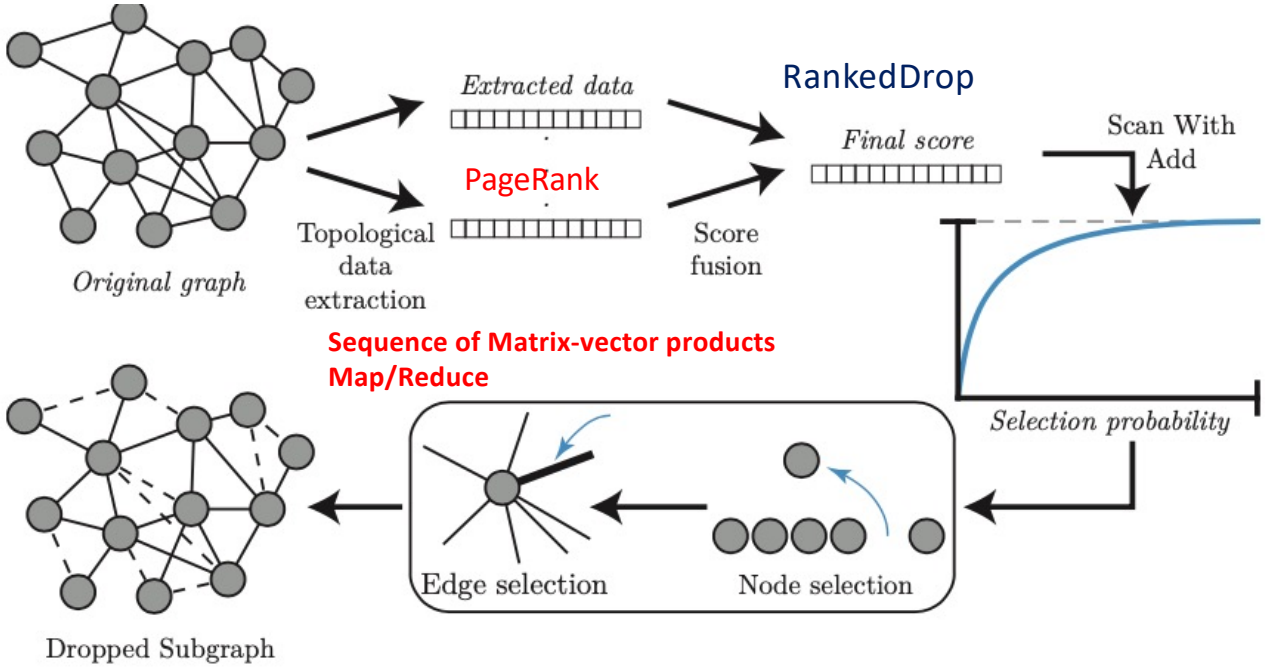
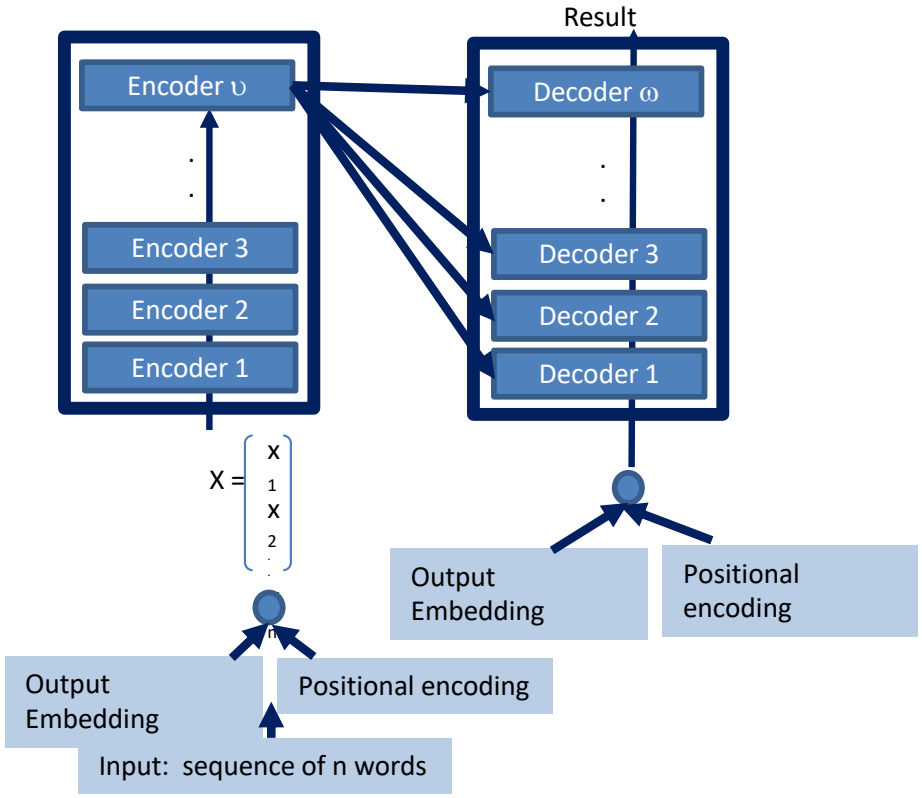


TABLE II: Accuracy comparison for semi-supervised learning methods for GCN architecure

DATASET	LAYERS	ORIGINAL	DE	RD
CORA	2	81.10	82.80	82.90
	4	78.50	78.80	82.00
	8	31.10	53.10	63.90
CITESEER	2	70.80	72.30	73.20
	4	61.20	68.80	71.30
	8	30.20	33.20	45.50
PUBMED	2	79.00	79.60	79.90
	4	78.30	77.70	79.40
	8	61.20	54.50	77.10

Transformer method

Collaboration with Huawei-Paris



BERT, GPT,..

Attention :

$$\left(\sigma\ \frac{QK}{\sqrt{d_k}}\right) V$$

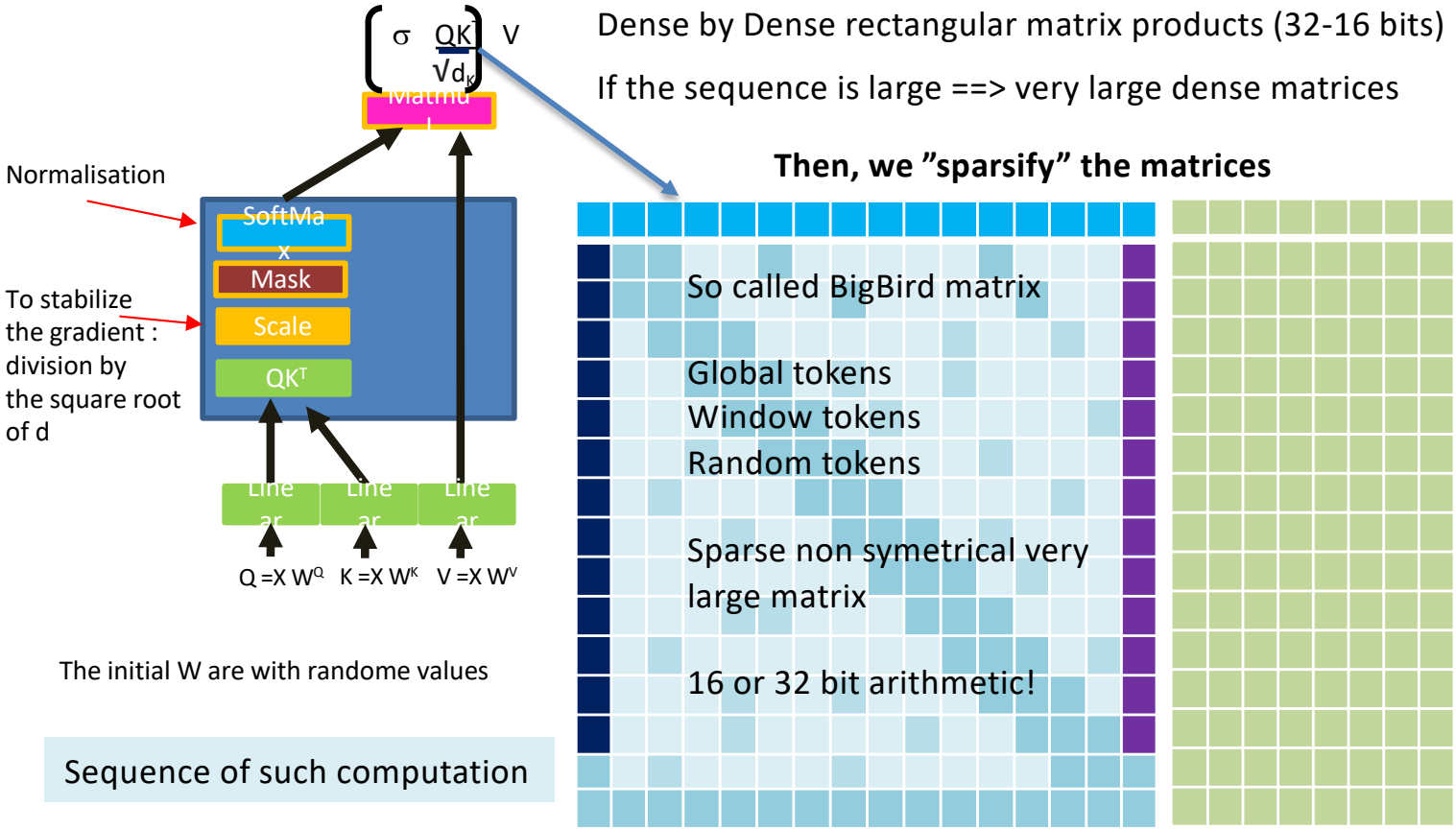
Mathu

Sequence of dense by dense rectangular matrix products

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

Dense case

Attention for very large sequence



Goal : a distributed and parallel transformer algorithm for the “BigBird” transformer method on large cluster-supercomputers or clouds, using mixed arithmetics (and YML-PGAS-like programmig paradigms).

Having a computational science approach for this machine learning method using very large data (matrix-linear algebra, sparse data structure,…)

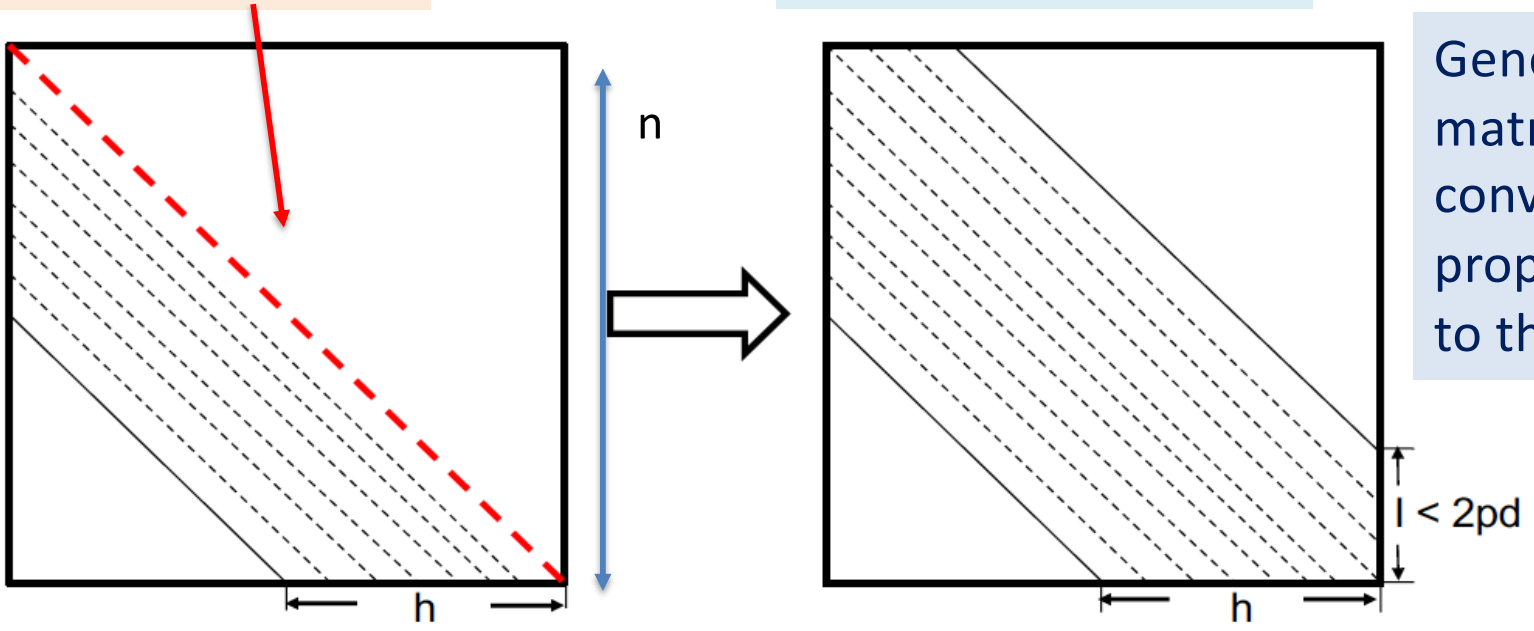
Outline

1. Introduction
2. Programming Paradigms
3. Methods and Algorithms (Unite&Conquer, PageRank)
4. HPC and Machine Learning (GCN, Transformer,..)
5. **Generators of Data Sets and matrices for brain-scale applicatons**
6. Conclusion

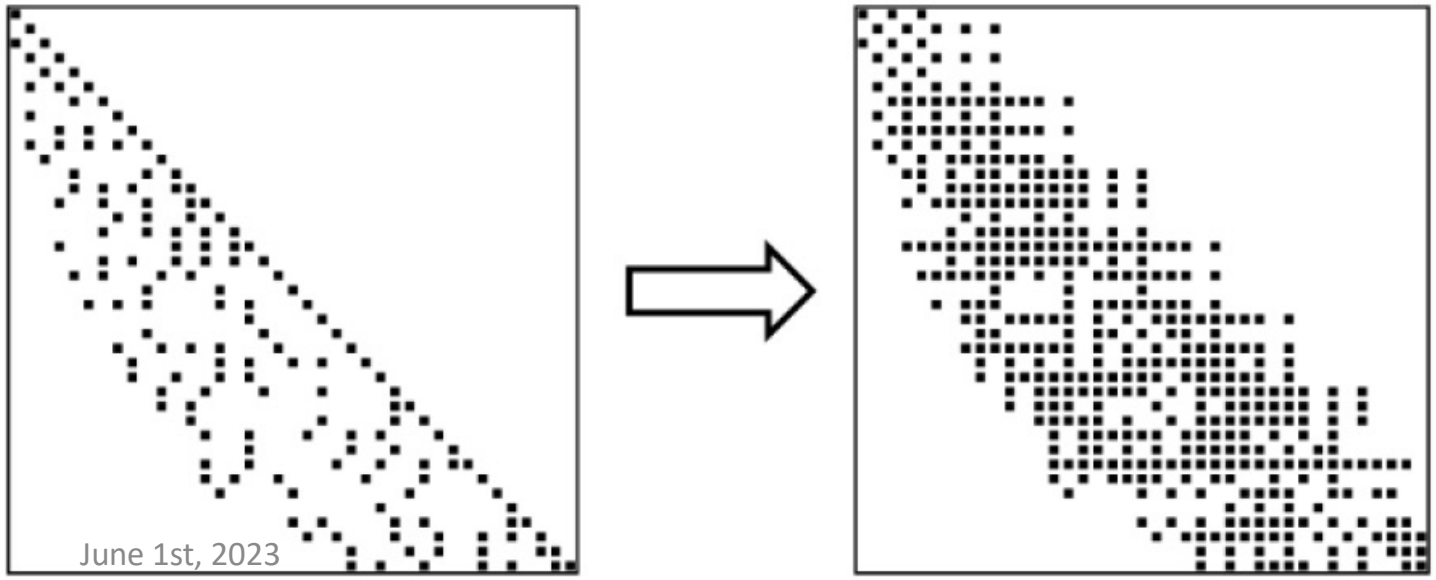
1 - Generator of non-Hermitian matrices, from given spectrum

given spectrum

Generated matrix



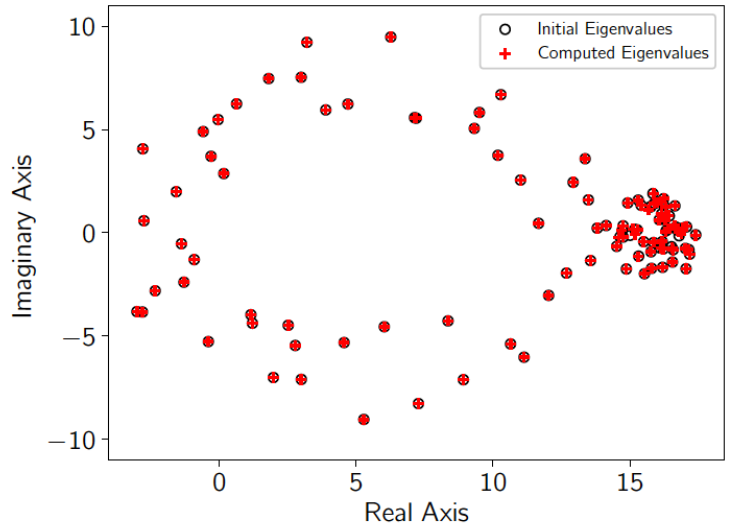
Generate sparse matrices to evalaute convergence and other properties with respcet to the spectrum



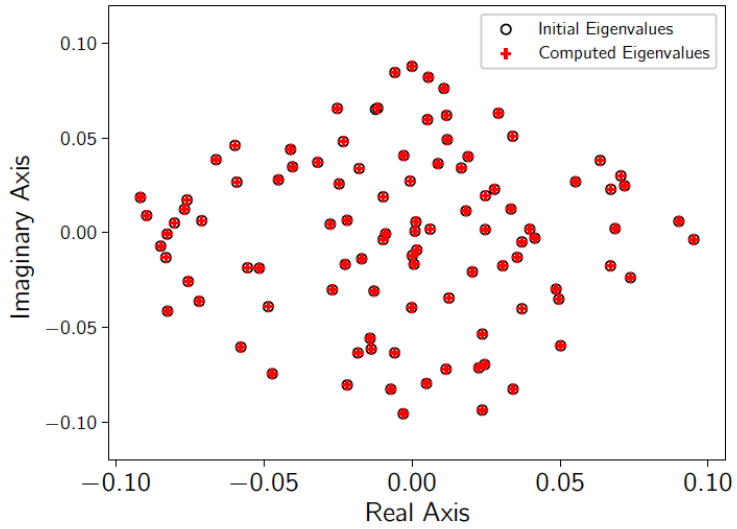
Directly compute in parallel.

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

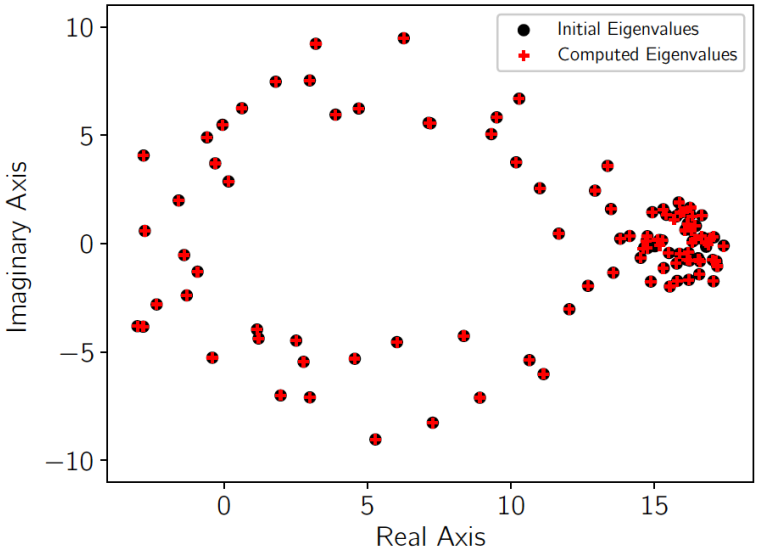
Example



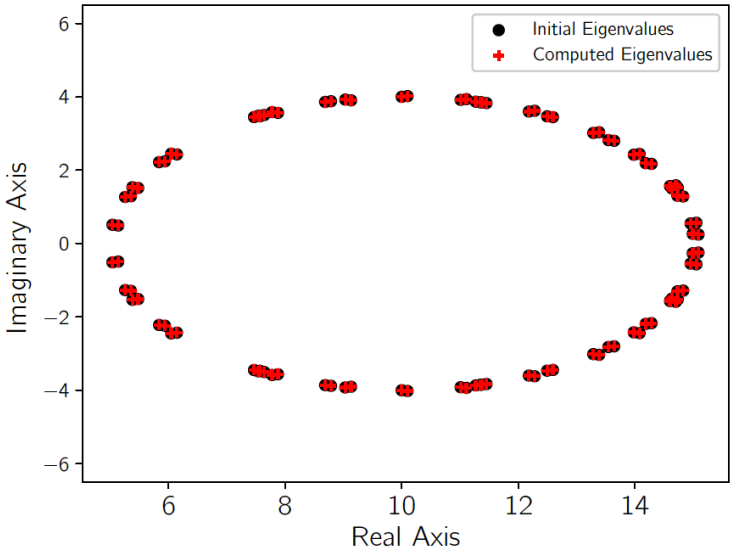
(a) Dominant Clustered Eigenvalues: acceptance = 94%, max *error* = 3×10^{-2}



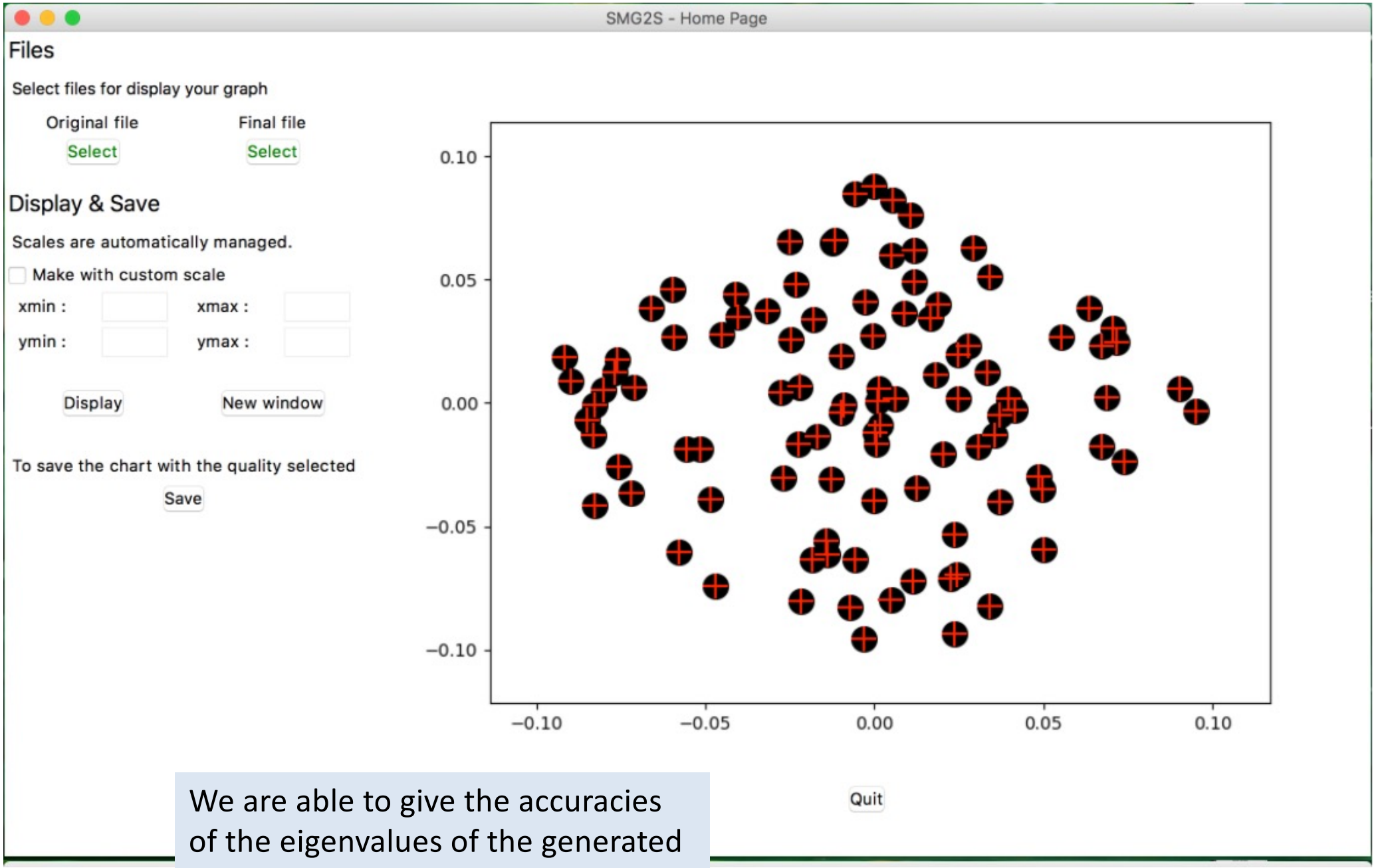
(b) Clustered Eigenvalues: acceptance = 100%, max *error* = 7×10^{-5}



(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

SMG2S

SMG2S: Scalable Matrix Generator with Given Spectrum

Jülich, Germany https://smg2s.github.io

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

Overview | Repositories 7 | Projects | Packages | People 1

Pinned

SMG2S Public
Scalable Matrix Generator with Given Spectrum
C++ ☆ 5 2

BTIDG2 Public
Brain Topology Inspired Distributed Graph Generator
C

Smg2s.jl Public
A julia implementation of SMG2S (Sparse Matrix Generator with Given Spectrum)
Julia

DEMAGIS Public
C++

People

Top languages

C++ C Julia TeX HTML

Website: https://smg2s.github.io

3

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

Repositories

Find a repository...

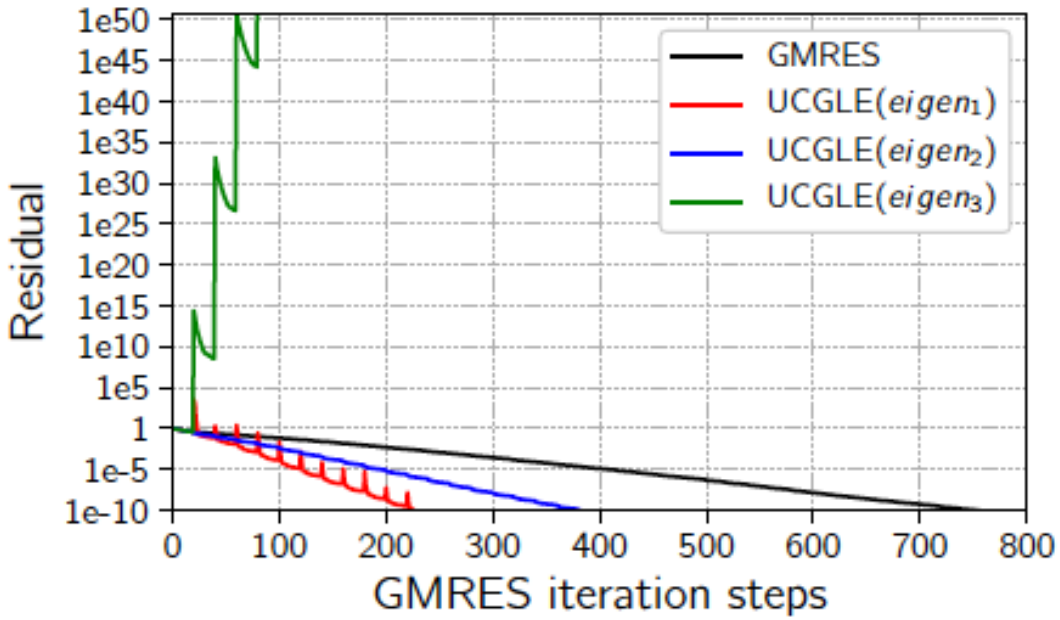
BTIDG2 Public
Brain Topology Inspired Distributed Graph Generator
C ☆ 0 MIT 0 0 0 Updated last week

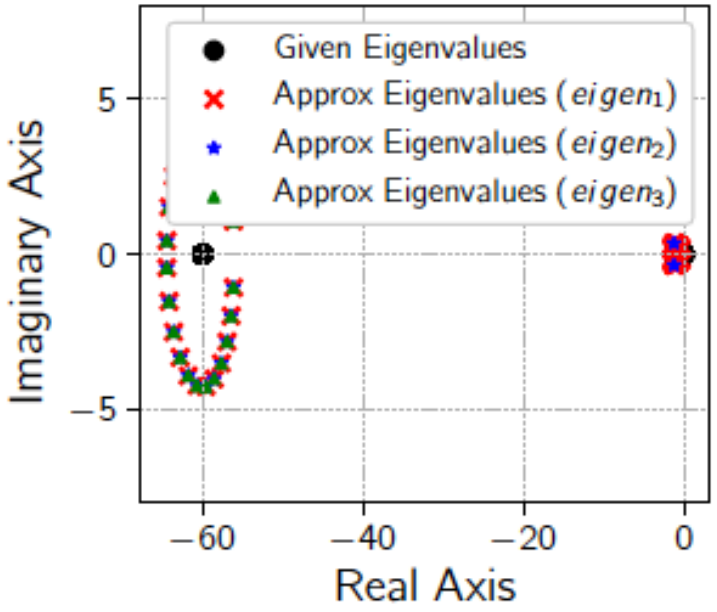
SMG2S Public
Scalable Matrix Generator with Given Spectrum
C++ ☆ 5 MIT 2 0 0 Updated on Jan 11

Smg2s.jl Public
A julia implementation of SMG2S (Sparse Matrix Generator with Given Spectrum)

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

Thianhe 2A





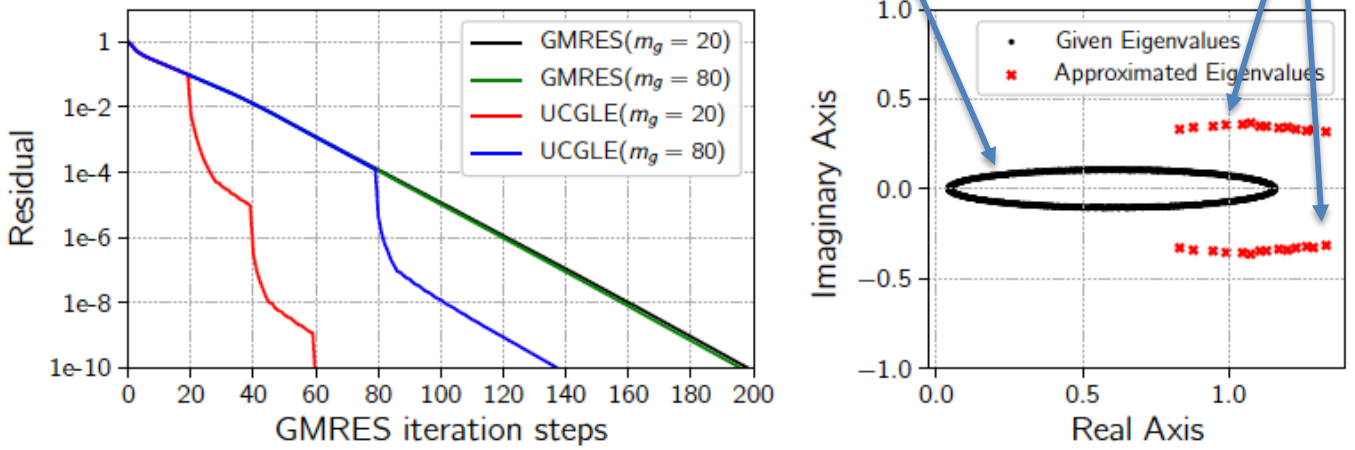
(d) Spectral Distribution IV: matrix size $= 2000$, $m_g = 20$, $d = 10$. UCGLE($eigen_1$) has $3\times$ speedup, and UCGLE($eigen_2$) has $2\times$ speedup.

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)

Thianhe 2

Given and “generated” spectrum

Ritz-Eigenvalues computed by the ERAM method (into the subspace)



(a) Spectral Distribution I: matrix size $= 2000$, $d = 10$. UCGLE($m_g = 20$) has $3\times$ speedup, and UCGLE($m_g = 20$) has $1.4\times$ speedup.

The generator allows to evaluate several behavior of the method with respect to the spectra

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.

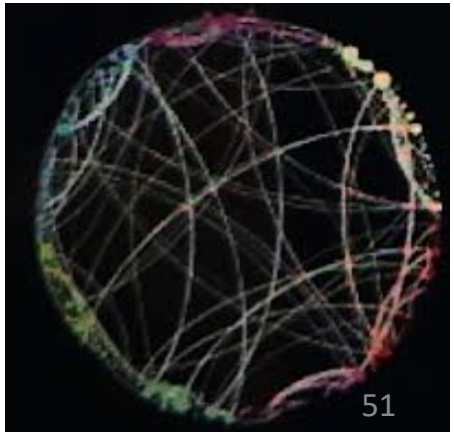
It is also a possible to generate such sparse matrices for other kind of experiments

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

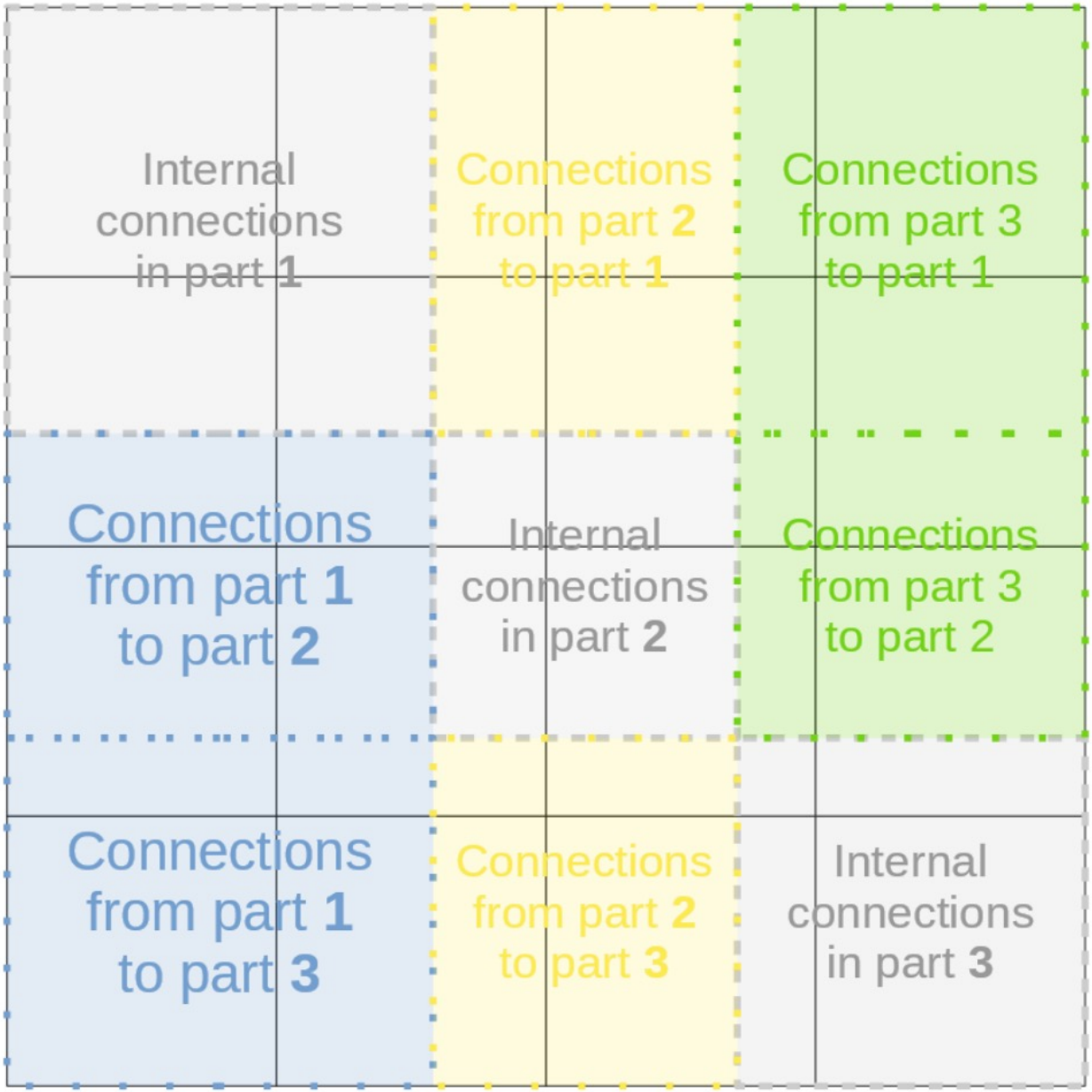
- Neurons

- 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)

BTIDG2's Documentation

INTRODUCTION

USER DOCUMENTATION

API

Indices and tables

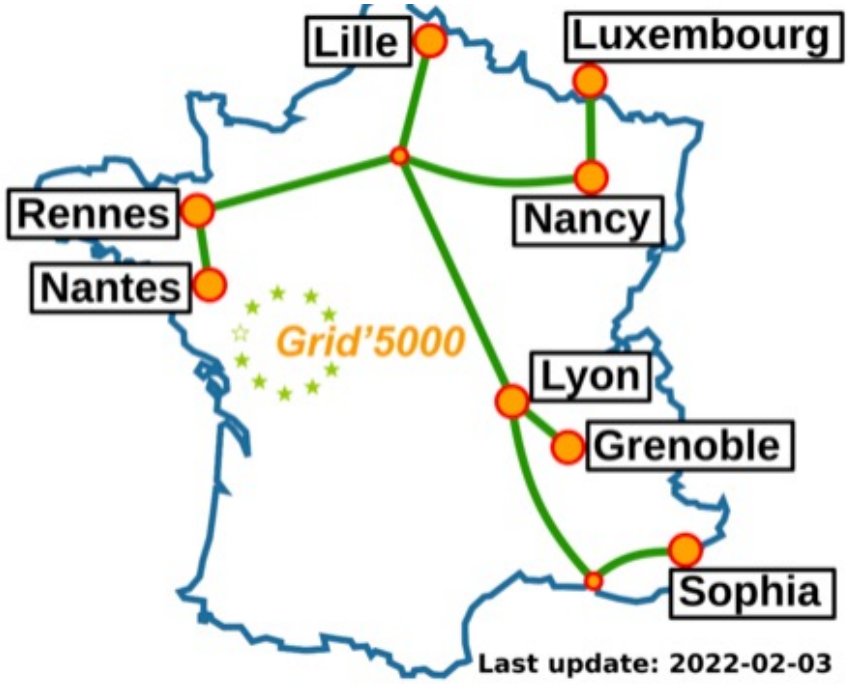
BTIDG2 github repository:
https://github.com/SMG2S/BTIDG2

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

License

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

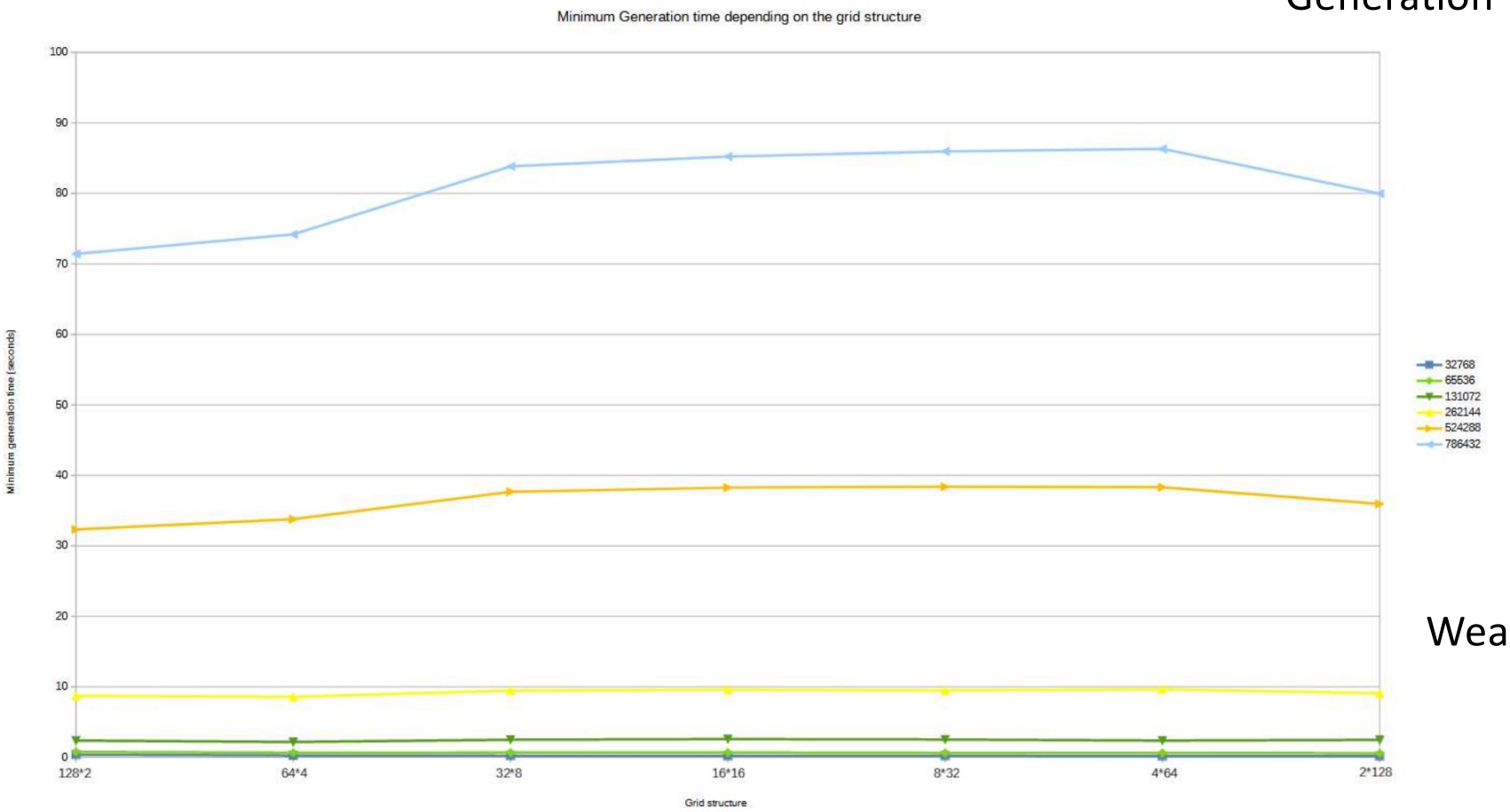
Block CSR
NNZ = 3,5 % of N^2
Densities
- 0.5% inside each parts
- 5% between parts

Diferent numbers of blocks

Process Grid tests

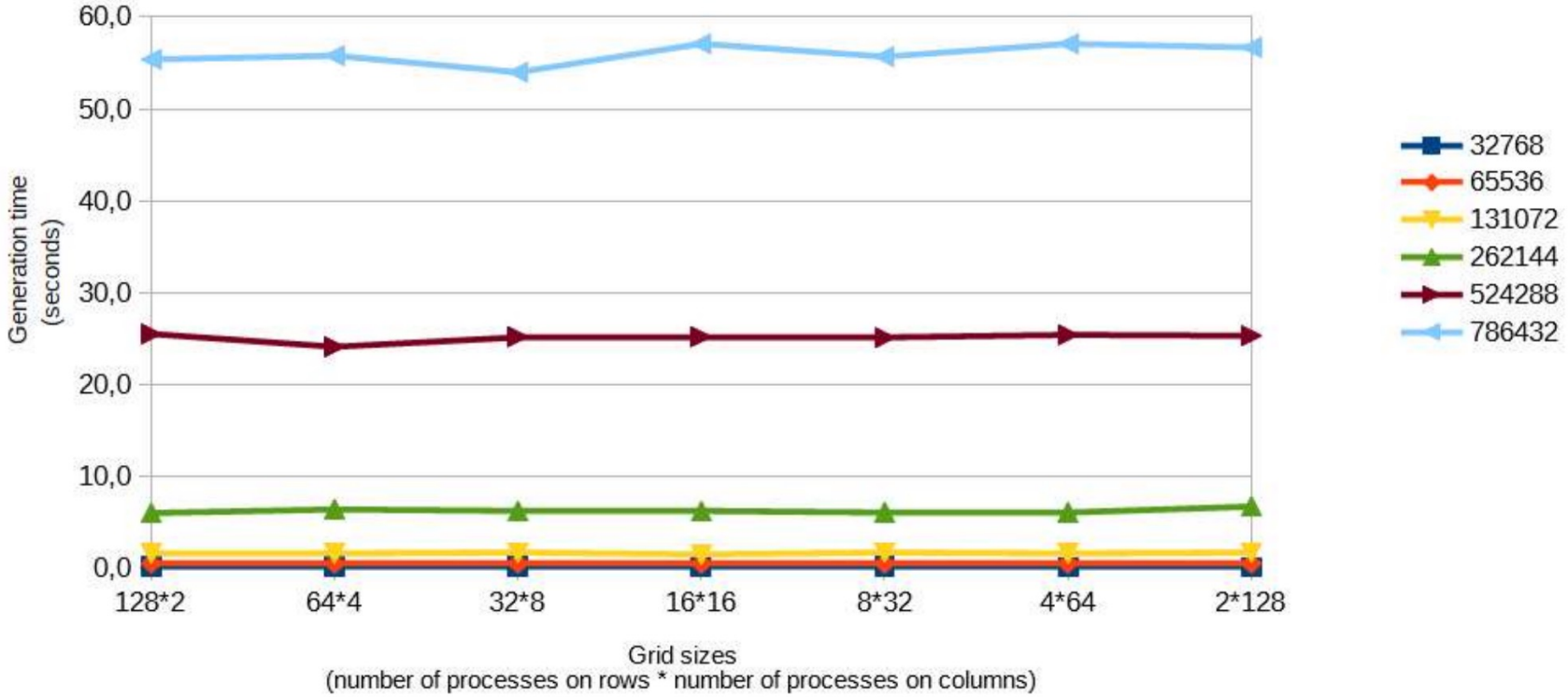
We applied the following tests to matrices whose dimension vary from 30k to 80k, while keeping the same number of allocated resources (256 cores)

Generation



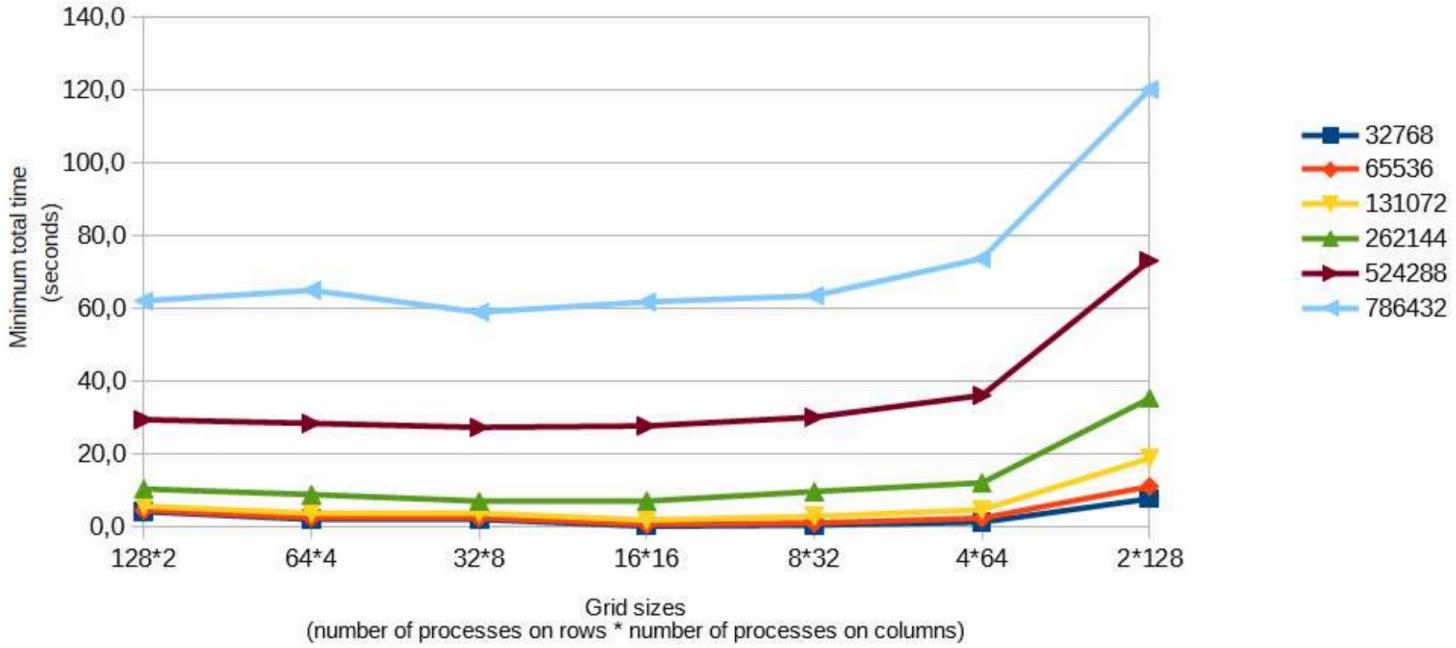
Weak scaling

Generation time :



Minimum generation time depending on the grid structure

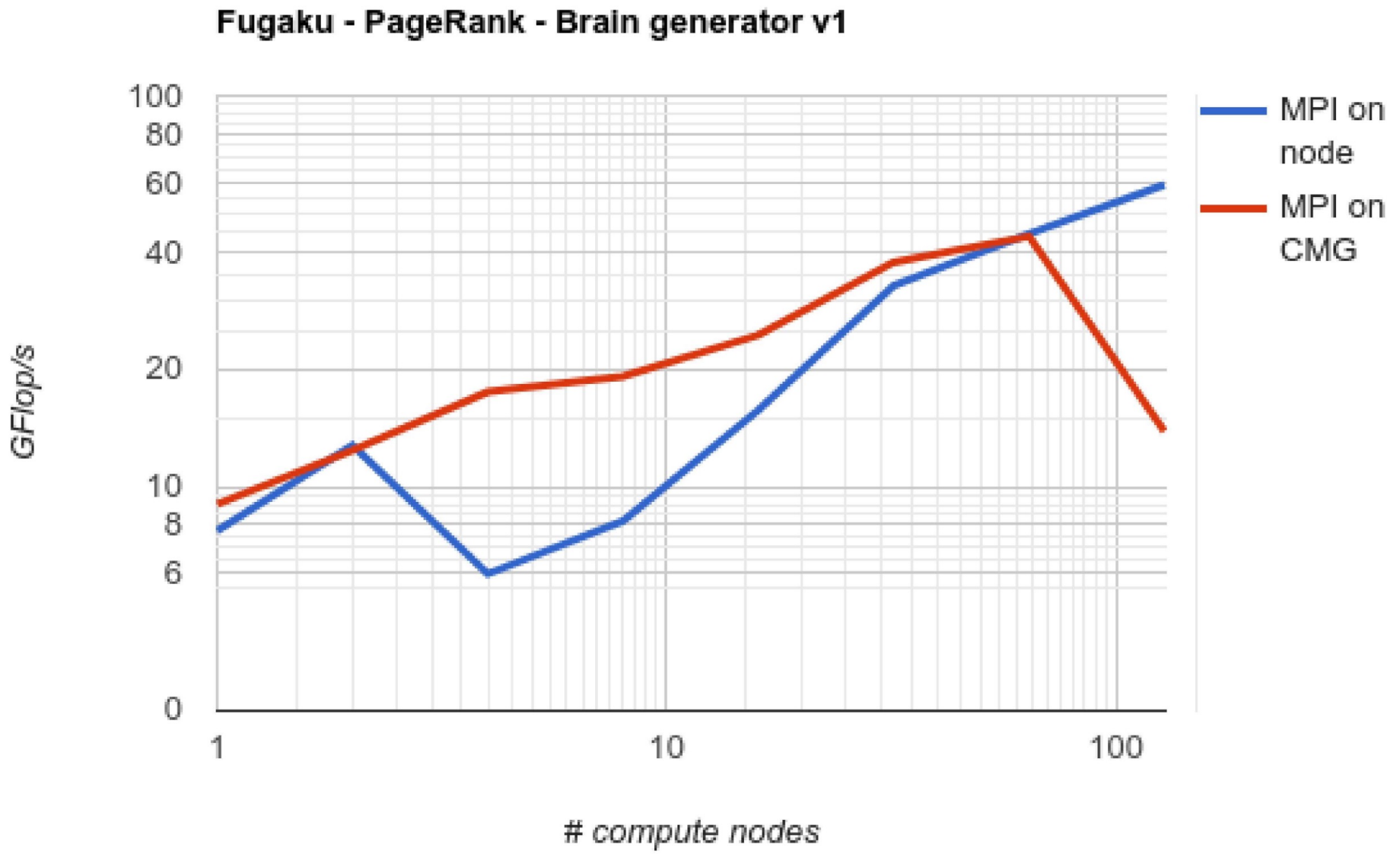
GRID5000



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

Pagerank : seqence of sparse matrix-vector products + reduction_with_add

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



Outline

1. Introduction
2. Programming Paradigms
3. Methods and Algorithms (Unite&Conquer, PageRank)
4. HPC and Machine Learning (GCN, Transformer,..)
5. Generators of Data Sets and matrices for brain-scale applicatons
6. **Conclusion**

Exascale machine are emerging but the convergence of computational science, big data and machine leraning-AI to develop new science, with often strong societal impacts, generates a complex unstable ecosystem, especially for large size problems.

Linear algebra (matrix computation, statistic, eigenvalue, projection, optimisation, …) computing would still be important, but would target new applications and new scientific communauties.

AI for science versus science for AI : or a new science

If we want to efficiently use the faster supercomputers (including new one developped for AI) for very large applications, we have to develop new multidisciplinary teams to create shared programming paradigms and new scientific developments (it already exists in some labs)

HPC challenge and new computing frontiers

- Arithmetic : mixed, new normalisations? Convergence evaluation using mixed arithmetic.
- 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, …
- Generation of data to expriments, optimisation of I/O and communications
- …..

HPC challenges for new extreme scale applications

Exascale machines are now available, based on several different arithmetic (from 64-bits to 16-8 bits arithmetic, including mixed versions and some no longer IEEE standard) and using different architectures (with network-on-chip processors and/or with accelerators). Some execution and programming paradigms are being rehabilitated, such as data flow models and data parallelism without shared memories. Brain-scale applications, from machine learning and AI for example, manipulate many huge graphs that lead to very sparse non-symmetric linear algebra problems, resulting in performance closer to the HPCG benchmark than to the LINPACK one.

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.

This two-day workshop aims to bring together senior scientists in the field of HPC and some major applications associated with it, to brainstorm on those challenges and propose potential research collaborations. The number of invited participants is expected to be less than 35 (from Asia, USA and Europe mainly). We plan to organize panels, combined with some talks.

COORDINATION SCIENTIFIQUE

CHRISTOPHE CALVIN

EWA DEELMAN

KENGO NAKAJIMA

SERGE PETITON

A 4-page report would be available by the end of June, and a larger document would be completed by the end of the summer.

https://www.association-aristote.fr/evenements/hpc-challenges-for-new-extreme-scale-applications/