The Design of Fast and Energy-Efficient Linear Solvers: On the Potential of Half-Precision Arithmetic and Iterative Refinement Techniques

Azzam Haidar[1(✉)], Ahmad Abdelfattah[1], Mawussi Zounon[4], Panruo Wu[2], Srikara Pranesh[4], Stanimire Tomov[1], and Jack Dongarra[1,3,4]

[1] Innovative Computing Laboratory, University of Tennessee, Knoxville, USA
{haidar,ahmad,pwu11,tomov,dongarra}@icl.utk.edu
[2] University of Houston, Houston, TX, USA
[3] Oak Ridge National Laboratory, Oak Ridge, USA
[4] University of Manchester, Manchester, UK
{mawussi.zounon,srikara.pranesh}@manchester.ac.uk

Abstract. As parallel computers approach exascale, power efficiency in high-performance computing (HPC) systems is of increasing concern. Exploiting both the hardware features and algorithms is an effective solution to achieve power efficiency, and to address the energy constraints in modern and future HPC systems. In this work, we present a novel design and implementation of an energy-efficient solution for dense linear systems of equations, which are at the heart of large-scale HPC applications. The proposed energy-efficient linear system solvers are based on two main components: (1) iterative refinement techniques, and (2) reduced-precision computing features in modern accelerators and coprocessors. While most of the energy efficiency approaches aim to reduce the consumption with a minimal performance penalty, our method improves both the performance and the energy efficiency. Compared to highly-optimized linear system solvers, our kernels deliver the same accuracy solution up to 2× faster and reduce the energy consumption up to half on Intel Knights Landing (KNL) architectures. By efficiently using the Tensor Cores available in the NVIDIA V100 PCIe GPUs, the speedups can be up to 4×, with more than 80% reduction in the energy consumption.

Keywords: FP16 · Tensor cores · Mixed-precision · HPC · Solvers

1 Introduction

As parallel computers approach exascale, power efficiency in high-performance computing (HPC) systems is of increasing concern. Over the last few decades, many challenges in science and engineering have been successfully addressed

Y. Shi et al. (Eds.): ICCS 2018, LNCS 10860, pp. 586–600, 2018.
https://doi.org/10.1007/978-3-319-93698-7_45

thanks to the improving performance of HPC systems. However, it comes at a cost: electrical power consumption. This leads to two main concerns—increase of the power bills beyond affordable budgets, and increasing impact on the environment.

To help mitigate the power constraints in modern and future HPC systems, different approaches have been investigated to assess and reduce the energy consumption of scientific applications. So far, the most promising solution is the intensive use of field-programmable gate arrays (FPGA) and graphics processing unit (GPU) technologies in HPC applications [2]. To reduce the power consumption of HPC applications that still require CPU processors, dynamic voltage and frequency scaling (DVFS) strategies are commonly used [6]. In fact, the two most influential factors on the power consumption of CPU cores are the clock frequencies and the voltages. As a result, most of the energy-efficient strategies focus on DVFS methods with low performance overhead. In this work we propose a new approach to energy efficiency, which, in addition to significantly decreasing the power consumption, radically improves the performance.

Another approach to energy efficiency is to redesign the most time-consuming kernels in HPC applications and provide energy efficient alternatives. In this work we use this approach. Solving linear systems of equations is at the heart of numerical simulations used in HPC application, and is one of the most time-consuming steps. In this work we design a novel energy-efficient algorithm for the solutions of linear system of equations. To that end, we exploit both hardware solutions, such as energy efficient NVIDIA GPUs and Intel Xeon Phis, and algorithmic techniques such as iterative refinement (IR) techniques.

The problem of interest in this work is the solution of linear systems of equations $Ax = b$, where A is a general nonsingular $n \times n$ dense matrix, and b is a general $n \times 1$ vector. In most HPC applications, the input data A and b are stored in double precision, and the solution x is expected in the same precision. The standard method for solving these linear system of equations is via Gaussian elimination. However, the accuracy of the obtained solution using this method is often unsatisfactory because of the round-off errors it generates. The iterative refinement technique, first introduced by Wilkinson [21], aims to improve the accuracy of the computed solution.

The iterative refinement algorithm for solving linear systems consists of the following three steps. First, the computation of the initial solution $\bar{x}$. This step is the most expensive because it consumes $O(n^3)$ floating-point operations (FLOPs). Second, computation of the residual $r = b - A\bar{x}$. This step consumes $O(n^2)$ FLOPs, and checks the accuracy of the computed solution. Finally, the correction d is computed by solving $Ad = r$, and next $\bar{x}$ is updated by $\bar{x} \leftarrow \bar{x} + d$, which also requires $O(n^2)$ FLOPs. The last two steps are iterated until a satisfactory accuracy is achieved. The original iterative refinement algorithm used double precision for the three steps. However, the emergence of multiple-precision floating-point arithmetic units in modern architectures motivated the design of mixed-precision variants.

On modern architectures, single-precision floating-point arithmetic (FP32) is twice as fast as double-precision floating-point arithmetic (FP64). For example, the Intel Knights Landing (KNL) can deliver 3 teraFLOP/s of FP64 performance, but in FP32, it can achieve more than 6 teraFLOP/s. In addition, the latest version of NVIDIA accelerators—the V100 PCIe GPU—provides hardware support for half-precision floating-point arithmetic (FP16). This new V100 PCIe GPU has a peak performance of 7 teraFLOP/s in FP64, 14 teraFLOP/s in FP32, and 112 teraFLOP/s in FP16 using the Tensor Cores. It is then possible to compute the most expensive operation (which is the matrix factorization) in FP32 or FP16, and use FP64 in accuracy refinement iterations. The different implementations of the resulting mixed iterative refinements are summarized in Table 1.

Table 1. Variants of iterative refinement implemented in this work. From left to right, the first column lists the different kernels where the first entry dgesv is the standard method without iterative refinement process. The second and the third columns specify, respectively, the precision used for the factorization and refinement, where TC stands for Tensor Core. In the last two columns, ✓indicates we have implemented for the corresponding architecture, ✗indicates "arithmetic not supported."

Kernel	Factorization	Refinement	KNL	V100
dgesv	FP64	–	✓	✓
dsgesv	FP32	FP64	✓	✓
dhgesv	FP16	FP64	✗	✓
dhgesv-TC	FP16-TC	FP64	✗	✓

2 Contributions

This work aims to respond to the power constraints in modern and future HPC systems through the design and implementation of fast and energy-efficient solvers for linear systems of equations. To this end, our main contributions are:

- The design and implementation of a highly-optimized iterative refinement kernel for Intel KNL architectures. Compared to the standard algorithm (dgesv), our kernel (dsgesv) is up to 2× faster in delivering the same accuracy solution, and reduces the power consumption by half.
- Analysis of the energy efficiency of the high-bandwidth memory (HBM) multi-channel dynamic random-access memory (MCDRAM) technology in Intel KNL architectures.
- The design and implementation of very efficient iterative refinement kernels for the NVIDIA V100 PCIe GPUs. Compared to the highly-optimized dgesv GPU kernel, our solution dhgesv-TC, exploiting the Tensor Cores, achieves the same accuracy of the solution up to 4× faster, and with up to 80% reduction in power consumption.

– Performance analysis of the NVIDIA V100 PCIe GPU, and insight into the possible energy efficiency opportunities.

The rest of the paper is organized as follows. We discuss related work in Sect. 3, and present the design and implementation details of our algorithm in Sect. 4. The experimental configurations and results are discussed in Sect. 5, followed by concluding remarks in Sect. 6

3 Related Works

Energy-Aware Algorithms for Scientific Computing: The first step toward the design of an energy-efficient system is an understanding of the power consumption of its components. The PowerPack project [7] serves this objective by providing detailed power-monitoring information on the disks, memories, NICs, processors, and even applications of HPC systems. Such power-monitoring details assist in identifying the most energy-consuming components, and working out energy reduction plans. For example, Global Extensible Open Power Manager (GEOPM) [5] provides a power management framework that enables an automatic online rebalancing of power among nodes. It also helps minimize the time-to-solution of applications while remaining within a target power budget. Another class of energy-efficient algorithms consists of designing energy-aware schedulers. The key idea is to divide an application into a set of tasks, and estimate the optimal power budget of each task. Then the energy-aware scheduler dynamically changes the frequency and voltage of CPU cores depending on the task to execute. This strategy is implemented by Adagio in a runtime system that makes dynamic voltage scaling (DVS) practical for complex scientific applications [17]. A variant has also been proposed by Kimura et al. [14], which uses DVFS to adapt the execution speed of each task to reduce the power consumption without increasing the overall execution time. Similar to DVFS, power-capping mechanisms, for example, to directly set power limits were introduced and accessed by tools like the Intel Running Average Power Limit (RAPL). Haidar et al. [8] studied these power-capping mechanisms and their effect in saving energy for various algorithms on Intel Xeon Phi architectures, specifically KNL.

Accurate power management for NVIDIA GPUs can be done using NVIDIA's Management Library (NVML) [16]. Work on validating it on dense linear algebra algorithms has shown that results are within 10% accurate [13]. Algorithmic work on making numerical libraries energy efficient for embedded systems with GPUs can be found in [9].

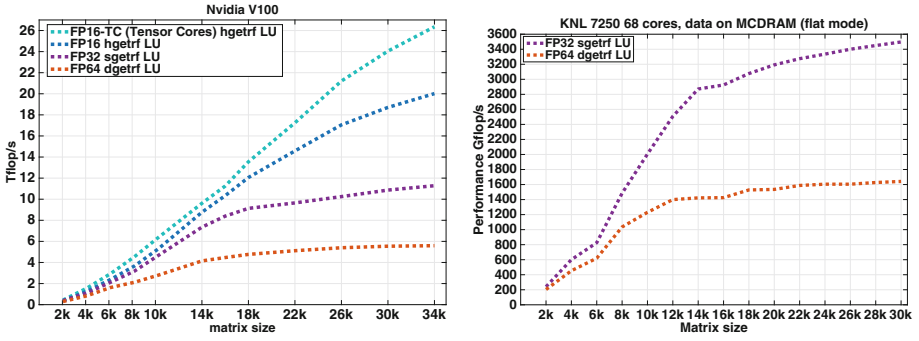
The History of Iterative Refinement: In the first version of iterative refinement introduced by Wilkinson, the factorization and the refinement process used the same precision [21]. The rounding errors analysis by Skeel [18] for LU solver, and extended by Higham [11] for a general solver, helped in gaining a deep understanding of iterative refinement. Other than the accuracy improvement, there has been a renewed interest in iterative refinement to improve the execution time of linear systems solvers in the 2000 s. With `FP32` twice as fast as `FP64`

on modern processors, Langou et al. [15], [1] proposed a mixed-precision iterative refinement where the matrix factorization step is in FP32 and everything else in FP64. More advanced versions of mixed-precision iterative refinement using FP16 have been recently studied by Carson and Higham [3,4] with the corresponding parallel implementations by Haidar et al. in [10].

4 Algorithmic Techniques Toward Energy Efficiency

4.1 Motivation

The main motivation for using lower-precision arithmetic is the speedup that can be achieved compared to the classical higher precision. We illustrate in Fig. 1 the performance that can be achieved by LU factorization using different precisions and on two different machines. In Fig. 1a, we show the obtained performance of the LU factorization (Xgetrf routine) on an NVIDIA V100 GPU for the four available precisions (FP64, FP32, FP16, and FP16-TC). We consider the FP16-TC as a precision since it consists of a mixed-precision Xgemm, where the multiplication is performed in FP16 while the accumulation is in FP32. Thus, FP16-TC is more accurate than the classical FP16 computation. We also note that, in addition to being more accurate, the FP16-TC is faster due to the use of Tensor Cores. As shown in Fig. 1a, the FP16-TC hgetrf-TC reaches about 4× speedup over its FP64 dgetrf counterpart. Furthermore, as expected, the FP16 hgetrf and the FP32 sgetrf are about 3× and 2× faster than the FP64 dgetrf. Similar behavior was observed on the Intel KNL 7250 system, reported in Fig. 1b. The LU factorization using FP32 achieves 2× speedup over the FP64 dgetrf. Such attractive performance results of the lower-precision LU guided our attention to the possibility of solving the linear system $Ax = b$ using a lower-precision LU factorization combined with an IR process to bring the solution to the FP64 arithmetic.



(a) On a Nvidia V100 GPU. (b) On an Intel KNL 7250 68 cores.

Fig. 1. Performance of the Xgetrf routine with different arithmetic precisions.

4.2 Iterative Refinement Techniques

IR is one of the most promising techniques used to obtain a high-precision solution to a linear equation using low-precision arithmetic for most of its computations. Specifically, we use `FP16` for the LU factorization, which consumes $2n^3$ FLOPs and `FP64` for everything else. The idea of (mixed precision) iterative refinement is to solve a linear system using low precision for its speed, and then refine the solution by solving the correction equation using high-precision arithmetic as shown in Algorithm 1. However, traditional convergence analysis of IR depends on the assumption that the matrix A is safely bounded away from singularity—meaning that its condition number ($\kappa(A)$) should be much less than u^{-1}, the inverse of the computing precision. Put differently, the condition $\kappa(A)u < 1$ should be satisfied. This condition seriously limits the applicability of `FP16` since the unit roundoff error is around $u \approx 5 \times 10^{-4}$, in which case the condition number of A should be much less than $u^{-1} \approx 2000$. Many well-conditioned matrices in `FP32` or `FP64` will become ill-conditioned in `FP16`.

Data: An $n \times n$ matrix A, and size n vector b
Result: A solution vector x_i approximating x in $Ax = b$, and a LU factorization of $A = LU$.
(`FP16`) Solve $Ax_0 = b$ using `FP16` LU factorization and triangular solve;
$i \leftarrow 0$;
repeat
  (`FP64`) Compute residual $r_i \leftarrow Ax_i - b$;
  (`FP64`) Solve $Ad_i = r_i$ using `IR`: triangular solve using the LU factors, or
  `IRGM`: GMRES preconditioned by $M = LU$;
  (`FP64`) Update $x_{i+1} = x_i + d_i$;
  $i \leftarrow i + 1$;
until $x^{(i)}$ *is accurate enough*;

Algorithm 1. `IR`: classic mixed-precision iterative refinement using triangular solve. `IRGM`: iterative refinement with GMRES to solve correction equation.

A recent study [4] relaxed this restrictive condition, and extended the application of IR for matrices where $\kappa(A) > u^{-1}$. They provided the following two new conditions to guarantee the convergence of IR:

- The correction equation ($Ad_i = r_i$) is solved relatively accurately: $\|d_i - \hat{d}_i\|_\infty / \|d_i\|_\infty = u\theta_i < 1$. Where θ_i is a constant depending on A, b, n, and u.
- The residual r_i contains a significant amount of components in every direction of the left singular vectors of A, such that we have $\mu_i \leq 1$ where μ_i defined as $\|r_i\| = \mu_i \|A\| \|x - \hat{x_i}\|$.

The first condition can be satisfied by replacing the typical LU-based solver for the correction equation with a variant of the generalized minimal residual method (GMRES), preconditioned by the low-precision LU factors. This is made

possible by two observations: (1) even for an ill-conditioned matrix, the partial pivoting LU still contains useful information. That is, using LU factors as preconditioner improves condition number: $\kappa(\hat{U}^{-1}\hat{L}^{-1}A) \approx 1 + \kappa(A)u$ even for $\kappa(A) \gg u^{-1}$; (2) GMRES is backward stable. The second condition is empirically observed in numerical experiments.

The convergence rate of `IRGM` depends on the convergence behavior of GMRES, which is complicated to predict. A preconditioner that is `FP16` accurate $M = LU \approx A$ further complicates the convergence rate picture. In general, for normal matrix A the GMRES converges faster as the condition number of A decreases, thus the low-precision LU would be of help because the preconditioning decreases condition number by a factor of $u = 5 \times 10^{-4}$; for a non-normal matrix the convergence rate cannot be entirely predicted by condition number. Thus, the convergence rate of `IRGM` depends on the matrix type, spectral properties, and matrix size. Therefore, we take a primarily empirical approach in the next sections.

We note that our iterative refinement process uses formula 1 as stopping criteria. The purpose of this paper is not the numerical study of the convergence of the IR methods, but rather to demonstrate how we can use techniques such as the IR methods to speed up the solution and obtain large energy gains. We note that, as described above, there is some limitation on where IR methods can work based on the matrices' condition number.

$$\frac{\| b - Ax \|_\infty}{\| x \|_\infty . \| A \|_\infty} \leq \epsilon\sqrt{n} \tag{1}$$

To make the paper self-contained and to highlight matrices' practical use of the IR methods, we show how the different IR methods discussed in this paper converge toward an `FP64` solution. For that we illustrate in Fig. 2 the convergence history of the three IR methods on an NVIDIA Volta GPU for a practical problem where the condition number of the matrix is about 10^4. The hardware detail is the same as the one described in the next section. This study aims to provide an analysis of each arithmetic as well as to provide insight into the expected performance from the iterative refinement methods.

We observe that the `FP32` technique requires 3 iterations, while the `FP16` slightly increases to about 7–8 iterations. Interestingly, the `FP16-TC` converges faster (4 iterations) than the `FP16` and slightly slower than the `FP32`. This is because the accumulation in the `FP16-TC` happens in `FP32` arithmetic and thus produces a better result than the `FP16`. We believe that the `FP32` routine will achieve a 2× speedup and that both of the `FP16` routines will achieve about 3×–4× speedup while delivering a solution at the `FP64` accuracy. More details about the performance are provided in the next section.

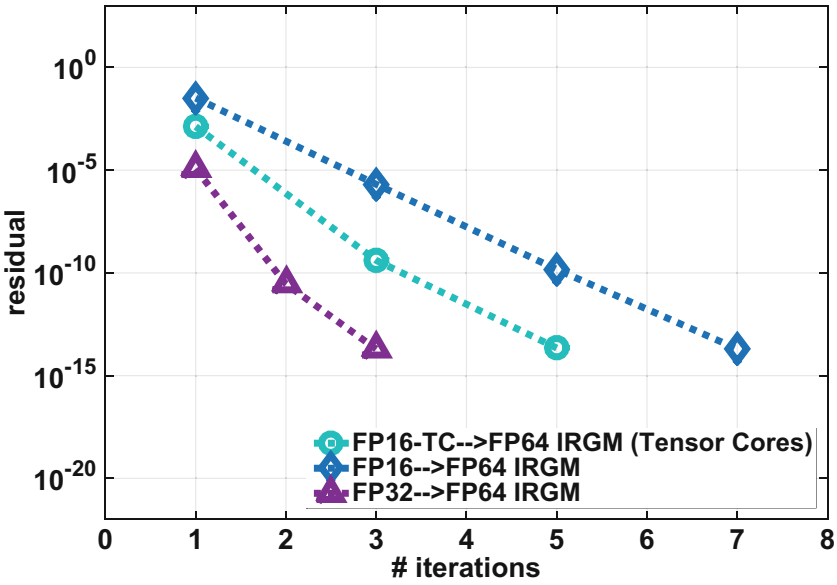


Fig. 2. Convergence history for the iterative refinement with GMRES using the three proposed low arithmetic for a matrix of size $n = 10000$, $\kappa_\infty(A) = 10^4$.

5 Experimental Results

This section presents the performance results and the power measurements of our iterative refinement methods—`dhgesv-TC`, `dhgesv`, and `dsgesv`—on either an NVIDIA V100 GPU or an Intel KNL 7250. The performances are computed by dividing the same FLOP count: $\frac{2}{3}n^3$ by the time to solution. As a result, a high performance reflects a fast time to solution. We used the KNL in self-hosted mode, i.e., without connection to CPU. This is not the case for the V100 GPU, which is used as an accelerator. We use LU factorization kernels from the Matrix Algebra on GPU and Multicore Architectures (MAGMA) library [19,20] in order to exploit both the CPU cores and the V100 GPU efficiently. Consequently, the V100 GPU performance results reported include both the CPU and GPU execution times. In the same way, the V100 energy efficiency results include both the power consumption on CPU and GPU. For the power measurement, we used the Performance Application Programming Interface (PAPI) [12], a performance-monitoring library recently updated for an efficient and accurate power measurement on both CPU and GPU.

5.1 Study of the Power Efficiency on KNL

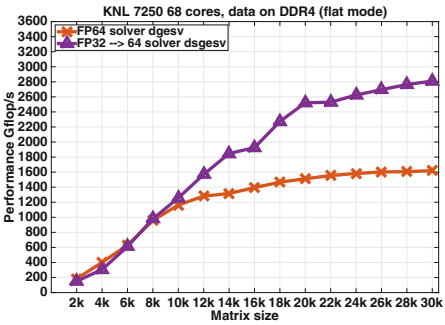
The Intel KNL 7250 has two types of memory. A large 96 GB DDR4 memory providing up to 90 GB/s of bandwidth (e.g., the conventional DRAM memory) and a 16 GB MCDRAM high-bandwidth memory that delivers up to 425 GB/s.

The MCDRAM can be configured into three modes: flat mode, cache mode and hybrid mode. In this experiment, the KNL has been configured in flat mode—that is, the entirety of the MCDRAM is used as an addressable memory. We mention that if the matrix size requires less than 16 GB, all these modes will behave the same.

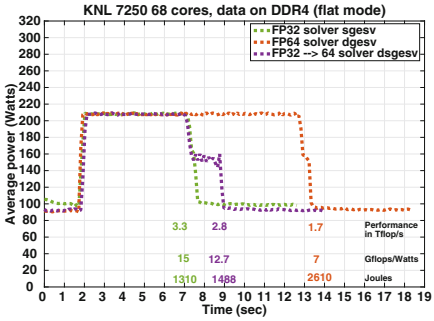
Figures 3a and 4a, show the performance obtained by our proposed `FP32` IR solver `dsgesv`, and the reference `FP64` `dgesv` solver for a matrices with $\kappa_\infty(A) \leq 10^4$. The number of iterations that the IR `dsgesv` required was not varying with the matrix size and took about 3 or 4 iterations to achieve the `FP64` solution. In the first experiments displayed in Fig. 3a, the data are allocated on the DDR4 memory. The direct solver `dgesv` reaches an asymptotic performance of 1600 gigaFLOP/s, while the IR solver `dsgesv` provides up to 2800 gigaFLOP/s; that represents 1.75× speedup over `dgesv`. That's the main motivation behind proposing IR methods to achieve higher performance and thus better energy efficiency.

The speedup of the IR method is directly translated into energy savings. The corresponding power consumption details are depicted in Fig. 3b. In total, `dgesv` (orange curve) consumed about 2610 Joules to compute the solution. The IR solver `dsgesv` (purple curve) helps, achieving 43% of energy reduction by using only 1488 Joules to deliver a similar accuracy solution. We have also displayed the gigaFLOP/s per Watt—the higher the better—which is the common energy efficiency metric used in the HPC community. The IR solver has an energy efficiency of 12.7 gigaFLOP/s per Watt, as opposed to 7 gigaFLOP/s per Watt for the standard solver `dgesv`; this demonstrates the energy efficiency of the IR solver. The power consumption of the `sgesv` function (green curve) is illustrated only for sake of completeness and to determine—when compared to the purple curve—the portion of the IR loop. In contrast to the compute intensive portion (e.g., the LU factorization), we can see that the power of the IR loop drops to about 160 W. This is normal because memory-bound routines do not drain high power since the CPU activity will be limited by the bandwidth and, thus, does not run at full speed in order to drain the maximal power.

We have repeated the same experiments, but this time, the data are allocated in the high-bandwidth memory MCDRAM. Since the MCDRAM has about 4× higher bandwidth, one can expect that memory-bound operations will be around 3–4 times faster. Note that, as described in Sect. 4.2, the IR method consists of the LU factorization and the iterative loop. The LU factorization is known to be a compute-intensive algorithm while the IR loop consists of a sequence of matrix-vector products (e.g., `dgemv`) and a linear solution (e.g., using `Xtrsv`), thus the memory-bound portion. This means that one can expect that the IR loop will be faster when the data are allocated in the MCDRAM rather than the DDR4, while the LU portion will achieve roughly same the performance wherever the data are allocated. One can expect the `dsgesv` routine to provide slightly higher performance than when the data are allocated on DDR4 because the IR iterations (usually 3 or 4 iterations) are faster. The performance and the energy efficiency results are displayed in Fig. 4a and b, respectively. As expected, one can observe that the MCDRAM provides no performance gain for the standard solver `dgesv`,



(a) Achieved Performance, $P = \frac{2n^3}{3}$ meaning higher is faster.



(b) Power and joules Consumption. Also shown is the Performance per Watt.

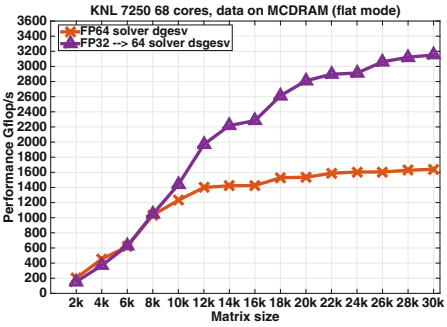
Fig. 3. Performance and power measurement of the linear solvers $Ax = b$ for the `IR` method compared with the `FP64` solver on KNL 7250 68 cores when data is on DDR4. (Color figure online)

this because `dgesv` is compute-bound and does not benefit from the high bandwidth. However, the IR solver `dsgesv` has shown a performance improvement of 14%, reaching 3200 gigaFLOP/s. As indicated above, this is due to the fact that the iterations of the IR consist of memory-bound kernels, which are sensitive to the bandwidth.

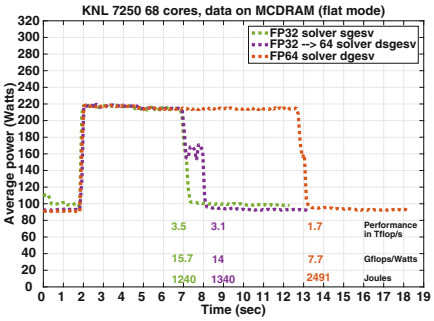
Regarding the energy efficiency, the IR technique revealed success. First, it brings an outstanding energy gain of 45% while providing a solution to the `FP64` accuracy. This is mainly due to the fact that (1) the LU factorization using the lower `FP32` precision is about 2× faster than its `FP64` counterparts, meaning it consumes about half the energy of the `FP64` and (2) the IR required less than 5 iterations. Further, we remark that both `sgesv` and `dgesv` consumed about 5% less energy. This energy reduction is due to the DDR4 being idle, which dropped its power consumption to 7 W, compared to 25 W in Fig. 3b where the DDR4 was used. `dsgesv` will also benefit from data on MCDRAM and will bring 5% energy economy compared to the one of Fig. 3b. In addition, since the MCDRAM provides higher bandwidth, the IR portion will be faster, as shown in Fig. 4b and thus will also offer further energy gains. Finally, the `dsgesv` showed an energy improvement of 10% thanks the MCDRAM.

5.2 Study of the Power Efficiency on GPU V100

NVIDIA's V100 PCIe GPU is the latest version of accelerator from NVIDIA with the Volta architecture. It has 5120 CUDA cores, along with the new 640 Tensor Cores. This new Tensor Core architecture is exclusively to accelerate GEMM-update operation in mixed precision. V100 has a peak performance of 7 teraFLOP/s in double precision, 14 teraFLOP/s in single precision, and 112 teraFLOP/s on Tensor Cores. It has 16 GB high-bandwidth memory, with a bandwidth of 900 GB/s. The interconnect bandwidth is 32 GB/s, and maximum energy consumption of the V100 is 250 W.



(a) Achieved performance, $P = \frac{2n^3}{3}$ meaning higher is faster.



(b) Power and joules consumption. Also shown is the performance per Watt.

Fig. 4. Performance and power measurement of the linear solvers $Ax = b$ for the `IR` method compared with the `FP64` solver on KNL 7250 68 cores when data is on MCDRAM.

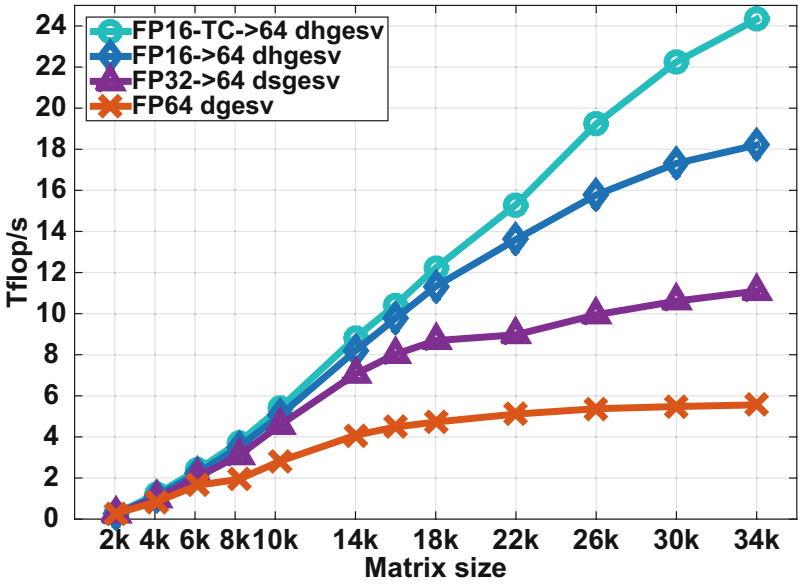


Fig. 5. Performance comparison of the linear solvers $Ax = b$ for the `IR` method using three different arithmetic and compared with the `FP64` solver on NVIDIA V100 GPU.

Figure 5 shows the performance obtained by the different IR solvers, as well as the reference `FP64 dgesv` solver for matrices with $\kappa_\infty(A) \leq 10^4$. All the IR variants' iterations ranged from 3 to 10 to converge for all matrix sizes. For example, the `FP32` algorithm converged with about 3 or 4 iterations while the `FP16` required between 7 and 10 iterations and the `FP16-TC` about 5 to 7 iterations. Thus, one can expect that the low-precision iterative refinement algorithms

will bring a large speedup compared to `dgesv`. Since the number of iterations is small, we envision that the speedup ratio will be similar to the one observed in Fig. 1a for the LU factorization. The `FP16-TC dhgesv-TC` solver is up to 4× faster than its `FP64 dgesv` counterpart. Similarly, the `FP16 dhgesv` and the `FP32 dsgesv` variants showed around 3× and 1.8× speedup over the `dgesv`, respectively. These observations endorse our findings that low-precision techniques can be used to speed up linear solvers by a large factor, and, as a consequence, one can expect similar improvements in terms of energy consumption.

The energy efficiency results are displayed in Fig. 6. We note that, here, since the GPU implementation is hybrid (meaning it uses the CPU and the GPU), we reported in Fig. 6 the sum of the CPU, DRAM, and GPU power measurement. The standard `dgesv` solver provides an energy efficiency of 14 gigaFLOP/s per Watt. Using the `FP32` IR `dsgesv` solver helps in doubling the energy efficiency, which increased up to 27 gigaFLOP/s per Watt. This follows our performance analysis described above, since the `dsgesv` is about twice as fast and thus we can observe twice the energy efficiency using the `dsgesv` routine. The results become more impressive with the `FP16 dhgesv`, which showed more than 3× the energy efficiency of `dgesv`. Finally, the most pronounced result is shown by the `FP16-TC dhgesv-TC` solver. It achieved an unprecedented energy efficiency of 74 gigaFLOP/s per Watt—that is a more than 5× improvement over the standard `dgesv` solver. These results demonstrate that the IR methods and half-precision arithmetic will be decisive in helping mitigate the power constraints in large-scale

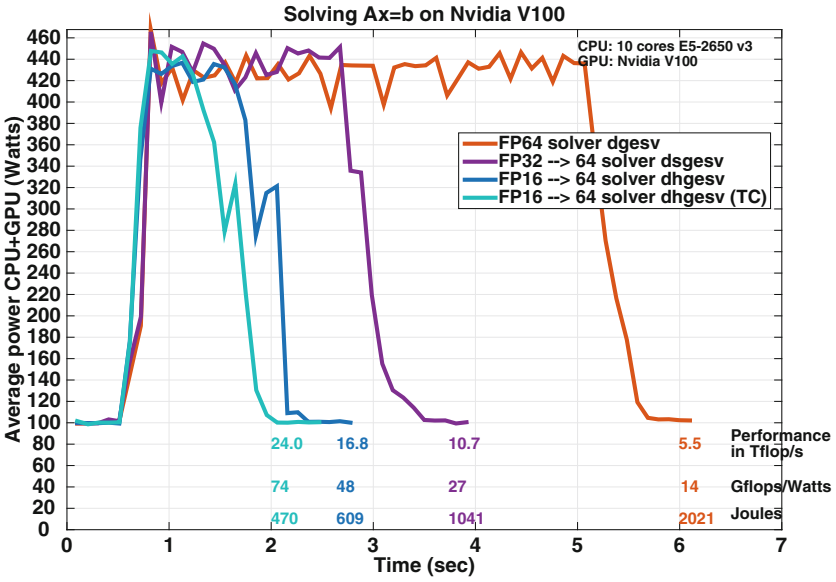


Fig. 6. Power consumption of the linear solvers $Ax = b$ for the `IR` method using three different arithmetic and compared with the `FP64` solver on NVIDIA V100 GPU. Also shown is the performance per Watt.

HPC systems. To make this description self-contained, we would also mention that similarly to the KNL observation, we can easily determine the portion of the IR loop in these graphs. It is the portion with the lower power consumption (e.g., the portion draining 300 W). We can also see that the IR portion for `dsgesv` is short compared to the one for either the `dhgesv` and the `dhgesv-TC`. This is normal since, as mentioned above, the `dsgesv` required about 3 or 4 iterations while both `dhgesv` and `dhgesv-TC` required 7–10 and 5–7 iterations respectively.

6 Conclusion

This work is a direct response to increasing concerns about power efficiency in the HPC community. Existing works focus on dynamically tuning hardware voltage and frequency to save energy at the cost of performance. In this work, we propose a new approach to power efficiency and demonstrate that it is possible to increase both performance and power efficiency by leveraging the knowledge of applications. For the solution to linear systems of equations, a novel algorithm is designed and implemented. The initial approximation of the solution is computed using power efficient and fast reduced-precision arithmetic. This is followed by accuracy iterations to improve the accuracy in a higher precision. We have shown that, by combining `FP32` and `FP64`, we can accelerate the execution time on Intel KNL architectures up to 2×—and reduce their power consumption by up to half. The results on the new NVIDIA V100 PCIe GPUs are even more promising. We have achieved 4× speedup, and more than 80% reduction in power consumption, by exploiting the `FP16` features of the V100 GPU Tensor Cores.

In the 2000s, the potential of mixed-precision iterative refinement has been investigated for performance reasons. To the best of our knowledge, this work is the first study that demonstrates the immense potential of mixed-precision iterative refinement for large-scale computation. In future work, we aim to extend this work to ARM and IBM POWER architectures, and build a framework that will automatically identify the operations to be executed in reduced precision in applications without compromising the final accuracy.

Acknowledgments. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. The work was also partially supported by NVIDIA and NSF grant No. OAC-1740250.

References

1. Baboulin, M., Buttari, A., Dongarra, J., Kurzak, J., Langou, J., Langou, J., Luszczek, P., Tomov, S.: Accelerating scientific computations with mixed precision algorithms. Comput. Phys. Commun. **180**(12), 2526–2533 (2009)
2. Betkaoui, B., Thomas, D.B., Luk, W.: Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing. In: 2010 International Conference on Field-Programmable Technology, pp. 94–101, December 2010

3. Carson, E., Higham, N.J.: Accelerating the solution of linear systems by iterative refinement in three precisions. MIMS EPrint 2017.24, University of Manchester (2017)
4. Carson, E., Higham, N.J.: A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. SIAM J. Sci. Comput. **39**(6), A2834–A2856 (2017). https://doi.org/10.1137/17M1122918
5. Eastep, J., Sylvester, S., Cantalupo, C., Geltz, B., Ardanaz, F., Al-Rawi, A., Livingston, K., Keceli, F., Maiterth, M., Jana, S.: Global extensible open power manager: a vehicle for HPC community collaboration on co-designed energy management solutions. In: Kunkel, J.M., Yokota, R., Balaji, P., Keyes, D. (eds.) ISC 2017. LNCS, vol. 10266, pp. 394–412. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58667-0_21
6. Etinski, M., Corbalán, J., Labarta, J., Valero, M.: Understanding the future of energy-performance trade-off via DVFS in HPC environments. J. Parallel Distrib. Comput. **72**(4), 579–590 (2012)
7. Ge, R., Feng, X., Song, S., Chang, H.C., Li, D., Cameron, K.W.: Powerpack: energy profiling and analysis of high-performance systems and applications. IEEE Trans. Parallel Distrib. Syst. **21**(5), 658–671 (2010)
8. Haidar, A., Jagode, H., YarKhan, A., Vaccaro, P., Tomov, S., Dongarra, J.: Power-aware computing: Measurement, control, and performance analysis for Intel Xeon Phi. In: 2017 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–7, September 2017
9. Haidar, A., Tomov, S., Luszczek, P., Dongarra, J.: Magma embedded: towards a dense linear algebra library for energy efficient extreme computing. In: 2015 IEEE High Performance Extreme Computing Conference (HPEC 2015), (Best Paper Award). IEEE, Waltham, September 2015
10. Haidar, A., Wu, P., Tomov, S., Dongarra, J.: Investigating half precision arithmetic to accelerate dense linear system solvers. In: SC16 Scal A17: 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems. ACM, Denver, November 2017
11. Higham, N.J.: Iterative refinement enhances the stability of QR factorization methods for solving linear equations. BIT Numer. Math. **31**(3), 447–468 (1991). https://doi.org/10.1007/BF01933262
12. Jagode, H., YarKhan, A., Danalis, A., Dongarra, J.: Power management and event verification in PAPI. In: Knüpfer, A., Hilbrich, T., Niethammer, C., Gracia, J., Nagel, W.E., Resch, M.M. (eds.) Tools for High Performance Computing 2015, pp. 41–51. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39589-0_4
13. Kasichayanula, K., Terpstra, D., Luszczek, P., Tomov, S., Moore, S., Peterson, G.: Power aware computing on GPUs. In: SAAHPC 2012 (Best Paper Award), Argonne, IL, July 2012
14. Kimura, H., Sato, M., Hotta, Y., Boku, T., Takahashi, D.: Empirical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster. In: 2006 IEEE International Conference on Cluster Computing, pp. 1–10, September 2006
15. Langou, J., Luszczek, P., Kurzak, J., Buttari, A., Dongarra, J.: Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). In: SC 2006 Conference, Proceedings of the ACM/IEEE, p. 50, November 2006
16. NVIDIA Management Library (NVML), NVIDIA (2018). https://developer.nvidia.com/nvidia-management-library-nvml

17. Rountree, B., Lownenthal, D.K., de Supinski, B.R., Schulz, M., Freeh, V.W., Bletsch, T.: Adagio: making DVS practical for complex HPC applications. In: Proceedings of the 23rd International Conference on Supercomputing, ICS 2009, pp. 460–469. ACM, New York (2009). https://doi.org/10.1145/1542275.1542340
18. Skeel, R.D.: Iterative refinement implies numerical stability for Gaussian elimination. Math. Comput. **35**(151), 817–832 (1980)
19. Tomov, S., Dongarra, J., Baboulin, M.: Towards dense linear algebra for hybrid GPU accelerated manycore systems. Parallel Comput. Syst. Appl. **36**(5–6), 232–240 (2010). https://doi.org/10.1016/j.parco.2009.12.005
20. Tomov, S., Nath, R., Ltaief, H., Dongarra, J.: Dense linear algebra solvers for multicore with GPU accelerators. In: Proceedings of the IEEE IPDPS 2010, Atlanta, GA, pp. 1–8, 19–23 April 2010
21. Wilkinson, J.H.: Rounding Errors in Algebraic Processes. Prentice-Hall, Upper Saddle River (1963)