Performance of Very Large Very Sparse Matrix Matrix and Very Large Very Sparse Matrix Vector Multiplication on Different Cluster Architectures

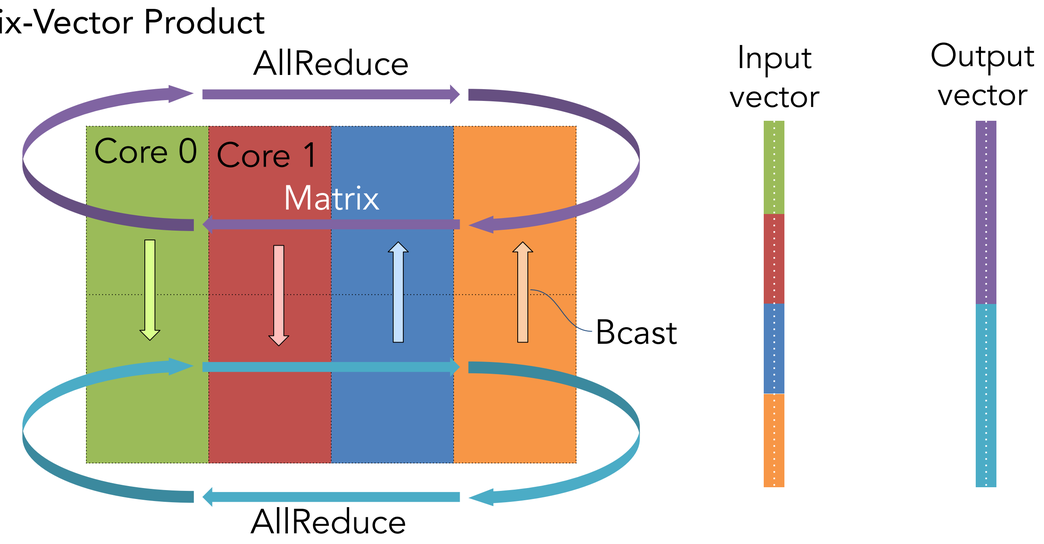
Maxence Buisson, Géraud Krawezik
Scientific Computing Core, Flatiron Institute, New York

Abstract

The Top500 is used to rank supercomputers. It benchmarks different architectures based mostly on dense matrix-matrix multiplications. However linear algebra problems studied in recent years require very large sparse matrices (eg: machine learning manipulating large datasets), meaning that the overall performance of the code will be far from the peak obtained with dense linear algebra on commonly used cache-heavy computer architectures. In this study, we focus on several benchmarks (PageRank, Conjugate Gradient, Matrix Multiplication from Attention Layer in Machine Learning) and test their performance on different cluster architectures (with different CPU families and multiple generations of interconnect). Our implementation is designed to handle very large, very sparse problems by distributing the vectors across the system.

Data Distribution

Matrix-Vector Product



1. Matrix distribution:
   a. Row, Col : Specified at runtime
2. All cores on a given block column hold the same parts of the input vector
3. All cores on a given block row hold the same parts of the output vector
   a. After the multiplication AllReduce is used on each block row
4. For the next iteration, Bcast is used on each block column to redistribute the input vector

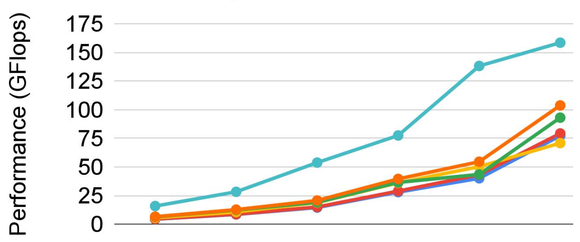
Matrix-Matrix Multiplication:

- Partially Distributed:
  - Input and output matrices are distributed like the vectors
- Fully Distributed:
  - Distribution of input and output matrices by column

Results

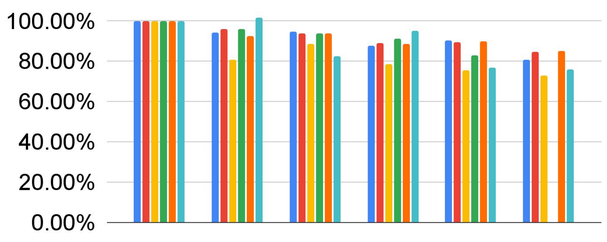
PageRank: Strong Scaling

N= 100000000 NNZ per row = 100



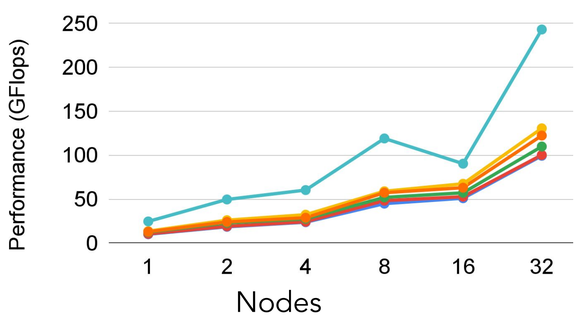
PageRank: Parallel Efficiency (Weak Scaling)

N = sqrt(nodes) * 10^7 NNZ / row = sqrt(nodes) * 100



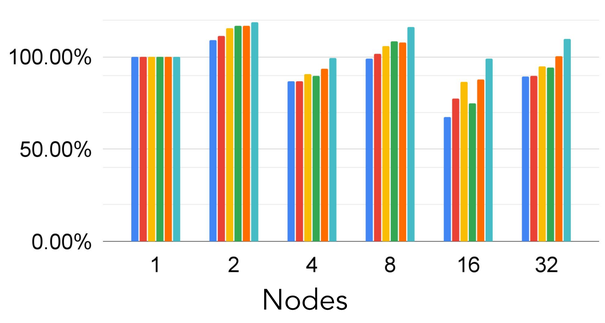
Conjugate Gradient: Strong Scaling

N = 10^7 NNZ / row = 100



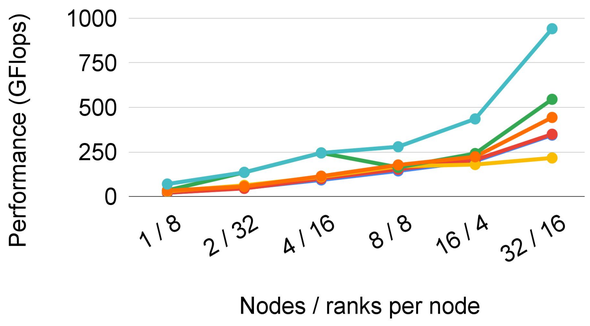
Conjugate Gradient: Parallel Efficiency (Weak Scaling)

N = sqrt(nodes) * 10^7 NNZ / row = sqrt(nodes) * 100



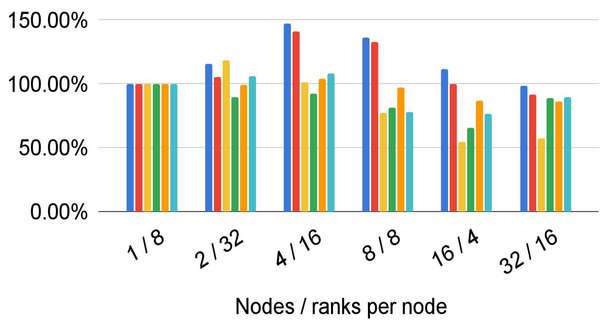
Matrix Multiplication: Strong Scaling

N = 10^7, NNZ / row = 100



Matrix Multiplication: Parallel Efficiency (Weak Scaling)

N = sqrt(nodes) * 10^6 NNZ / row = sqrt(nodes) * 100



Skylake Cascadelake Rome Icelake Icelake Popeye Genoa

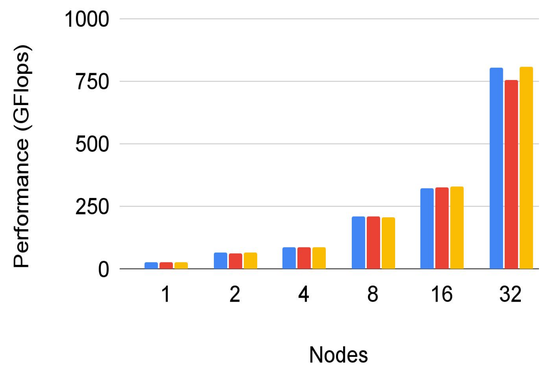
Analyse

Number of Floating operation per cycle during conjugate gradient: 0.3

Number of Floating operation per cycle during Matrix Multiplication: 0.5

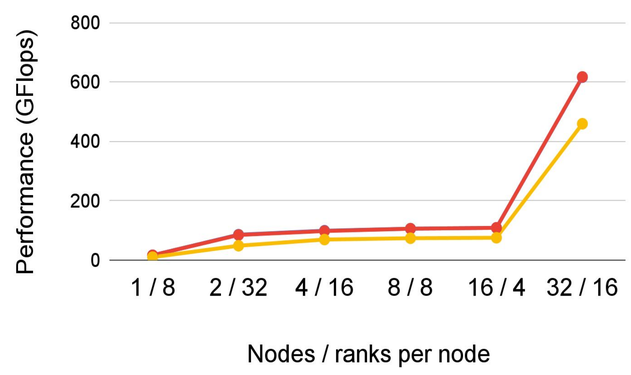
Conjugate Gradient : SIMD on weak scaling

N = sqrt(nodes) * 10^7 NNZ / row = sqrt(nodes) * 100



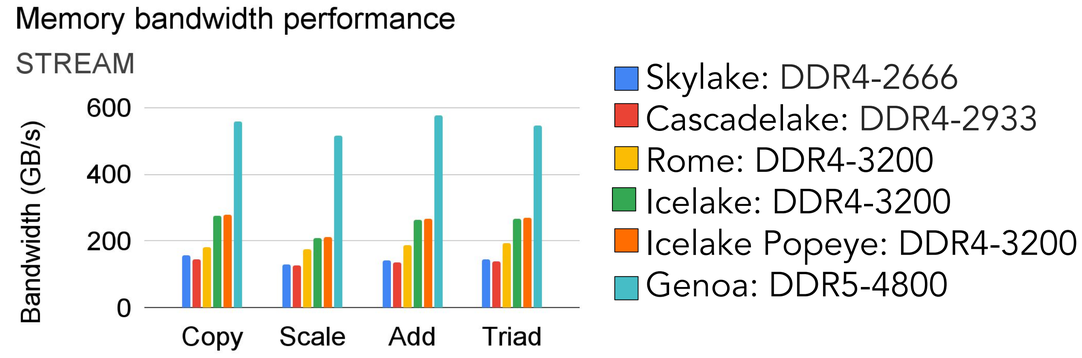
Matrix Multiplication: SIMD on weak scaling

N = sqrt(nodes) * 10^6 NNZ / row = sqrt(nodes) * 100



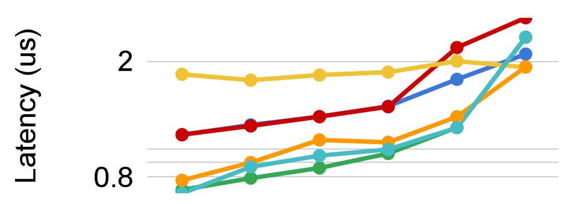
Avx512 Avx2 Without

Memory bandwidth performance



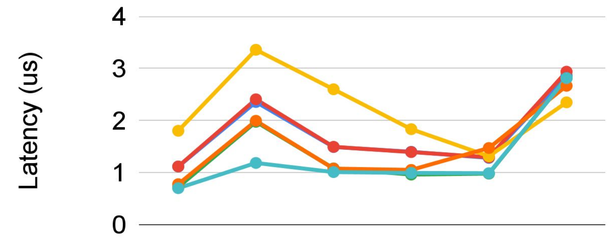
MPI_AllReduce performance

Pattern similar to Conjugate Gradient



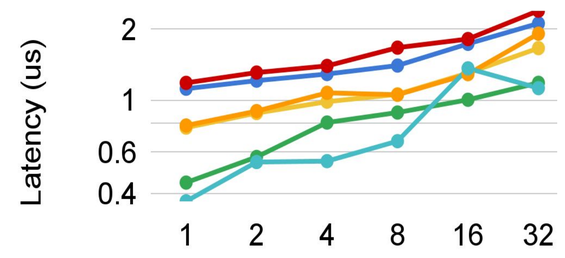
MPI_AllReduce performance

Pattern similar to Matrix Multiplication



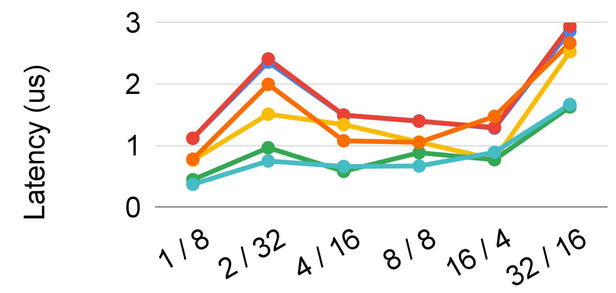
MPI_BCast performance

Pattern similar to Conjugate Gradient



MPI_BCast performance

Pattern similar to Matrix Multiplication



Outlook

- Test with different NUMA settings in the BIOS (for now only used 1 NUMA region per socket)
  - Recommended by some vendors for hybrid MPI + OpenMP codes
  - How does the software view matching the hardware impact performance?
- Compare Fugaku results with Ookami (tested at Stony Brook without Tofu)
- Compare the 2 implementations of sparse matrix to dense matrix multiplication

CONTACT :
maxence.buisson2@orange.fr