CS639: Homework 5 (git) Rajesh Shashi Kumar

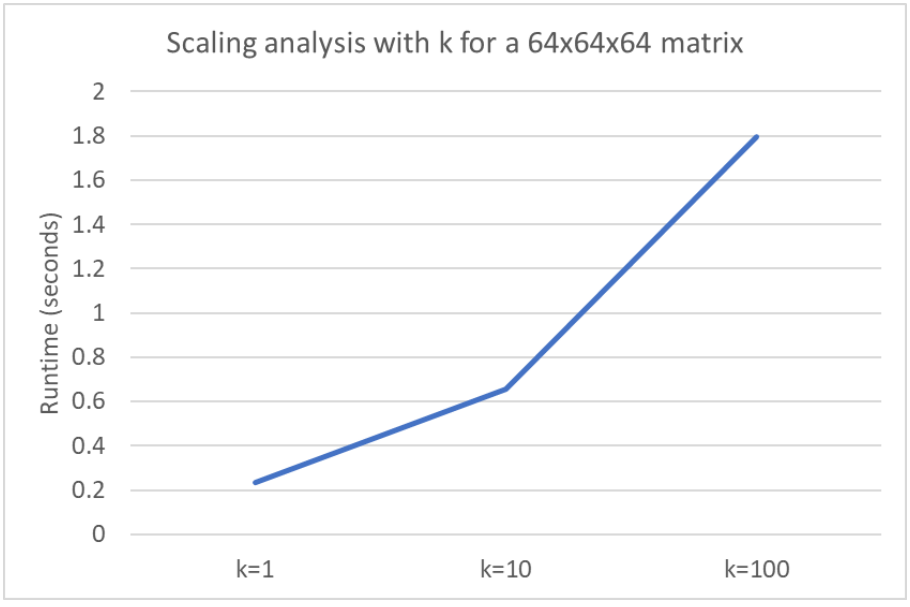
1 Machine Configuration

Attribute	**Value**
Hostname	barolo.cs.wisc.edu
OS	Ubuntu 20.04.5 LTS
Compiler	g++ (gcc version 9.4.0), OpenMP version 4.5
Compile command	*(Makefile included in root directory of zip file)* *icc *.cpp -Wall -O3 -o pardiso -qopenmp -mkl*
CPU	AMD EPYC 7451 24-Core Processor
Cache configuration	L1-3MiB, L2-24MiB, L3-128MiB
Memory bandwidth	~150 GiB/s (Source,2). The machine has 256GB of memory spread across (8 out of 16) slots each housing 32GB stick. This information was retrieved using *sudo lshw -class memory*
Number of threads	24 cores * 2 threads/core = 48 threads
Dependencies	GCC, OpenMP, ICC (source /s/intelcompilers-2019/bin/iccvars.sh intel64)

	A	B	C	D	E	F
1	**Matrix: 64x64x64**		**Sparsity of L factor (nnz)**	**Factorization Operations (GFLOPS)**	**Factorization Bandwidth (GFLOPS/s)**	**Solve runtime (s)**
2		Minimum Degree Algorithm	187686049	981.144287	361.292603	0.490172
3	User-defined permutation	Minimum Degree Algorithm	901970427	3449.491455	13.608821	3.294694
4		Nested Dissection	901928763	3449.17627	12.953135	3.782434

Fig 1. Role of re-ordering in parallel efficiency of PARDISO

- From the MKL docs, "PARDISO uses the user supplied fill-in reducing permutation from theperm array. iparm[1] is ignored". We can see that this is true, from rows 2 and 3 of Fig 1.
- The user-defined "no-permutation" on row 3 (Fig 1), performs significantly worse than the Minimum Degree Algorithm (on row 2 (Fig 1)). This is because we completely eliminate the search for favorable conditions in the sparsity patterns in Phase 1 that plays an important role in performance of sparse computation.
- From Columns C and D of Fig 1, we can observe that the number of non-zero entries in L directly affects the number of operations needed in factorization almost linearly. However, no permutation achieves poor factorization throughput (col E) due to limited opportunities in vectorization due to lack of favorable reordering by engineering regularity.
- From Fig 2, we can see that 100 right hand sides costs less than 100x the cost of solving for a single one due to the reuse of memory elements pushing it towards compute-boundedness.



	A	B
1		**Runtime (s)**
2	k=1	0.235021
3	k=10	0.656408
4	k=100	1.79532

Fig 2. Linear scaling of multiple-RHS in the solve step