Performance Evaluation and Analysis of Linear Algebra Kernels in the Prototype Tianhe-3 Cluster

Xin You, Hailong Yang(✉), Zhongzhi Luan, Yi Liu, and Depei Qian

Sino-German Joint Software Institute, School of Computer Science and Engineering, Beihang University, Beijing 100191, China
{youxin2015,hailong.yang,zhongzhi.luan,yi.liu,depeiq}@buaa.edu.cn

Abstract. As the supercomputing system entering the exascale era, power consumption becomes a major concern in the system design. Among all the novel techniques for reducing power consumption, ARM architecture is gaining popularity in the HPC community due to its low power footprint and high energy efficiency. As one of the initiatives for addressing the exascale challenges in China, Tianhe-3 supercomputer has adopted the technology roadmap of using the many-core ARM architecture with home-built phytium-2000+ and matrix-2000+ processors. In this paper, we evaluate several linear algebra kernels such as matrix-matrix multiplication, matrix-vector multiplication and triangular solver with both sparse and dense datasets. These linear algebra kernels are good performance indicators of the prototype Tianhe-3 cluster. Comprehensive analysis is performed using roofline model to identify the directions for performance optimization from both hardware and software perspectives. In addition, we compare the performance of phytium-2000+ and matrix-2000+ with widely used KNL processor. We believe this paper provides valuable experiences and insights as work-in-progress towards exascale for the HPC community.

Keywords: Exascale · Performance evaluation and analysis · Roofline model · Tianhe-3 cluster

1 Introduction

Evolving the supercomputing towards the exascale still remains an open challenge for the entire HPC community. Although the technical roadmap varies within the community, there is a consensus that the power consumption must be constrained for the next generation supercomputer to be practically sustainable. For instance, the US Department of Energy Exascale Initiative Steering Committee establishes a 20 MW power budget for the exascale supercomputer [27]. Among the innovative approaches that have been exploited to achieve such power efficiency at large scale, the ARM architecture has drawn the attention of the

D. Abramson and B. R. de Supinski (Eds.): SCFA 2019, LNCS 11416, pp. 86–105, 2019.
https://doi.org/10.1007/978-3-030-18645-6_6

HPC community for its merit of lower power consumption yet competitive performance. Benchmarks have been evaluated to show the effectiveness of using ARM based processors for scientific applications under power constraint [20,25,26]. In addition, experimental clusters have been built with scientific benchmarks evaluated to demonstrate the feasibility of using ARM based processors for constructing supercomputers [23,24]. Therefore, ARM based solutions have already shown their potential to achieve the power efficiency towards exascale.

Among the exascale initiatives in China, Tianhe-3 has adopted the ARM based many-core architecture roadmap using home built phytium and matrix processors. Especially, matrix-2000 processor has already demonstrated its capability for performance acceleration on the previous generation supercomputer Tianhe-2A [9]. Recently, the supercomputing team for Tianhe-3 has opened a prototype Tianhe-3 cluster built upon phytium-2000+ (FTP) and matrix-2000+ (MTP) processors to the public for performance evaluation. This paper takes this rare opportunity to perform comprehensive evaluation of the prototype Tianhe-3 cluster and report the evaluation results as work-in-progress for the HPC community towards exascale.

During the performance evaluation, we use several important linear algebra kernels such as matrix-matrix multiplication, matrix-vector multiplication and triangular solver with both dense and sparse datasets. These linear algebra kernels serve as the fundamental building blocks not only for scientific applications such as computational fluid dynamics (CFD) [12] and molecular dynamics (MD) [22], but also for emerging applications such as graph computing [14] and deep neural networks [13]. We also compare the performance of FTP and MTP processors with widely adopted Intel KNL processor [28] quantitatively. We hope the evaluation results and roofline model analysis from this paper serve in two folds. On one hand, it reveals the architecture designs that are important to achieve the exascale performance with limited power budget for hardware architects. On the other hand, it highlights the factors that software developer should take into consideration for writing efficient code on the near future exascale supercomputers.

Specifically, this paper makes the following contributions:

- We provide a comprehensive performance evaluation of the prototype Tianhe-3 cluster that uses ARMv8-based many-core FTP and MTP processors with important linear algebra kernels.
- We compare the performance of the FTP and MTP processors with their industry counterpart Intel KNL many-core processor, which reveals the strengths and weaknesses among these architecture designs.
- We build roofline models for FTP, MTP and KNL processors to understand the limiting factors that impact the performance of these linear algebra kernels and highlight the directions for performance optimization.

The remainder of this paper is organized as follows. In Sect. 2, we describe the background of our evaluation, including the mathematics of the linear algebra kernels as well as the specifications of the prototype Tianhe-3 cluster. Section 3 presents the evaluation results on both single node FTP and MTP processor

as well as at cluster scale. In addition, we compare the performance results on both FTP and MTP processors with Intel KNL processor. In Sect. 4, we build the roofline models to better understand the evaluation results and identify the directions for performance optimization. The related work is illustrated in Sect. 5. We conclude this paper in Sect. 6.

2 Background

2.1 Linear Algebra Kernels

Matrix-Matrix Multiplication. GEMM (General Matrix-Matrix Multiplication) is the most commonly used linear algebra kernel in scientific applications. As shown in Fig. 1(a), the GEMM routine can be described as Eq. 1, where A, B and C are Matrices with dimensions $(n \times k)$, $(k \times m)$ and $(n \times m)$, α and β are scalars. As GEMM can reach high arithmetic intensity to stress the processor when matrix size is large enough, it is an ideal benchmark kernel to evaluate the performance of a particular processor. On the other hand, GEMM is also a key kernel for widely used deep neural networks such as AlexNet [13] and ResNet [30]. The performance of GEMM reflects how well these deep neural networks run on the prototype Tianhe-3 cluster.

$$C = \alpha AB + \beta C \tag{1}$$

Matrix-Vector Multiplication. Matrix-vector multiplication can be defined as Eq. 2, where A is a $(n \times m)$ matrix, x and y are vectors of n rows and α, β are scalars. For applications that use sparse matrix, sparse matrix-vector multiplication (SpMV) is proposed to avoid storing and computing redundant zero values to reduce both storage and computation complexity. The computation of SpMV is shown in Fig. 1(b), where matrix A is sparse matrix, x and y vectors are dense. Different storage forms are proposed with different SpMV algorithms, such as CSR [32], CSR5 [15] and ELLPACK [16]. There are several attributes that can describe the property of a sparse matrix, including matrix size n, the number of non-zero values nnz and sparsity nnz/n. The computation challenge of SpMV is the high memory bandwidth demand due to its poor data locality. Therefore, we choose SpMV as a memory-bound kernel to evaluate the prototype Tianhe-3 cluster.

$$y = \alpha Ax + \beta y \tag{2}$$

Triangular Solver. The math form of TRSV (Triangular Solver) is defined as Eq. 3, where L is the triangular matrix and x is the unknown vector to be solved, which has the same shape as the given vector b. Figure 1(c) shows the computation of TRSV where matrix L is non-unit lower triangular matrix. In general, TRSV is less computation intensive as GEMM. However, the computation of TRSV involves strong data dependency, which becomes more difficult to solve

when scaling up to multiple computing nodes. Specifically, TRSV stresses the computation of a single node as well as the interconnect across multiple nodes.

$$Lx = b \tag{3}$$

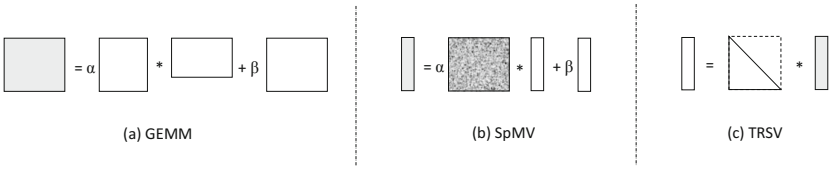

Fig. 1. The computation illustration of linear algebra kernels: (a) GEMM (b) SpMV (c) TRSV. The gray rectangle is the output of the kernel, the white rectangle is dense matrix/vector and the rest is sparse matrix.

2.2 Prototype Tianhe-3 Cluster

The prototype Tianhe-3 cluster is located in Tianjin, China. However, due to the confidentiality agreement, very few technical details about Phytium FT-2000+ (FTP) and MT-2000+ (MTP) processors are released to us. Based on the public reports [9,10,35] as well as the information told by the managing staffs, FTP contains 64 ARMv8 cores, which are organized into eight panels as shown in Fig. 2(a). Each core can run up to 2.4 GHz with the entire processor offering around 614.4Gflops double-precision peak performance and consuming 100 Watts at maximum. Whereas for MTP, it contains 128 ARMv8 cores, which are organized into 4 supernodes as shown in Fig. 2(b). Each core can run up to 2.0 GHz with the entire processor offering around 4.096Tflops double-precision peak performance and consuming 240 W.

During our evaluation, however, the core resources in the prototype cluster are deliberately split at the granularity of 32 cores (one computing node) for both FTP and MTP processors. The reason is that the supercomputing center can offer more computing nodes to serve the demanding evaluation requests in the prototype cluster. The computing nodes are managed and assigned by the batch scheduling system. The user can request the computing node to be allocated as either a FTP node with 32 cores and 64 GB memory or MTP node with 32 core and 16 GB memory. Both FTP and MTP nodes are running Kylin 4.0-1a OS with kernel v4.4.0.

The interconnect in the prototype cluster is built by the National University of Defense Technology (NUDT) that provides 200 Gbps bi-directional bandwidth. The distributed storage nodes are managed by Lustre that provides the shared file system for the prototype cluster. For the compile environment, both GCC v4.9.3 and v4.9.1 as well as a customized MPICH v3.2.1 are supported. The prototype cluster also supports widely used libraries such as BLAS and

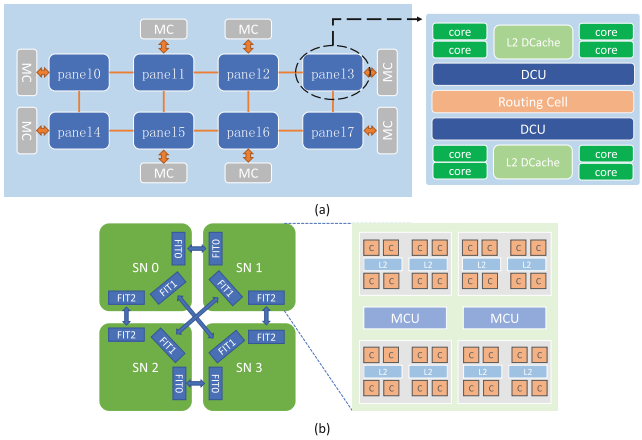

Fig. 2. The architecture of (a) FT-2000+ processor and (b) MT-2000+ processor.

Boost. Therefore, it is very smooth for most of the scientific applications to be ported to run on the prototype cluster. The available hardware and software specifications of the prototype cluster are listed in Table 1.

Table 1. The available hardware and software specifications of the prototype cluster.

Specifications		FT-2000+	MT-2000+
Hardware	Nodes	128	512
	Cores in a node	32	32
	Frequency	2.4 GHz	2.0 GHz
	Memory	64 GB	16 GB
	Interconnect bandwidth	200 Gbps	
Software	OS	Kylin 4.0-1a OS with kernel v4.4.0	
	File system	Lustre	
	MPI	MPICH v3.2.1	
	Compiler	GCC v4.9.1/v4.9.3	
	Supported libraries	Boost, BLAS, OpenBLAS, Scalapack, etc.	

3 Evaluation

3.1 Experimental Setup

To evaluate the linear algebra kernels in the prototype cluster, we choose the widely used library implementations whenever possible. In addition, we also

choose open source implementations that are highly rated in the literature. We explicitly choose the dense and sparse implementations since they use different optimization strategies and stress different aspects of the processor. The selection of linear algebra kernels is detailed in Table 2.

Table 2. Linear algebra kernels under evaluation.

Platforms	FTP			MTP			KNL		
Kernels	GEMM	TRSV	SpMV	GEMM	TRSV	SpMV	GEMM	TRSV	SpMV
Openblas [34]	✓	✓		✓	✓				
Intel MKL [31]							✓	✓	✓
Scalapack [2]	✓	✓	✓	✓	✓	✓			
CSR [32]			✓			✓			
distributedSpMV [11]			✓			✓			

For the datasets, we generate the dense square matrices ($N \times N$) with random double-precision values. We scale the dense matrices from $N = 32$ to $N = 6400$ to see how they affect the processor performance at scale. For the sparse matrices, we use the 20 square matrices from the popular Florida Sparse Matrix Collection [6]. These sparse matrices are representative of a wide variety of application domains such as graphic computing and scientific application. The unique characteristics of each sparse matrix are listed in Table 3.

We evaluate the linear algebra kernels on a single node as well as across multiple nodes with both FTP and MTP processors. For both FTP and MTP processors, we use up to 64 nodes (2048 cores) at the largest scale that we can apply. For comparison, we also evaluate the Intel KNL many-core processor Xeon Phi 7210 that contains 64 cores with each running at 1.3 GHz. We use the MKL libraries on KNL that are highly optimized for the linear algebra kernels on Intel architecture. We use the flat mode of the hybrid memories on KNL and allocate the data on the High Bandwidth Memory (HBM), which provides higher bandwidth for memory accesses and thus better performance. OpenMP and MPI are used as the parallel execution models during the evaluation.

3.2 Performance Comparison on Singe Node

The evaluation of each processor using specific kernel implementation is shown in Table 2. To measure the performance of a single node, we utilize all the cores to run the kernels on each particular processor. Specifically, we run 32 threads on FTP and MTP node respectively, whereas 64 threads on KNL. Figure 3 shows the box plot of the single node performance when running GEMM, TRSV and SpMV on FTP, MTP and KNL respectively. We can see that KNL achieves the best average performance across all three kernels. For dense kernels such as GEMM, KNL achieves 6.8× and 14.0× performance speedup compared to FTP and MTP respectively. The large performance gap of GEMM on KNL compared to FTP

Table 3. The sparse matrix datasets under evaluation.

Matrix shape	Matrix	row×col	nnz	nnz/row
	G24	2K × 2K	39.9K	19.9
	windtunnel_evap2d	8K × 8K	109K	13
	vsp_c-30_data_data	11K × 11K	124K	11
	TEM152078	15K × 15K	6.5M	42
	EAT_RS	23K × 23K	325K	14
	epb2	25K × 25K	175K	7
	cit−HepTh	27K × 27K	352K	13
	invextr1_new	30K × 30K	1.8M	59
	ship_001	35K × 35K	3.9M	112
	onetone1	36K × 36K	335K	9
	bcsstk32	44K × 44K	2.0M	45
	venkat01	62K × 62K	1.7M	28
	nd24k	72K × 72K	28.7M	398
	ifiss_mat	96K × 96K	3.6M	37
	barrier2−10	115K × 115K	2.1M	18
	torso1	116K × 116K	8.5M	73
	scircuit	171K × 171K	959K	6
	offshore	259K × 259K	4.2M	16
	ASIC_680ks	682K × 682K	1.7M	2.5
	thermal2	1.2M × 1.2M	8.6M	7

and MTP is due to the limited core count assigned for each computing node in the prototype Tianhe-3 cluster. For instance, on both FTP and MTP computing nodes, there are only 32 cores. Whereas on KNL, there are 64 cores available. Large core count clearly gives an advantage from the performance perspective. It is also noticed from Fig. 3(a) and (b), FTP achieves better performance than MTP for the dense kernels (GEMM and TRSV). This is because, although FTP and MTP node contain the same core count (e.g., 32), the cores in FTP run at higher frequency (e.g., 2.4 GHz) than the cores (e.g., 2.0 GHz) on MTP.

For the sparse kernel such as SpMV, the performance gap of FTP and MTP processor compared to KNL becomes even larger as shown in Fig. 3(c). The average performance of SpMV on KNL is 15.4× and 16.6× better than on FTP and MTP respectively. It is well understood that the performance of SpMV is

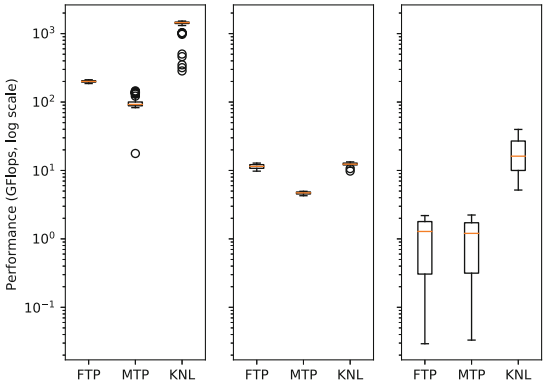

Fig. 3. The performance comparison among FTP, MTP and KNL running linear algebra kernels of (a) GEMM, (b) TRSV and (c) SpMV.

bounded by the memory bandwidth due to its nature of poor data locality. Therefore, the core count as well as the core frequency should not be the dominating factors for the performance disparity among the processors. We believe the performance advantage of KNL can be partially attributed to the high bandwidth memory (HBM) integrated into the processor, which provides much higher memory bandwidth than FTP and MTP that use the traditional DRAM. In addition, the MKL provides highly optimized SpMV implementation that leverages the powerful vectorization capability of KNL through AVX512 instructions, which achieves tremendous speedup of SpMV. In contrast, the capability of vectorization on FTP and MTP is quite limited compared to KNL. Recent work [5] even claims vectorization of SpMV on FTP provides no performance benefit if not slow down. In general, the low memory bandwidth as well as the limited vectorization of FTP and MTP hurt their ability to deliver comparable performance of SpMV to their counterpart KNL.

3.3 Scalability Comparison

In order to compare the performance scalability of the kernels on different processors, we scale the kernel execution on both a single node and across multiple nodes. For single node scalability, we run each kernel from 1 to 32 threads on FTP and MTP, whereas from 1 to 64 threads on KNL. The speedup of each kernel is compared to the single thread execution. Figure 4(a) shows the single node scalability of GEMM on these three processors. We can see that GEMM reaches good scalability on a single node with maximum speedup of 23.8× on FTP, 20.3× on MTP and 42.7× on KNL. The huge speedup achieved by GEMM when scaling on KNL can be attributed to the large core count compared to FTP and MTP. For TRSV, the scalability on KNL starts to drop beyond 32 threads. The maximum speedup achieved on FTP, MTP and KNL is 6.9×, 3.3× and 3.8× respectively as shown in Fig. 4(b). However, the absolute performance on

KNL is always better than FTP and MTP at all scales. For SpMV, the scalability of FTP and MTP is extremely poor. The maximum speedup of SpMV is 2.4× and 2.7× on FTP and MTP respectively when utilizing half of the cores as shown in Fig. 4(c). In contrast, KNL scales well and reaches maximum speedup of 30.1× when all cores are fully utilized. Since SpMV is memory bounded, the good scalability is primarily due to the high bandwidth memory (HBM) on KNL that offers 400+ GB/s bandwidth compared to FTP and MTP that use DRAM for quite limited bandwidth.

For the scalability across multiple nodes, we run each kernel from 1 to 64 computing nodes with each node fully utilized (e.g., running 32 threads). We do not include the results of multiple KNL nodes since we only have one KNL node available. The speedup of each kernel is compared to the single node execution. Figure 5(a) shows that the performance speedup of GEMM starts to drop on FTP when the number of nodes scales beyond 32. Therefore, MTP has better scalability when running GEMM compared to FTP. However, the absolute performance of GEMM is, on the contrary, better on FTP even scaling beyond 32 nodes. For TRSV shown in Fig. 5(b), the performance speedup starts to drop on both FTP and MTP processor when the number of nodes scales beyond 32. The maximum speedup is 3.5× and 5.7× when running on 32 nodes of FTP and MTP respectively. For SpMV, the maximum performance speedup is 1.8× on FTP with 8 nodes and 5.8× on MTP with 32 nodes as shown in Fig. 5(c). The scalability of FTP is much worse than MTP, where the performance speedup starts to drop beyond eight nodes.

4 Discussion

4.1 Building the Roofline Model

To better understand the evaluation results on FTP, MTP as well as KNL, we build the Roofline Model [33] to investigate the strengths and weaknesses of each processor architecture. The advantage of the roofline model is that it establishes a quantitative relationship among floating-point performance, operational intensity and memory performance using a 2D graph, which captures the intrinsic characteristics of hardware and software designs. Using the roofline model, it is easy to reveal the performance upper bound on each processor. The *roof* in the roofline model indicates the peak performance of the processor, whereas the *slope* indicates the peak memory bandwidth. The x axis measures the operational intensity of the program under evaluation, and the y axis indicates the attainable performance (GFlops). Depending on whether the column of the operational intensity hits the flat part of the roof, we can easily identify whether the program under evaluation is compute-bound or memory-bound.

To obtain the peak floating-point performance of FTP and MTP processors, we scale down the original processor specifications [9,10,35] proportional to the core count of a compute node in the prototype cluster. For KNL, we provide the theoretical peak floating-point performance from processor specifications. To

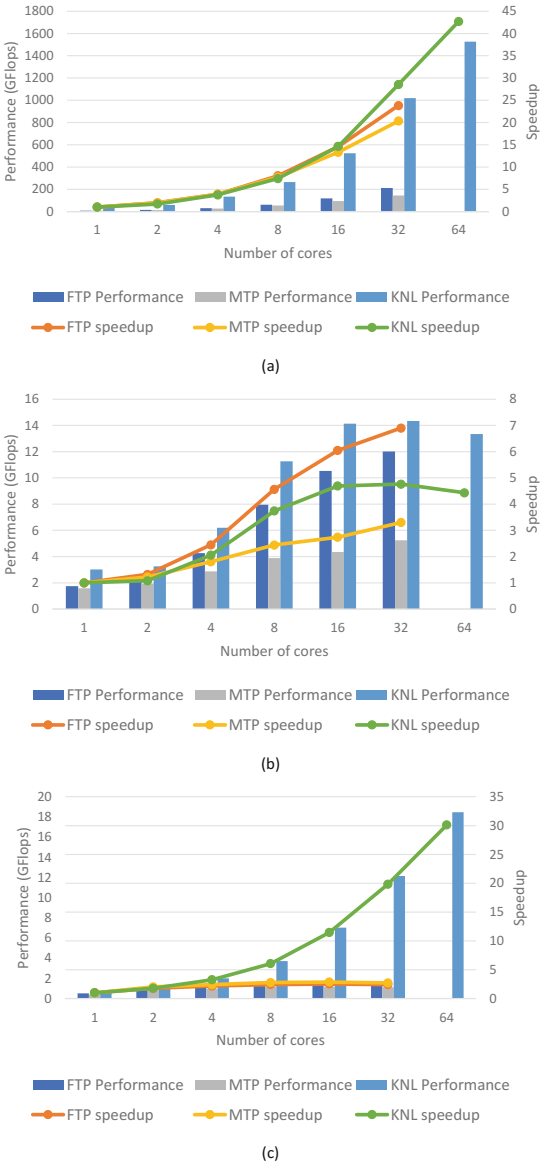

Fig. 4. Scalability of (a) GEMM, (b) TRSV and (c) SpMV on a single node.

obtain the peak memory bandwidth, we use *STREAM* benchmark [17] to measure the three processors directly. We also add multiple ceilings to the roofline model by using different optimizations. For instance, we add one memory ceiling by using the memory affinity optimization and several compute ceilings by using thread-level parallelism (TLP), instruction-level parallelism (ILP) as well

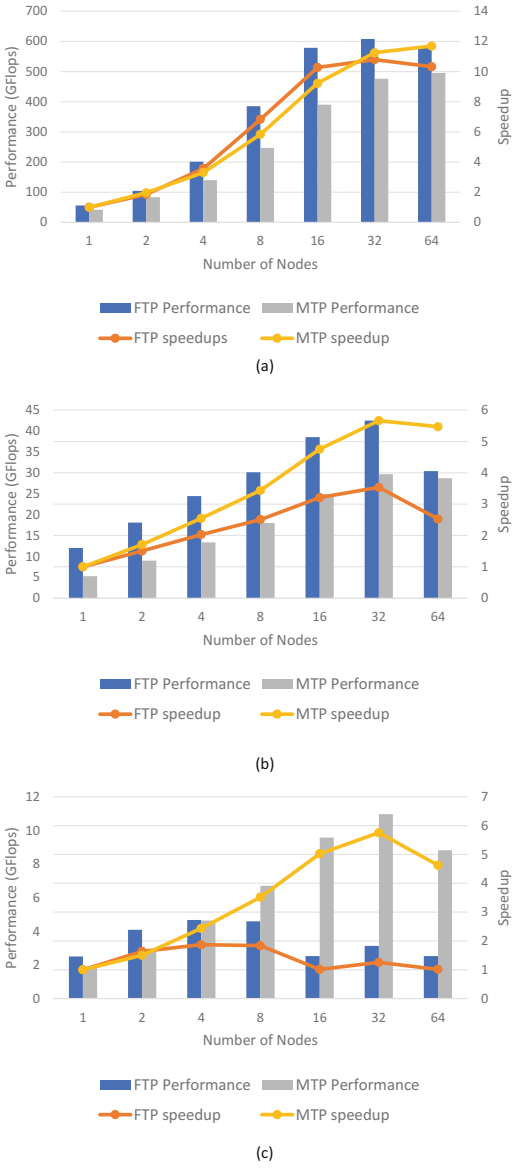

Fig. 5. Scalability of (a) GEMM, (b) TRSV and (c) SpMV across multiple nodes.

as SIMD instructions. These ceilings in the roofline model are intuitive to guide the directions for performance optimization.

$$OperationalIntensity = Flops/Bytes \tag{4}$$

Table 4. The formulas [21] for calculating operational intensity of evaluated kernels, where n is the size of matrix, nnz is the number of non-zero values in sparse matrix.

Kernel	Flops	Data movement (bytes)	Operational intensity
GEMM	$2n^3$	$4n^2 * 8$	$n/16$
TRSV	n^2	$n(n+1)/2+n$	$2n/(n+3)$
SpMV	$2nnz$	$4*(n+1+nnz)+8*(2n+nnz)$	$nnz/(6n+2+10nnz)$

To measure the operational intensity, we calculate the flops and data movements of each kernel based on the given input. Generally, the operational intensity is calculated as shown in Eq. 4, where *Flops* is the number of floating point operations and *Bytes* are the total bytes of data movements from DRAM. The formulas for calculating *Flops*, *Bytes* and *OperationalIntensity* for the evaluated kernels are shown in Table 4. Specifically, *Data Movement* differs when using different implementations, therefore we use the theoretical minimal of *Data Movement*, assuming all data can be fully reused. The results shown in Figs. 6, 7 and 8 are evaluated against different kernels with different inputs running on each of the three processors.

4.2 Insights for Software Optimization

As shown in Fig. 6, the kernel GEMM achieves the highest operational intensity across all three kernel, which is consistent with its nature of high intensity of arithmetic operations. It is also clear that GEMM is compute-bound on FTP. Since GEMM is usually one of the highly optimized kernels in modern linear algebra libraries, it is quite close to the theoretical ceiling of FTP especially when the matrix size scales. Therefore, there is not too much opportunity from

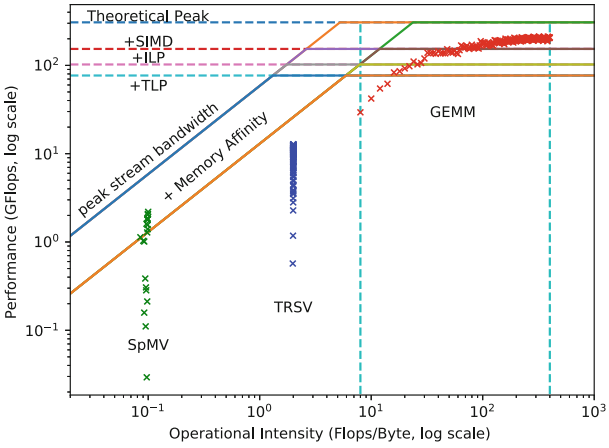

Fig. 6. The roofline model of FTP.

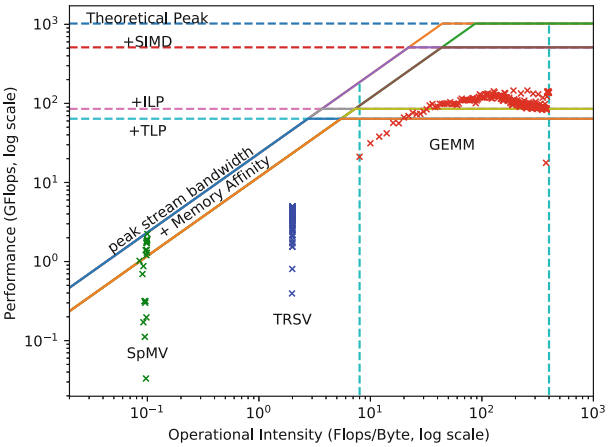

Fig. 7. The roofline model of MTP.

the software perspective for future performance optimization unless more cores are added or the frequency of each core is increased. However, developers still need to consider the memory affinity when the matrix size is small. Otherwise, the performance of GEMM could be bounded by the memory indicated by the lower memory ceiling in Fig. 6.

TRSV and SpMV show much lower operational intensity on FTP compared to GEMM. In addition, the performance of both TRSV and SpMV is memory bounded as shown in Fig. 6. Especially for SpMV, the operational intensity is the lowest among all three kernels due to its poor data locality. Different from GEMM where the operational intensity covers a wide range as the matrix size

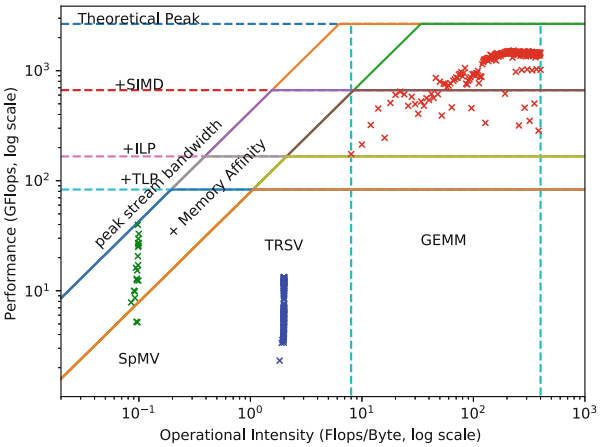

Fig. 8. The roofline model of KNL.

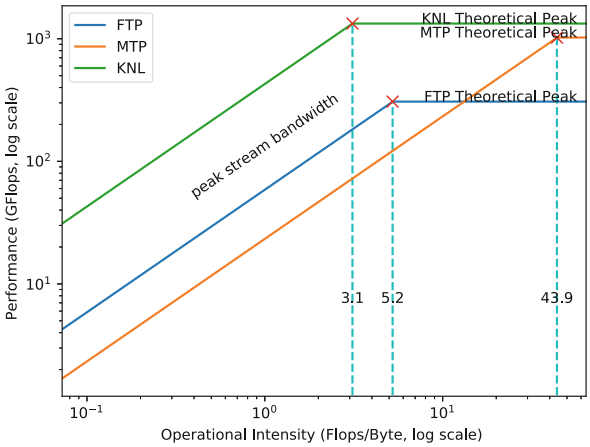

Fig. 9. The performance comparison of different processors using roofline model.

scales, the operational intensity of both TRSV and SpMV converges when the matrix size is large enough. As shown in Fig. 6, the performance of both TRSV and SpMV is still bounded by the lower memory ceiling (e.g., memory affinity). Therefore, using the memory node close to the computation could benefit the performance of both TRSV and SpMV on FTP.

Note that memory affinity is an important factor to achieve better performance on FTP. As shown in Fig. 2(a), the cores in FTP are organized into several panels and each panel has a local memory node attached. Therefore, the developers should pay special attention to the memory affinity when writing applications on FTP in case of bounding by the lower memory ceiling. Another interesting thing we can notice from Fig. 6 is that the computing ceilings (e.g., TLP, ILP and SIMD) are quite near to each other, which means applying a single optimization on FTP could not increase the performance significantly. However, there is still a large performance space between the ceiling of TLP and theoretical peak. Therefore, it still worths the effort to optimize the applications from computation aspect on FTP.

Although the performance trends of TRSV and SpMV on MTP are similar (e.g., memory-bound) to FTP, the behavior of GEMM is somehow different as shown in Fig. 7. Half of the cases, the performance of GEMM is memory-bound. When the operational intensity is high enough, it becomes compute-bound. However, we notice when GEMM becomes compute-bound, its performance starts to drop. The reason for this interesting trend of GEMM can be explained that when the operational intensity is low (e.g., small matrix size), the performance is bounded by the limited memory bandwidth (e.g., 16 GB on MTP compared to 64GB on FTP). As the operational intensity increases, the insufficient computing capacity (e.g., 2.0 GHz on MTP compared to 2.4 GHz on FTP) prevents GEMM from achieving higher performance.

We also notice that the performance space between the ceiling of TPL and the theoretical peak is quite large in Fig. 7. The SIMD instructions are wider on MTP than FTP. The wider SIMD instructions on MTP indicate a large performance opportunity if the application can vectorize its computation on MTP. The computation of GEMM itself fits well for vectorization. Therefore, how to leverage the SIMD instructions on MTP should be the direction for further performance optimization of GEMM from the software perspective.

The computing ceilings are quite far from each other as shown in Fig. 8. The similar trend is also observed with memory ceilings. This indicates performance optimizations are indispensable for applications to run efficiently on KNL, especially for TRSV, which achieves even worse performance than SpMV in many cases. Two potential directions for improving the performance of TRSV on KNL are *(1)* breaking the memory ceiling by leveraging the memory affinity, and *(2)* breaking the TLP ceiling by exposing sufficient parallelism. To break the memory ceiling, exploiting the unique high bandwidth memory (HBM) on KNL should benefit the performance by providing higher memory bandwidth. Whereas to break the ILP ceiling, loop unrolling and reordering should be applied to increase the instruction parallelism.

4.3 Insights for Hardware Optimization

Obviously shown in Fig. 9, KNL delivers the highest performance compared to FTP and MTP due to its large number of cores and wider SIMD units. Therefore, to approach exascale, sufficient core count and powerful vectorization is essential for the future architecture improvement on both FTP and MTP. Another interesting observation is that the ridge point of KNL is more left than FTP and MTP in the roofline model. The ridge point indicates the minimum operational intensity required to achieve the peak performance. Therefore, the more left the ridge point is, the fewer restrictions there are for application to reach the peak performance on the processor. For instance, the ridge points for KNL, FTP and MTP are 3.1, 5.2 and 43.9 Flops/Byte respectively, which means MTP is the most difficult processor for developers and compiler writers to produce high-performance programs. To improve the productivity on the future exascale supercomputer, reducing the operational intensity of both FTP and MTP benefits from all types of software optimizations. In addition, the diagonal line of KNL is also much higher than FTP and MTP, which means KNL provides much higher memory bandwidth than the other two processors. This can be attributed to the adoption of high bandwidth memory (HBM) in KNL that application can leverage by expressing the memory affinity. Integrating the traditional DRAM with novel memory technologies such as HBM could be another hardware optimization for FTP and MTP in order to eliminate the potential memory bound.

5 Related Work

5.1 Performance Optimization of Linear Algebra Kernels

Linear Algebra Kernels such as GEMM, TRSV and SpMV are widely used in scientific computing and machine learning. Many optimization works are focused on these Linear Algebra Kernels to gain full advantage of specific architectures. For dense matrix multiplication and solvers, BLAS gives an overall interface for all kinds of linear algebra kernels. OpenBLAS [34] is an open source implementation of BLAS interface with optimization of thread parallelization and blocking techniques. Scalapack [2] is also available in Tianhe-3 prototype cluster for scaling the BLAS interface to the distributed cluster. Intel Math Kernel Library (MKL) [31] is specially designed for x86 processors and by using parallelization, vectorization, blocking and other specified optimizing techniques, it reaches a notable performance gain than many other open source libraries.

In the case of sparse matrix-vector multiplication, Liu and Vinter proposed new sparse matrix storing format CSR5 [15], a SIMD-friendly format for efficient computations of SpMV. Their approach can make SpMV kernel more SIMD friendly and ease to parallel and thus can gain performance speedup compared to MKL. They also developed CSR5-based SpMV algorithm on AMD and NVIDIA GPU which has better average performance than other existing formats. On the other hand, a thread-level parallel algorithm called merge-based SpMV [19] also claims to have great speedup for multi-core processors. BML [4] is an open source distributed library which supports for both dense and sparse matrix multiplication. They support for both ELLPACK and CSR format for sparse storage and implemented Gustavson algorithm as well as merge-based algorithm.

5.2 Performance Optimization Techniques on ARM

One optimization techniques on ARM architecture is tuning compilation flags as well as compiler itself to generate more efficient codes. Blackmore et al. [3] developed an auto-tuning method based on a collection of compilation flags used for GNU C compiler on ARM Cortex-M3 processor (CM3). They used a machine learning iterative method to obtain the optimal selections of optimization flags and finally gained two extra collections of compilation flags that outperforms standard *-O3* optimization for CM3 as well as AVR and CA8. On the other hand, Melnik et al. [18] made a case study on *libevas* to evaluate the impact of compiler optimization. They indicate the inefficiency of generated assembly code introduced by GCC's global common subexpression elimination (GCSE). They claim that original GCSE dose not aware whether the constant value will fit into ARM's 8 bit limited immediates. They also find that loop prefetching flags that show performance gains on ARMv6 architectures are not working well on ARMv8 based Cortex-A8 processor. They indicate that tuning with specific architecture's parameters for prefetching flags will gain as much as 20% performance gain in their evaluation.

Some other ARM-based optimization works are focused in the current ARM's many-core system as well as its SIMD unit called NEON. Bez et al. [1] performed HPC applications on ARMv8 Yggdrasil cluster and analyzed different optimization from time and energy aspects. They mainly reach performance gain from specific ARM compilation flags and NEON optimizations. Besides, Ruiz et al. [26] work on performance analysis and optimization of HPCG benchmark on ARM-based platform. In addition to applying optimal compilation flags and ARM-optimized math libraries, they also report multi-color reordering method and multi-block color reordering method to have less OpenMP thread synchronizations which will improve performance on current many-core ARM architecture. For ARMv8 based FTP processors, Chen et al. [5] benchmarked different formats of sparse matrix storage and developed a prediction model to choose an optimal format of sparse matrix storage of an unknown matrix. They claimed that NUMA-aware optimization on FTP can make notable speedup. They also claimed that vectorizing with NEON on ARMv8-based FTP led to performance loss since there were no efficient *gather* vector operations realized in ARMv8 architecture. Our work focuses on different architecture issues and gives some insights on future designs by benchmarking popular linear algebra kernels while they are interested in how different sparse matrix formats affect the performance on this specific architecture.

As ARM's low power and potentially high performance interest people to use in embedded systems as well as high-performance clusters, ARM developer releases collections of ARM performance libraries including BLAS, LAPACK, FFT and other commonly used math routines [8]. They officially claimed that their library's performance is better than widely-used OpenBLAS library. For machine learnings, they also developed a library called Compute Library [7] which targets Arm Cortex-A family of CPU processors and the Arm Mali family of GPUs. A case study [29] implements deep learning's embedded inference engine with Compute Library and they showed an overall speedup of 25% to Tensorflow.

6 Conclusion

In this paper, we evaluate the prototype Tianhe-3 cluster using representative linear algebra kernels with both dense and sparse datasets. The evaluation results are good performance indicators for assessing both the software and hardware designs as we are moving towards exascale. To better understand the evaluation results, we build roofline models for FTP and MTP processors that reveal the directions for future performance optimizations from the perspectives of both software developers and hardware architects. In addition, we compare the performance of FTP and MTP processors with Intel many-core KNL processor, which highlights the strengths and weaknesses among the architecture designs. We hope this paper can shed the lights on the path pursuing exascale supercomputers by taking the chance to report the work-in-progress of one of China exascale initiatives with Tianhe-3 for the HPC community. For the future work,

we would like to compare with more architectures such as GPU and evaluate ARM high-performance libraries when they become available on FTP and MTP.

Acknowledgments. We would like to thank the National SuperComputer Center in Tianjin for offering us this opportunity to evaluate the prototype Tianhe-3 Cluster. This work is supported National Key R&D Program of China (Grant No. 2017YFB0202202) and National Natural Science Foundation of China (Grant No. 61502019).

References

1. Bez, J.L., Bernart, E.E., dos Santos, F.F., Schnorr, L.M., Navaux, P.O.A.: Performance and energy efficiency analysis of HPC physics simulation applications in a cluster of arm processors. Concurrency Comput.: Pract. Experience **29**(22), e4014 (2017)
2. Blackford, L.S., et al.: ScaLAPACK Users' Guide. SIAM, Philadelphia (1997)
3. Blackmore, C., Ray, O., Eder, K.: Automatically tuning the GCC compiler to optimize the performance of applications running on the ARM cortex-M3. CoRR (2017)
4. Bock, N., et al.: The basic matrix library (BML) for quantum chemistry. J. Supercomput. **74**(11), 6201–6219 (2018)
5. Chen, D., Fang, J., Chen, S., Xu, C., Wang, Z.: Optimizing sparse matrix-vector multiplications on an ARMv8-based many-core architecture. Int. J. Parallel Program. 1–15 (2018)
6. Davis, T.A., Hu, Y.: The university of florida sparse matrix collection. ACM Trans. Math. Softw. (TOMS) **38**(1), 1 (2011)
7. Arm Developer: Compute Library (2018). https://developer.arm.com/technologies/compute-library
8. ARM Developer: Arm performance libraries reference guide. ARM Developer (2018)
9. Dongarra, J.: Report on the TianHe-2a system. Technical report, ICL-UT-17-04, September 2017
10. FT-2000: Phytium Technology Co., Ltd. (2017). http://www.phytium.com.cn/Product/detail
11. hir0shim: Open source implentention of distributed SpMV on GitHuB (2015). https://github.com/hir0shim/distributedSpMV
12. Jacobsen, N.G., Fuhrman, D.R., Fredsøe, J.: A wave generation toolbox for the open-source CFD library: openfoam®. Int. J. Numer. Methods Fluids **70**(9), 1073–1088 (2012)
13. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
14. Langville, A.N., Meyer, C.D.: Google's PageRank and Beyond: The Science of Search Engine Rankings. Princeton University Press, Princeton (2011)
15. Liu, W., Vinter, B.: CSR5: an efficient storage format for cross-platform sparse matrix-vector multiplication. In: Proceedings of the 29th ACM on International Conference on Supercomputing, pp. 339–350. ACM (2015)
16. Liu, X., Smelyanskiy, M., Chow, E., Dubey, P.: Efficient sparse matrix-vector multiplication on x86-based many-core processors. In: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing, pp. 273–282. ACM (2013)

17. McCalpin, J.D.: Stream benchmark, vol. 22 (1995). www.cs.virginia.edu/stream/ref.html#what
18. Melnik, D., Belevantsev, A., Plotnikov, D., Lee, S.: A case study: optimizing GCC on ARM for performance of libevas rasterization library. In: Proceedings of GROW (2010)
19. Merrill, D., Garland, M.: Merge-based parallel sparse matrix-vector multiplication. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 58. IEEE Press (2016)
20. Padoin, E.L., de Oliveira, D.A., Velho, P., Navaux, P.O.: Time-to-solution and energy-to-solution: a comparison between ARM and Xeon. In: 2012 Third Workshop on Applications for Multi-core Architectures (WAMCA), pp. 48–53. IEEE (2012)
21. Peise, E.: Performance modeling and prediction for dense linear algebra (2017). arXiv preprint arXiv:1706.01341
22. Plimpton, S., Crozier, P., Thompson, A.: Lammps-large-scale atomic/molecular massively parallel simulator. Sandia Nat. Laboratories **18**, 43 (2007)
23. Rajovic, N., et al.: The mont-blanc prototype: an alternative approach for HPC systems. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, pp. 444–455. IEEE (2016)
24. Rajovic, N., Rico, A., Puzovic, N., Adeniyi-Jones, C., Ramirez, A.: Tibidabo1: making the case for an ARM-based HPC system. Future Gener. Comput. Syst. **36**, 322–334 (2014)
25. Rajovic, N., Vilanova, L., Villavieja, C., Puzovic, N., Ramirez, A.: The low power architecture approach towards exascale computing. J. Comput. Sci. **4**(6), 439–443 (2013)
26. Ruiz, D., Mantovani, F., Casas, M., Labarta, J., Spiga, F.: The HPCG benchmark: analysis, shared memory preliminary improvements and evaluation on an arm-based platform (2018)
27. Shalf, J., Dosanjh, S., Morrison, J.: Exascale computing technology challenges. In: Palma, J.M.L.M., Daydé, M., Marques, O., Lopes, J.C. (eds.) VECPAR 2010. LNCS, vol. 6449, pp. 1–25. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19328-6_1
28. Sodani, A.: Knights landing (KNL): 2nd generation Intel® Xeon Phi processor. In: 2015 IEEE, Hot Chips 27 Symposium (HCS), pp. 1–24. IEEE (2015)
29. Sun, D., Liu, S., Gaudiot, J.L.: Enabling embedded inference engine with arm compute library: a case study (2017). arXiv preprint arXiv:1704.03751
30. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: AAAI, vol. 4, p. 12 (2017)
31. Wang, E., et al.: Intel math kernel library. In: Wang, E., et al. (eds.) High-Performance Computing on the Intel® Xeon Phi™, pp. 167–188. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06486-4_7
32. Williams, S., Oliker, L., Vuduc, R., Shalf, J., Yelick, K., Demmel, J.: Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC 2007, pp. 1–12. IEEE (2007)
33. Williams, S., Waterman, A., Patterson, D.: Roofline: an insightful visual performance model for multicore architectures. Commun. ACM **52**(4), 65–76 (2009)
34. Xianyi, Z., Qian, W., Saar, W.: OpenBLAS: an optimized BLAS library (2016). http://www.openblas.net/. Accessed 12 May 2016
35. Zhang, C.: Mars: a 64-core ARMv8 processor. In: 2015 IEEE Hot Chips 27 Symposium (HCS), pp. 1–23. IEEE (2015)