Block Wiedemann Algorithm on Multicores Architectures

Bastien Vialla

LIRMM, CNRS-UM2 France

bastien.vialla@lirmm.fr

Notations

- $\mathbb{K}$ a commutative field.
- $\mathcal{M}_{n\times m}(\mathbb{K})$, ring of matrices of size $n \times m$.
- We denote by γ the number of non zeros in a sparse matrix.
- NUMA : Non Uniform Memory Access

Motivations

Solving a linear system with large sparse matrices is a computational kernel used in a wide range of applications. The block version of Wiedemann's algorithm proposed in [1] take advantage of the sparsity to achieve better performance.

Objectives

An efficient implementation of block Wiedemann algorithm on NUMA multicores architectures.

Contribution

- We efficiently incorporate the sparse block into the first step of BW algorithm.
- We provide an efficient implementation for NUMA multicores using tbb/MPI that provides excellent scaling.

Block Wiedemann algorithm

Let $A \in \mathcal{M}_{n\times n}(\mathbb{K})$, $U, V \in \mathcal{M}_{n\times k}(\mathbb{K})$ random matrices with $k \leq n$.
Block Wiedemann algorithm follows three steps:

1. Compute the first $O(\frac{2n}{k})$ elements of $S = [U^T A^i V]_{i\in\mathbb{N}}$.
2. Find the minimal matrix polynomial generator of the sequence S.
3. Compute the solution using the polynomial found in step 2.

The cost of the first step is dominant, therefore its parallelization is crucial.

Sparse blocks

We generalize sparse block from [2] in block Wiedemann algorithm. We permute non zeros elements to have a cache efficient version.

$$U = \left|\begin{array}{ccccccc} \delta_1 &&&&&& \\ \vdots &&&&&& \\ \delta_s &&&&&& \\ \delta_{s+1} &&&&&& \\ & \delta_1 &&&&& \\ & \vdots &&&&& \\ & \delta_s &&&&& \\ & \delta_{s+1} &&&&& \\ && \ddots &&&& \\ &&& \delta_1 &&& \\ &&& \vdots &&& \\ &&& \delta_s &&& \\ &&& \delta_{s+1} &&& \\ \hline &&&& \delta_1 && \\ &&&& \vdots && \\ &&&& \delta_s && \\ &&&&& \ddots & \\ &&&&&& \delta_1 \\ &&&&&& \vdots \\ &&&&&& \delta_s \end{array}\right| \begin{array}{l} \left.\vphantom{\begin{array}{c}1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\end{array}}\right\} \lfloor n/k \rfloor \text{ times} \\ \left.\vphantom{\begin{array}{c}1\\1\\1\\1\\1\\1\\1\end{array}}\right\} n \bmod k \text{ times} \end{array}$$

where $s = \lfloor n/k \rfloor$, $\delta_1, \cdots, \delta_{s+1} \in \mathbb{K}$ chosen at random.
Complexities of the first step of block Wiedemann algorithm:

	Dense blocks	Sparse blocks
Sequential	$O(n\gamma + n^2k^{\omega-2})$	$O(n\gamma + n^2)$
Parallel k cores	$O(\frac{n\gamma}{k} + n^2)$	$O(\frac{n\gamma+n^2}{k})$

Experimentations

- We use LinBox [3] for dense blocks code, and tbb for parallelization. We use an NUMA with four Intel XEON E4620 with 8 cores at 2.2Ghz and 384GB of RAM.
- The block size does not impact the time with the use of sparse blocks, see figure 1.
- tbb implementation performs badly on NUMA architectures, see table 1.
- We design an Hybrid MPI/tbb implementation that allows good speed-up on NUMA, see table 1.
- Sparse blocks perform always better, see table 1.

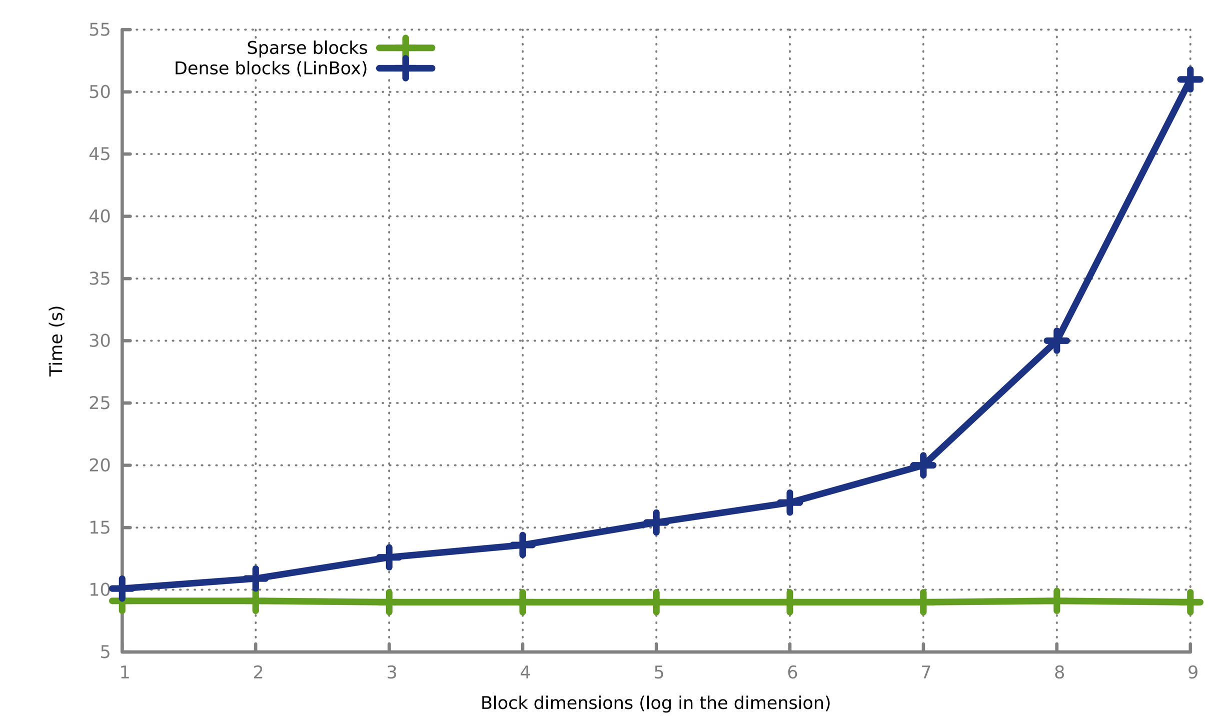

Figure 1: Sparse blocks size influence in comparison with dense blocks. Computations made with the matrix EX5 with 1 core.

	Dense blocks (LinBox)				Sparse blocks			
	tbb		MPI/tbb		tbb		MPI/tbb	
	time	speed-up	time	speed-up	time	speed-up	time	speed-up
1 cpu : 1 core	2205	1	2207	1	2160	1	2165	1
1 cpu : 8 cores	540	4	540	4	308	7	308	7
2 cpus : 16 cores	623	3.5	279	7.9	310	6.8	154	14
3 cpus : 24 cores	798	2.7	183	12	242	8.8	102	21.2
4 cpus : 32 cores	960	2.2	135	16.3	177	12	77	28.1

Table 1: Timings, in seconds, and speed-up of tbb and MPI/tbb implementations. We use the matrix rand100k.

Additional Informations

Matrices characteristics :

Name	Size	Non zeros	Problem
rand100k	$100k \times 100k$	$1.5M$	randomly generated
EX5	6545×6545	295680	symmetric powers of graphs

References

[1] Don Coppersmith.
Solving Homogeneous Linear Equation Over GF(2) via Block Wiedemann Algorithm.
Mathematics of Computation, 62(205):333–350, 1994.

[2] Wayne Eberly, Mark Giesbrecht, Pascal Giorgi, Arne Storjohann, and Gilles Villard.
Faster inversion and other black box matrix computations using efficient block projections.
Proceedings of the 2007 international symposium on Symbolic and algebraic computation - ISSAC '07, 3(1):143, 2007.

[3] http://www.linalg.org.