EVALUATION OF SPARSE LINEAR ALGEBRA OPERATIONS IN TRILINOS

Mohammad Siahatgar\textsuperscript{1,2}, Gabriele von Voigt\textsuperscript{1,2}

\textsuperscript{1}Leibniz Universität IT Services
Leibniz Universität Hannover
Schloßwender Straße 5, 30159 Hannover, Germany
e-mail: \{siahatgar, vonvoigt\}@luis.uni-hannover.de

\textsuperscript{2}L3S Research Center
Appelstrasse 9a, 30167 Hannover, Germany

**Keywords:** Performance Evaluation, Sparse Algebra, Energy Consumption, Trilinos, HPC

**Abstract.** The performance of numerous scientific libraries and applications depends heavily on efficiency of sparse linear algebra operations. In this paper, we survey the performance of several parallel sparse vector and matrix kernels provided in the Trilinos framework on supercomputer systems Cray XC30/40 and IBM Blue Gene/Q. The linear algebra operations in Trilinos are handled by one of the two packages Epetra or Tpetra. While the former is the most-used, the latter is the target of future developments and supports larger scale problems as well as shared memory parallelism. We compare the results obtained from both packages together with the MPI only and hybrid solutions. The hybrid parallelism is managed by the package Kokkos, which aims for performance portability among different architectures. We report the efficiency of a single node of the system and demonstrate the scalability behavior of the benchmarks up to 38,400 cores of the HLRN-III systems. Furthermore, for the Intel processors used in the Cray system we present measurements of the energy consumption of the kernels and compare the Energy-to-Solution between different compilers and parallel programing paradigms. In addition, we discuss the effect on the performance and the energy consumption by linking the vendor provided libraries compared to the user-compiled versions. These extensive comparisons obtained on the top most performant supercomputer systems help users and developers as a starting point for determining an optimal development strategy.
Mathematical libraries contribute at the core level to any scientific application and compose a basic framework for scientific computing. Many complex but well-known and often-used algorithms have been already implemented, tested and are ready to be incorporated in user applications. With an increasing number and coverage, a key question is to determine the importance and effectiveness of any library. Developers often carefully consider the functionality of the libraries regarding the desired capability besides their performance on their target hardware architecture. In many cases, the latter point can be important and even is the decisive factor. For example, quantum chemistry and molecular dynamics packages benefit much from GPUs, since most of their calculations are in essence stream processing. Furthermore, great efforts are being made to optimize the essential and low-level libraries needed for general applications. Examples are the BLAS and LAPACK libraries, of which many HPC vendors (such as IBM, Intel, Cray, AMD, Nvidia and Intel) are providing highly tuned versions for their hardware. Moreover, the dependency as well as cooperation between related libraries is constantly increasing. With a growing number of options (in both software and hardware terms), evaluation of libraries is becoming more and more relevant in the lifetime of scientific applications. Such evaluations should help developers to make informed decisions about the value of a particular library in regard to their objectives.

Sparse linear algebra calculations play a significant role in many algorithms in large-scale computational codes. The sparse matrix-vector and matrix-matrix multiplications arise in many contexts such as iterative solvers, eigenvalue problems and graph algorithms. These two key kernels dominate the performance of many solvers and may easily correspond to the most of the total run time. Thus studying the behavior of the implementations on top of the hardware (such as many-core, many-node and hybrid systems) is highly influential concerning the performance of the application. Numerous efforts [1-6] have been made to implement these kernels with optimized algorithms and tuned to specific memory and processor architectures.

In this paper, we have evaluated several sparse vector and matrix operations provided from the Trilinos library. The Trilinos project [7,8] from Sandia National Laboratories aims to develop and implement robust algorithms and software packages within an object-oriented framework for the solution of large-scale, complex multi-physics engineering and scientific problems in parallel. It consists of many self-contained packages each designed for specific objectives. The project also helps leveraging the well-established libraries such as BLAS and LAPACK with emphasizing abstract interfaces for maximum flexibility of component interchanging. Furthermore, many applications are standardizing on the Trilinos APIs, resulting to access of all Trilinos solver components as well as underlying libraries without unnecessary interface modifications. Several supercomputing facilities provide an installed version of Trilinos for their users. For example, Cray supercomputers come with Trilinos installed as part of the Cray Scientific and Math Libraries [9]. This release includes the Cray Adaptive Sparse Kernels (CASK) which are tuned for the Cray environment, and provide improved performance over native installation under most circumstances especially for operations with very sparse matrices [10].

Trilinos relies on two of its packages which implement most of the linear algebra capabilities, namely Epetra and Tpetra. The package Epetra has been the core linear algebra package, but due to its limitations in design and lack of support for large systems and new architectures, the package Tpetra is the target of future developments. In summary, their features can be compared by the following criteria:
• **Data-type**: Epetra supports only double-precision floating-point operations (of C++ type `double`). Other types such as complex numbers or higher precision floating-points are exclusively in the scope of Tpetra.

• **Very large problems**: Initially, Epetra used the C++ `int` data-type for storing global and local indices (32 bits), therefore limiting the maximum number of degrees-of-freedom in the problem to less than 2 billion. Needless to say, this limit is undesirable for high performance applications and the current systems can go beyond this limit. A workaround has been introduced [11] for the support for 64-bit global indices (C++ data-type `long long`) in Epetra and other depending packages. Tpetra has also optional support for 64-bit local indices, which allows that many entities within a single MPI process.

• **Shared Memory Parallelism**: While supporting MPI parallelism, Epetra only supports OpenMP shared memory parallelism for a few kernels but usable up to a few threads per MPI rank. Tpetra has a more general “MPI+X” approach, where X stands for various shared memory parallel programming models. Tpetra implements this in a performance-portable way by using the Kokkos shared memory parallel programming package.

The standalone package Kokkos [12] is the Trilinos foundation for thread-scalable parallelization. It implements a C++ template library for writing performance-portable applications targeting major HPC platforms. For that purpose, Kokkos provides abstractions for both parallel execution of code and data management. The goal of the package is to implement a performance-portable shared memory parallel programming model and data containers, in order to let users to write an algorithm once, and then just change a template parameter to get the optimal data layout for a particular hardware.

2 **BENCHMARKS**

In this paper, we have surveyed parallel sparse linear algebra kernels provided by both packages Epetra and Tpetra. For the Epetra benchmark, we have used the Epetra Kernels Benchmark driver from the Mantevo project [13]. Mantevo proxy applications are open source software and the Epetra benchmarks is available for download under a GNU Lesser General Public License (LGPL). This driver executes key performance-impacting Epetra kernels for sparse matrix-vector and sparse matrix-multivector kernels. Additionally, we have refactored a version of this benchmark to use Tpetra. The operations performed are calculation of sparse vector 2-norm (NORM), dot product of two vectors (DOT), generalized sparse vector addition of the form $\alpha x + y$ (AXPY), and sparse matrix-vector (SpMV) as well as sparse matrix-matrix (SpMM, with 2, 4, and 8 columns) products (with same global dimensions for vectors and matrices). Among these, SpMV product is by far the most computationally expensive component of sparse iterative linear solvers [3]. SpMV is a memory bandwidth bound problem and its performance changes highly with respect to matrix characteristics. One should note that there is no “General Purpose” SpMV code, since the applicability of an optimization applies only to types of matrices obtained in different problem or set of problems. For our benchmarks, we have constructed a two dimensional 25-point finite difference stencil matrix, which is stored in the Compressed Row Storage (CSR) format.

2.1 **Hardware resources**

We have evaluated the performance of the two aforementioned benchmarks and compared the results in two supercomputing centers HLRN-III and JUQUEEN, ranked 83rd and 11th in the Top 500 list of November 2015, respectively [14]. The former system is operated by North-German Supercomputing Alliance (Norddeutscher Verbund zur Förderung des Hoch- und
Höchstleistungsrechnens – HLRN) and is a distributed supercomputer system hosted at two locations Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and Leibniz Universität IT Services (LUIS, formerly RRZN) at University of Hannover. The HLRN-III system consists of two almost identical complexes based on Cray XC30/XC40 supercomputers and the total system comprises about 87,500 cores with a total peak performance of 2.7 PFLOP/s. The latter system is an IBM Blue Gene/Q system built for the Jülich Research Centre (Forschungszentrum Jülich – FZJ) with 458,752 cores and overall peak performance of 5.9 PFLOP/s.

The benchmarking on the Cray system is performed at the Hannover complex, which composes of two Cray systems with Intel processors and Cray Aries interconnect [15]. The nodes of the Cray XC30 system contain two Intel Xeon Ivy Bridge CPUs (E5-2695v2) operated at 2.6 GHz while in the Cray XC40 system the Intel Xeon Haswell CPUs (E5-2680v3) operated at 2.5 GHz are used. The IBM Blue Gene/Q system at Jülich uses IBM PowerPC A2 processors at 1.6 GHz with 16 cores per node [16]. The energy usage of each rack (1024 nodes, 16384 cores) without jobs running is about 30 kW and on load on average 60-70 kW [16]. The peak consumption of each A2 processor is about 55 W. This is more or less half the TDP value of the Intel Xeon CPUs installed at HLRN-III systems (with 12 cores per CPU). In comparison, the theoretical peak performance for the A2 processor is 204.8 GFLOP/s, and 230 GFLOP/s for the Intel Xeon Ivy Bridge CPUs at HLRN. For the comparison between two systems, we executed the benchmarks on subset of two systems with similar number of physical cores, remarking that the PowerPC A2 cores are capable of four-way multithreading.

2.2 Details of execution

For the Cray system, both benchmark compiled against a local installation of Trillions as well as the Cray provided library. The Trilinos library provided from Cray has the version 11.12 and supports only MPI parallelism. This is the latest version of Trilinos without the explicit requirement of C++11, which the Cray compiler (at the time of writing) does not support. For the local installation we used Trilinos version 12.2.1 linked against the Cray LibSci library [9] for BLAS and LAPACK functions, which includes the tuned CASK library. The codes have been compiled on Cray systems using Intel compiler version 15.0.1 and GNU compiler version 4.9.1. The architecture dependent instruction sets were enabled (i.e. AVX for Ivy Bridge and AVX2 and FMA for Haswell) and optimization flag –Ofast was used in all compilers. Additionally, 8 MB huge pages were also allocated. The effect of using such larger memory pages was found to be substantial (e.g. more than 60% increase in performance using 6,000 MPI processes).

For the Epetra Benchmark, we have compiled both serial (MPI only, without the OpenMP flag) and hybrid (MPI+OpenMP) versions. Cray MPI library uses MPICH2 distribution from Argonne, enhanced by tunings for the Aries interconnect. For the Tpetra benchmark, similar versions were produced as well as another hybrid option using POSIX threads (MPI+Pthreads). For the hybrid versions, we ran the benchmarks on a single node with varying number of threads from one to the number of cores (enabling hyper-threading did not lead to any improvement in performance). In case of hybrid parallelism, a single MPI process per node (with no core binding) has been assigned. For the MPI only case, one core has been reserved for one MPI rank. For comparison of the performance of different benchmarks with different programming model (MPI, OpenMP, Pthreads), the matrix size per node has been kept constant (i.e. solving precisely the same equation with same number of FLOPs in each node). The FLOP/s is calculated by $2nmN^2 / t$, in which $n$ is the number of point in the stencil (in our case 25), $m$ the number of trials (on average 100), $N$ the size of the global matrix and $t$ the time of the calculation of the kernel. The local matrix size per node kept constant and of the order $10^3$. 

1384
We compiled Trilinos as well as both benchmarks on the JUQUEEN system using GNU compiler version 4.8.1 linking to the ESSL library for BLAS and LAPACK functions. Each JUQUEEN node (16 physical cores) is capable of running 64 threads at the same time (since each core is 4-way simultaneously multithreaded). For the hybrid MPI+OpenMP case, we have used all those 64 threads, but for the MPI only version, due to the resulting limited amount of memory per process, we have used only 16 MPI ranks per node.

2.3 Energy consumption

The Cray supercomputer operated by HLRN provides users and system administrators with extensive tools to comprehensively measure and monitor the energy consumed by the programs and the system [17]. Power consumption is sampled periodically and is monitored at the blade level with each blade reporting usage of the nodes and the network. Data is aggregated by the cabinet controllers and reported out-of-band to the system management workstation where it is logged for subsequent processing. Information on the allocation of nodes to jobs and the CPU time is logged at the same time, enabling energy consumption to be added to the data collected for each job. This enables the system administrators to measure and account for the energy efficiency of the system on production workload and the project partners to make power consumption measurements and optimizing for both performance and power consumption. With the help of power management enabled compute nodes, users are also able to easily access the per-node power and energy data in real-time in a set of virtual files. These counters are updated at 10 Hz frequency, however polling them causes a system interrupt that may affect the user code, so their use is not recommended for high-frequency performance monitoring. We have used the same approach for measuring the energy measurement of the sparse kernels as in Refs. [18,19], and recorded the total energy used per node before and after each calculation.

3 RESULTS

In the following we present the measurements for the performance and energy consumption of the sparse linear algebra kernels.

3.1 Single Node of Cray XC30/40

The results of the execution of the benchmarks for several sparse kernels on a single node of both Cray XC30 and XC40 systems using Epetra and Tpetra packages are shown in Figure 1. Performance from using all 24 cores of both MPI and OpenMP only versions compiled by different compilers have been shown in the plots. The GNU compiler surprisingly outperforms other compilers, with Cray compiler catching up in a few cases. Furthermore, the MPI only version performs the best. Tpetra (with help of Kokkos) incorporates better use of cache locality leading to remarkably better performances compared to Epetra. For the shared memory case, Kokkos is also able to manage POSIX threads directly, and achieving better performance in some cases compared to OpenMP.

Figure 2 corresponds to the same calculation but from the point of view of the energy consumed. MPI only version of Tpetra wins regarding the lowest energy consumption (mostly due to having less total run time). GNU and Cray compiled executables also consume the lowest power, signifying that the Cray compiler is heavily optimized for low-energy consumption for the CPUs used for the Cray systems.
Figure 1. The performance of various sparse linear algebra kernels on one node of Cray XC30 (left) and Cray XC40 (right) using the Epetra (top) and Tpetra (bottom) packages.

Figure 2. The energy consumption of kernels. The legend and the order of plots are the same as Figure 1.
To see the performance using the shared memory parallelism, we have also run the benchmarks with varying number of threads. Figure 3 demonstrates the scalability of the sparse matrix-multivector (of length 8) multiplication kernel on a single node of both Cray XC systems. The code uses threads from 1 to 24 (maximum per node).

![Figure 3. The performance of SPMM8. The order of plots is the same as Figure 1.](image)

The Tpetra version of the benchmark compiled with GNU compiler using only MPI on the Haswell CPU is the winner in overall performance, with the Cray provided library being a close second. This is a hint to show that migrating from Epetra to Tpetra (even in case of not using shared memory parallelism) improves the performance. In the case of Epetra benchmark, the intrinsic support of CASK (included in Cray LibSci) leads to better performance.

It is also interesting to look at the energy consumption of the same operation in Figure 4. MPI only version of Tpetra again wins regarding the lowest energy consumption. Obviously allocating more cores decreases the energy consumption of the node, but with using more than half the cores, the total energy consumed by the node does not change considerably.

### 3.2 Many nodes

Here we present the performance results of running the benchmarks in parallel on the Cray XC30 system in Hannover. We have used the Cray provided library as well as our own installation, and using MPI only and MPI+OpenMP for both benchmarks. In the former case, all the cores within a (single) node have been assigned one MPI rank, and in the latter, one MPI rank is used per node with 24 OpenMP threads. Based on the results from single node performance, we omitted using the Intel compiler. The results of the weak scaling is presented in Figure 5. For Epetra, the Cray provided library performs the best, while the Tpetra delivers better overall performance using GNU compiler. The last point in the plot corresponds to using 1600 nodes.
(38,400 cores), which is the largest partition in the Hannover system combining both Cray XC30 and XC40 systems. For that, the executable compiled for the Ivy Bridge CPUs has been used for all the nodes. As expected and also similar to the single node performance results, the MPI+OpenMP hybrid parallel benchmark is falling behind.

Figure 4. The energy consumption of SPMM8. The order of plots are the same as Figure 1.

Figure 5. The weak scaling plot of SPMM8 using the Epetra (left) and Tpetra (right). The solid (dashed) line belongs to the results from the Cray XC30 (XC40) system.
The MPI only Epetra version scales well and behaves in accordance with a single-node performance model. However, Tpetra performs remarkably better than Epetra, therefore, this can be seen as practically enough reason to suggest the migration from Epetra to Tpetra. With this move, support for shared memory parallelism also comes for free.

### 3.3 Comparison of Cray XC30 and IBM Blue Gene/Q

We have compared the performance of the Ivy Bridge CPUs with the PowerPC A2 processors for the exact same calculation on a similar number of physical cores on both systems. Table 1 contains the details of the execution of benchmarks.

<table>
<thead>
<tr>
<th>Parallelism</th>
<th># Nodes</th>
<th># MPI Ranks per Node</th>
<th># Threads per MPI Rank</th>
<th># Cores</th>
<th>Epetra</th>
<th>Eff. %</th>
<th>Tpetra</th>
<th>Eff. %</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Blue Gene/Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td>64</td>
<td>16</td>
<td>1</td>
<td>256</td>
<td>72.2</td>
<td>0.6</td>
<td>193</td>
<td>1.5</td>
</tr>
<tr>
<td>MPI+OpenMP</td>
<td>64</td>
<td>16</td>
<td>4</td>
<td>1024</td>
<td><strong>216</strong></td>
<td><strong>1.6</strong></td>
<td><strong>460</strong></td>
<td><strong>3.5</strong></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>8</td>
<td>8</td>
<td>1024</td>
<td>206</td>
<td>1.6</td>
<td>438</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4</td>
<td>16</td>
<td>1024</td>
<td>190</td>
<td>1.4</td>
<td>414</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>1</td>
<td>64</td>
<td>1024</td>
<td>131</td>
<td>1.0</td>
<td>295</td>
<td>2.3</td>
</tr>
<tr>
<td>Cray XC30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td>64</td>
<td>16</td>
<td>1</td>
<td>1024</td>
<td><strong>1,470</strong></td>
<td><strong>5.0</strong></td>
<td><strong>2,210</strong></td>
<td><strong>7.5</strong></td>
</tr>
<tr>
<td>MPI+OpenMP</td>
<td>48</td>
<td>6</td>
<td>4</td>
<td>1152</td>
<td>756</td>
<td>3.4</td>
<td>1,060</td>
<td>4.8</td>
</tr>
<tr>
<td></td>
<td>48</td>
<td>3</td>
<td>8</td>
<td>1152</td>
<td>744</td>
<td>3.4</td>
<td>921</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>1</td>
<td>16</td>
<td>1024</td>
<td>603</td>
<td>2.0</td>
<td>590</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>1</td>
<td>24</td>
<td>1536</td>
<td>614</td>
<td>2.1</td>
<td>558</td>
<td>1.9</td>
</tr>
</tbody>
</table>

Table 1: Comparison of Cray XC30 and IBM Blue Gene/Q benchmarks. Performance is in GFLOP/s and the efficiency percentage is the ratio to the peak performance. The topmost performance is shown in boldface.

The results are from the same matrix problem on both systems using different parallelization techniques. The topmost FLOP/s in each system is denoted in bold. The same data has been shown in Figure 6, with the percentages displaying the ratio of the performance of the Blue Gene/Q to the Cray XC30 system (for a comparable total number of cores). We have observed that the Intel CPUs are more efficient and outperform PowerPC A2s for sparse matrix vector operations (despite consuming more power). Unlike the Cray system, hybrid parallelization leads to much better performance compared to the MPI only case. In practice, it is possible to fill up all four threads of each core of the A2 processor with MPI processors but the memory overhead from the MPI library limits the performance of the benchmark. On the other hand, the thread management overhead from using only OpenMP per node decreases the performance. Also, the Blue Gene/Q chip is able to use idle threads for asynchronous progress on MPI. We have found that one MPI rank per core and four OpenMP threads gives the best performance. Also the performance improvement of Tpetra over Epetra kernels is significantly larger compared to the Cray system.
4 CONCLUSIONS

In this article, we have surveyed the fundamental packages of Trillions for linear algebra operations, namely Epetra and Tpetra, regarding their real-life performance. We performed several sparse linear algebra operations using two packages on leading supercomputing systems, Cray XC30/40 and IBM Blue Gene/Q. The results from different compilers with tuned libraries have been compared suggesting that the MPI only Tpetra benchmark on the Cray XC system is getting better performance and efficiency. We have also reported results for actual energy consumed by the processors of the Cray system during the run time of the sparse algebra calculations.

Using different hardware architectures and software parallelism models, our results present examples in order to help decision making in incorporation of Trilinos in other scientific applications. The Tpetra package and its underlying package Kokkos show much improved performance and functionality over its counterpart standard linear algebra package, Epetra. We strongly advise the developers to plan accordingly in order to refactor their code base into Tpetra as well using other packages of Trilinos which are going through the same path. By migrating from Epetra to Tpetra and from internal OpenMP development (or similar programming models) to Kokkos, one can expect short term benefits (such as increase in performance) as well as opening the path for using extended set of architectures. Kokkos can also help in reaching more portability (along with performance portability) in shared memory parallelism.

5 ACKNOWLEDGEMENTS

This work has been funded by the European Research Council under the FP7 NUMEXAS project under grant agreement 611636. The authors gratefully acknowledge the Gauss Centre for Supercomputing (GCS) for providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS share of the supercomputer JUQUEEN [16] at Jülich Supercomputing Centre (JSC). GCS is the alliance of the three national supercomputing centers HLRS (Universität Stuttgart), JSC (Forschungszentrum Jülich), and LRZ (Bayerische Akademie der Wissenschaften), funded by the German Federal Ministry of Education and Research (BMBF) and the German State Ministries for Research of Baden-Württemberg (MWK), Bayern (StMWFK) and Nordrhein-Westfalen (MIWF). The authors would like to thank the anonymous referee for the comments.
REFERENCES


