A UT/ORNL PARTNERSHIP
NATIONAL INSTITUTE FOR COMPUTATIONAL SCIENCES
NICS

Performance Comparison Framework for Numerical Libraries on Cray XT5 Systems

Bilel Hadri and Haihang You

University of Tennessee
NICS

Introduction and motivation

- Facts
  - ALTD (Automatic Tracking Library Database) ref Fahey, Jones, Hadri, CUG 2010
    - Numerical libraries are one of the most used packages
    - LAPACK library is linked with different package
      - Craypat, Amber, nwchem, Cactus, Abinit, Chromo and qdp
  - Kraken supports optimized version of LibSci (Cray), ACML (AMD) and MKL (Intel) with different compilers (PGI, GNU, Intel, Cray)
  - Architecture-optimized versions of libraries is not always used on Kraken by the users.
- Goal
  - Study Choosing the most efficient library for a given application is essential for achieving good performance.
  - Design a framework to help researchers to determine the fastest library choices for their applications.

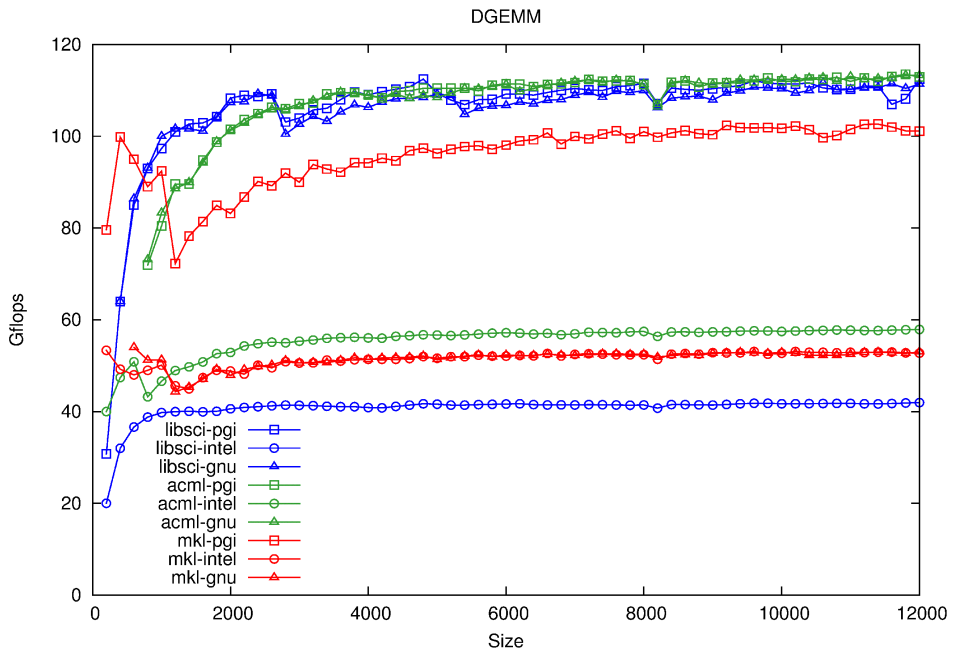
NICS

Interface framework

- Three numerical libraries have been studied: LibSci (10.4.5), ACML (4.4.0) and Intel MKL (10.2) using dense and random matrix.
- Each numerical library is built with the following compilers: PGI (10.6.0), GNU (4.4.3) and Intel (11.1.038).
- Developed a set of scripts to build executables, submit jobs, gather data and plot results automatically.
- Store the results to build a Knowledge database
  - Function, Data size (vector/matrix),
  - For ScaLAPACK, FFT, need to add other dimensions:
    - number of cores, topology , accuracy
- Further development is needed to provide an API for the users
  - to query the system for suggestions of using certain library with certain compiler for better performance for their scientific applications

BLAS level 3 : Matrix matrix multiplication

~ 3 times

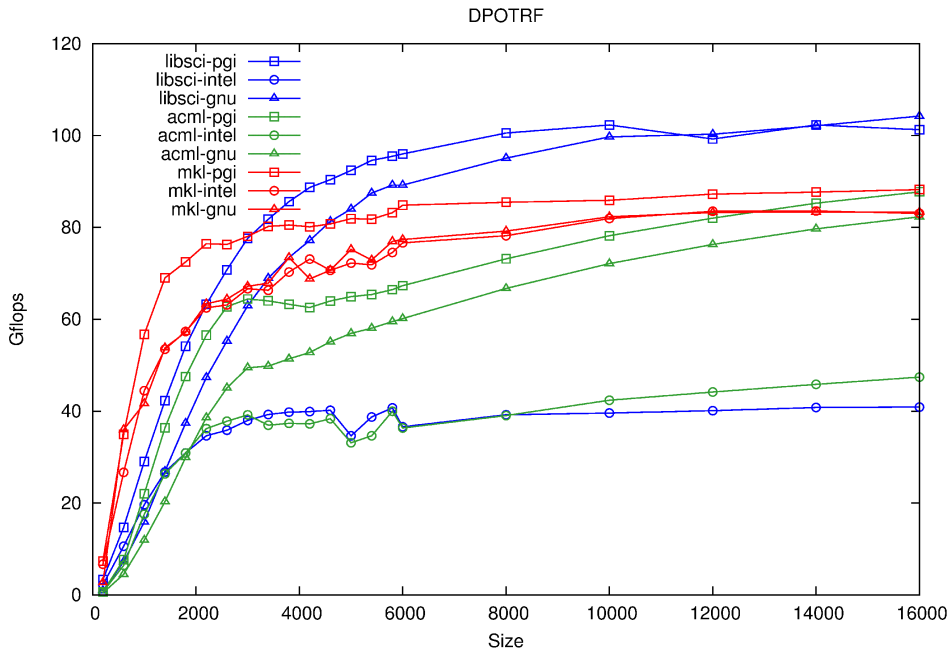


~ 2 times

- ACML library has the fastest implementation for DGEMM, reaching 113 Gflop/s (91% of the theoretical peak). LibSci is 2% slower than ACML.
- Libraries compiled with Intel perform less than 40% of the theoretical peak.
- For very small matrices, MKL is the fastest implementation,

NICS

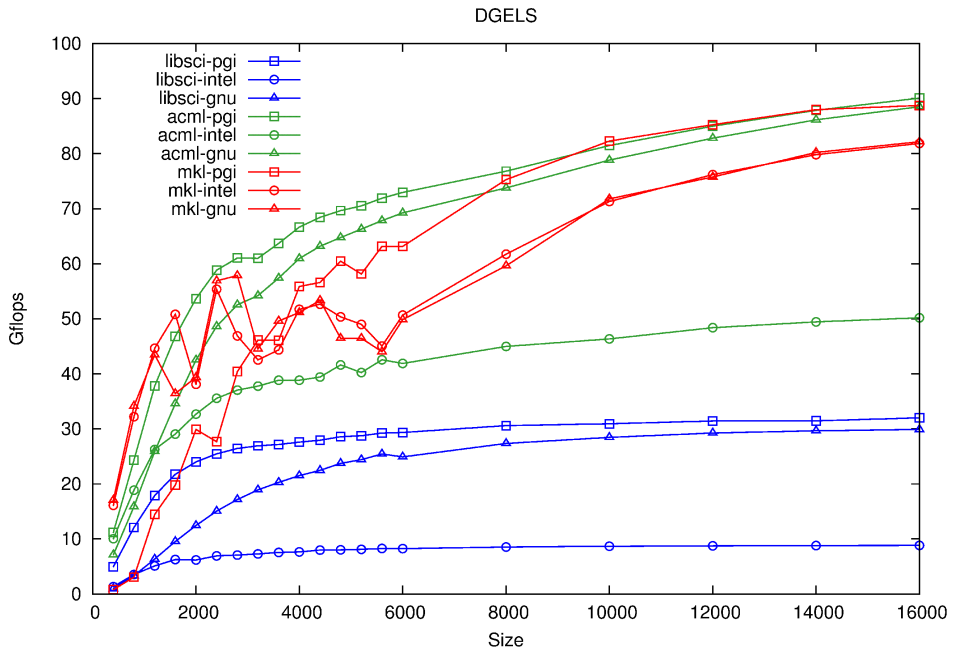
LAPACK – DPOTRF : Cholesky factorization



- DPOTRF is based mostly on DGEMM. Both MKL and LibSci are the fastest implementation for the small and large matrix size respectively.
- ACML and LibSci built with Intel compiler gives the worst performance

NICS

LAPACK – DGELS – QR factorization and solve

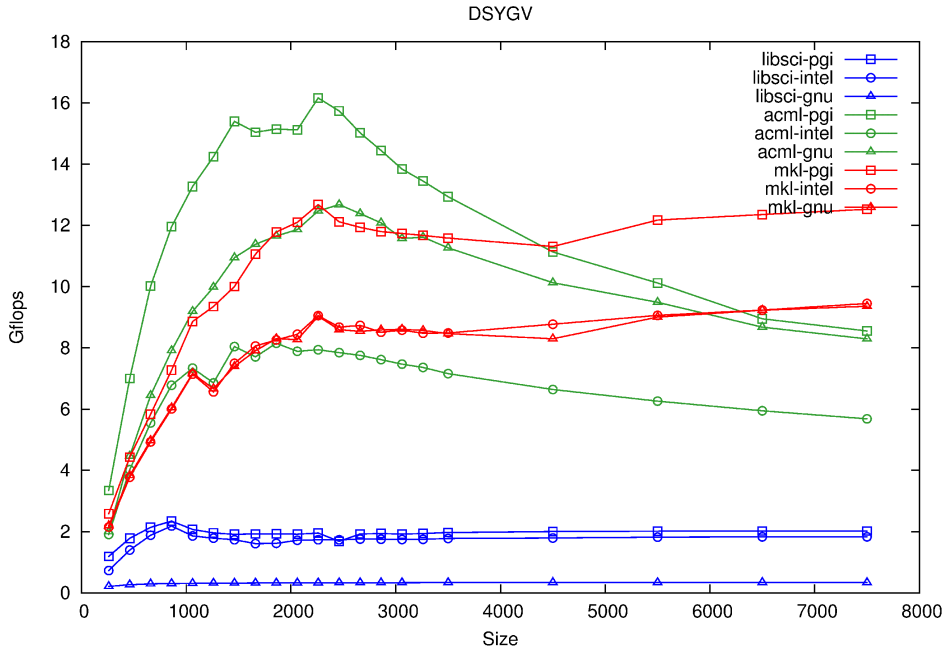


~ 3 times

- ACML achieved the best performance with PGI and GNU.
- LibSci has a behaviour of a library not used the optimizing BLAS. Compare to the ACML, LibSci is 3 times slower with PGI and GNU and 10 times slower with Intel
- Libsci has the small block size (NB)and it is set by default to 32, while for MKL, it depends on the matrix size (it varies from 16 to 128)

NICS

LAPACK : DSYGV : generalized symmetric-definite eigenproblem

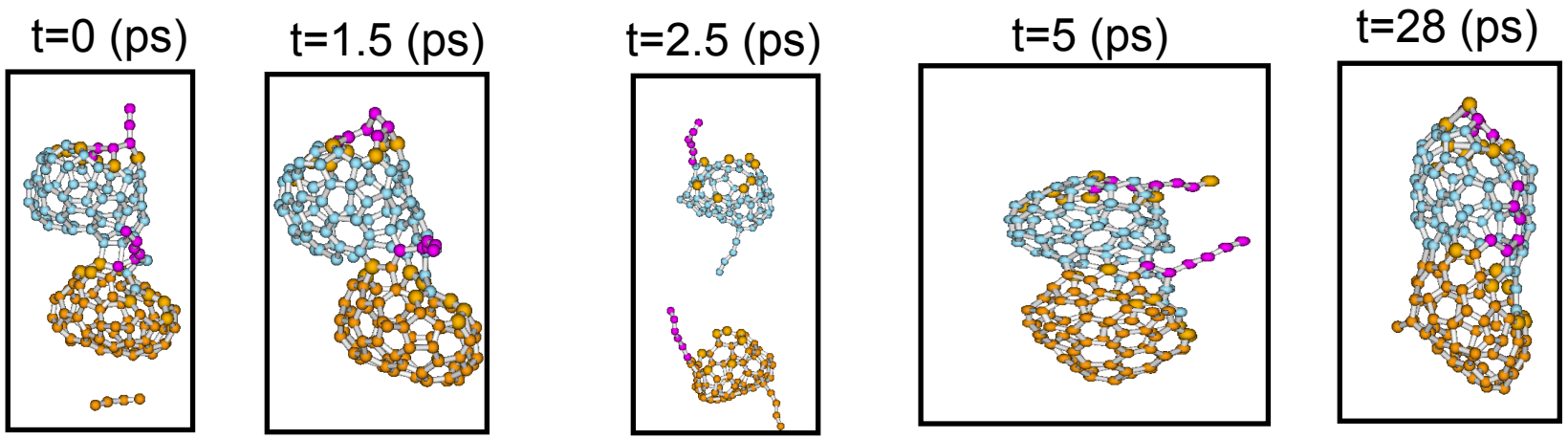


~ 6 times

- ACML and MKL gives the fastest implementation.
- LibSci is the slowest implementation and with GNU, the performance does not exceed 0.3 Gflops while ACML reaches 16 Glops.

NICS

DFTB density-functional tight-binding



DFTB is a quantum chemistry molecular dynamic application solving eigenvalues of the Hamiltonian :

$$H(R)\Psi = E(R)\Psi$$ refs: Zheng, Morokuma, Jakowski, Int. J.Quantum Chem. (2009)

Algorithm:

1) Solve electronic Schrödinger equation (with DSYGV) at nuclear configuration until convergence (~10-20 iterations per MD step)

2) calculate forces (gradient of energy)

3) move nuclei classically (Newton Eq. and quantum forces from 2)

4) repeat step 1-3 several thousand of times (typically 10,000 MD steps)

NICS

DFTB results

- Two simulations are considered : small with 121 atoms and a larger one with 363 which makes the matrix size 460 and 1460 respectively.
- Memory bound. Problem scales cubically with number of atoms $O(N^3)$
- The default is the LibSci with PGI. Solving the system takes about 80% of the total execution

		DSYGV		Total Time in sec		Speedup	
Library	Compiler	small	large	small	large	small	large
LIBSCI	PGI	31.3	1193	38.9	1416	1.00	1.00
	GNU	205	7034	221	7136	0.18	0.20
	Intel	42.3	1401	51.2	1599	0.76	0.89
ACML	PGI	7.49	137	15.3	355	2.54	3.99
	GNU	11.5	177	22.3	374	1.74	3.79
	Intel	15.3	275	22.8	468	1.71	3.03
MKL	PGI	11.7	284	21.2	502	1.83	2.82
	GNU	15.8	388	25.5	573	1.53	2.47
	Intel	16.2	292	23.9	475	1.63	2.98

- Performance match the DSYGV Performance. ACML best implementation with a speedup of 4 (reduced to 50% of the overall time in DSYGV) while LibSci with GNU is the worst !
- For Larger size, MKL library improves and reduce the gap from ACML

NICS

Conclusions

	Small	Medium	Large
DAXPY	MKL/PGI	LibSci/PGI	LibSci/PGI
DGEMV	LibSci/PGI	LibSci/PGI	ACML/ LibSci
DGEMM	LibSci/PGI	ACML/PGI	ACML/PGI
DGETRF	ACML/PGI	LibSci/PGI	LibSci/PGI
DGESV	ACML/PGI	LibSci/PGI	LibSci/PGI
DPOTRF	MKL/PGI	LibSci/PGI	LibSci/PGI
DGELS	MKL/Intel	ACML/PGI	MKL/PGI
DGEEV	ACML/PGI	LibSci/PGI	LibSci/PGI
DSYGV	ACML/PGI	ACML/PGI	MKL/PGI

- Different library implementations have different strong points.
  - LibSci gives generally the fastest implementation (except for DGELS and DSYGV).
  - ACML should be considered to be the safest solution to avoid weak performance
- The best library implementation often varies depending on the individual routine and the size of input data
- Experiment with different versions and parameters and find what works for your code

NICS

Future work

- Similar performance on other architectures and other vendor libraries
- Expand the work to ScaLAPACK and FFT
- Provide an API for the users in order to query the system for suggestions of using certain library with certain compiler for better performance for their scientific applications

NICS

Bilel Hadri
University of Tennessee, NICS
bhadri@utk.edu

Haihang You
University of Tennessee, NICS
hyou@utk.edu