MANCHESTER 1824
The University of Manchester

The University of Manchester Research

# The Design of Fast and Energy-Efficient Linear Solvers: On The potential Of Half Precision Arithmetic And Iterative Refinement Techniques

**DOI:**
10.1007/978-3-319-93698-7_45

**Document Version**
Accepted author manuscript

Link to publication record in Manchester Research Explorer

**Citation for published version (APA):**
Haidar, A., Abdelfattah, A., Zounon, M., Wu, P., Pranesh, S., Tomov, S., & Dongarra, J. (2018). The Design of Fast and Energy-Efficient Linear Solvers: On The potential Of Half Precision Arithmetic And Iterative Refinement Techniques. In *Computational Science – ICCS 2018* (pp. 586-600) https://doi.org/10.1007/978-3-319-93698-7_45

**Published in:**
Computational Science – ICCS 2018

**Citing this paper**
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

**General rights**
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

**Takedown policy**
If you believe that this document breaches copyright please refer to the University of Manchester's Takedown Procedures [http://man.ac.uk/04Y6Bo] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.

OPEN ACCESS

Download date:27. Feb. 2020

# 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
[2] University of Houston, TX, USA
[3] Oak Ridge National Laboratory, Oak Ridge, USA
[4] University of Manchester, Manchester, U.K.
{haidar,ahmad,pwu11,tomov,dongarra}@icl.utk.edu
{mawussi.zounon,srikara.pranesh}@manchester.ac.uk

**Keywords:** FP16, Tensor Cores, Mixed-precision, HPC, Solvers

**Abstract.** As parallel computers approach the 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 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 system 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 the modern accelerators and co-processors. 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 optimised linear system solvers, our kernels are up to $2\times$ faster to deliver the same accuracy solution, and reduce the energy consumption up to half on Intel KNL architectures. By using efficiently the tensor cores available in the NVIDIA V100 PCIe GPUs, the speedups can be up to $4\times$ with more than 80% reduction on the energy consumption.

## 1 Introduction

As parallel computers approach the 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 due to the increasing performance of HPC systems. However it comes with a cost: electrical power consumption. This leads to the following two main concerns, increase of the power bills beyond affordable budgets, and increasing impact on the environment.

To help meet 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 Array (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 strategies with low performance overhead. In this work we propose a new approach to energy efficiency, which in addition to significantly decreasing the power consumption, also 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. Solution of 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. We exploit both hardware solutions such as energy efficient NVIDIA GPUs and Intel Xeon Phis, and algorithmic techniques such as iterative refinement 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 of the 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 the Gaussian elimination. However, the accuracy of the obtained solution is often unsatisfactory because of the round-off errors generated during the Gaussian elimination. The iterative refinement technique, first introduced by Wilkinson [18] 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 $r = b - Ax$. 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 $\bar{x} \leftarrow \bar{x} + d$, that 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 KNL can deliver 3 teraflops per second (Tflop/s) of `FP64` performance, but in `FP32`, it can achieve more than 6 Tflop/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 PICe GPU has a peak performance of 7 TFlop/s in `FP64`, 14 TFlop/s in `FP32`, and 112 TFlop/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 summarised in Table 1.

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

Table 1: Variants of iterative refinement implemented in this work. From left to right, the first column list 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".

## 2 Contributions

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

- The design and implementation of a highly optimised iterative refinement kernel for Intel KNL architectures. Compared to the standard algorithm (dgesv), our kernel (dsgesv) is up to 2× faster to deliver the same accuracy solution, and reduces the power consumption by half.
- Analysis of the energy efficiency of the high bandwidth 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 optimised dgesv GPU kernel, our solution dhgesv-TC, exploiting the tensor cores, is up to 4× faster to achieve the same accuracy of the solution, and with up to 80% reduction on the power consumption.
- Performance analysis of the NVIDIA V100 PCIe GPU, and insight on the possible energy efficiency opportunities.

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

monitoring information of 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 minimising 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 in 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 [14]. A variant has also been proposed by Kimura et al. [12] which uses dynamic voltage and frequency scaling (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, e.g., 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.

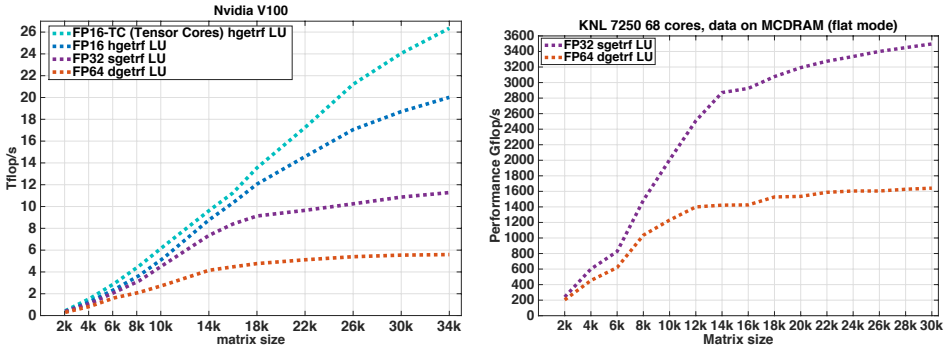
***The History of Iterative Refinement:*** In the first version of iterative refinement introduced by Wilkinson, the factorization and the refinement process used the same precision [18]. The rounding errors analysis by Skeel [15] for LU solver, and extended by Higham [10] for a general solver, helped in gaining a deep understanding of iterative refinement. Other than the accuracy improvement, there has been a renewed interest in iterative refinement since 2000s, to improve the execution time of linear systems solvers. With `FP32` twice as fast as `FP64` on modern processors, Langou et al. [13], [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 [9].

## 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 Figure 1 the performance that can be achieved by the LU factorisation using different precisions and on two different machines. In Figure 1a, we show the obtained performance of the LU factorisation (`Xgetrf` routine) on a 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 also faster due to the use of Tensor Cores. As shown in Figure 1a, the `FP16-TC`

hgetrf-TC reaches about 4X speedup over its FP64 dgetrf counterpart. Furthermore, as expected, the FP16 hgetrf, and the FP32 sgetrf are about 3X and 2X faster than the FP64 dgetrf. Similar behavior was observed on the Intel KNL 7250 system, reported in Figure 1b. The LU factorisation using FP32 achieve 2X 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 factorisation combined with Iterative Refinement 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

Iterative refinement (IR) is one of the most promising techniques to obtain 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.

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:

**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.

- 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 $\theta_i$ is a constant depending on $A$, $b$, $n$, and $u$.
- The residual $r_i$ contains significant amount of components in every direction of the left singular vectors of $A$, such that we have $\mu_i \leq 1$ where $\mu_i$ 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 GMRES, preconditioned by the low precision LU factors. This is made possible by two observations: 1) even for a 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 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.

## 5 Experimental Results

This section presents the performance results and the power measurement of our iterative refinement methods – `dhgesv-TC`, `dhgesv`, and `dsgesv`– on either a 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 that is used as an accelerator. We use LU factorization kernels from the MAGMA library [16,17] 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 PAPI [11], 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 require less then 16 GB, all these modes will behave the same.

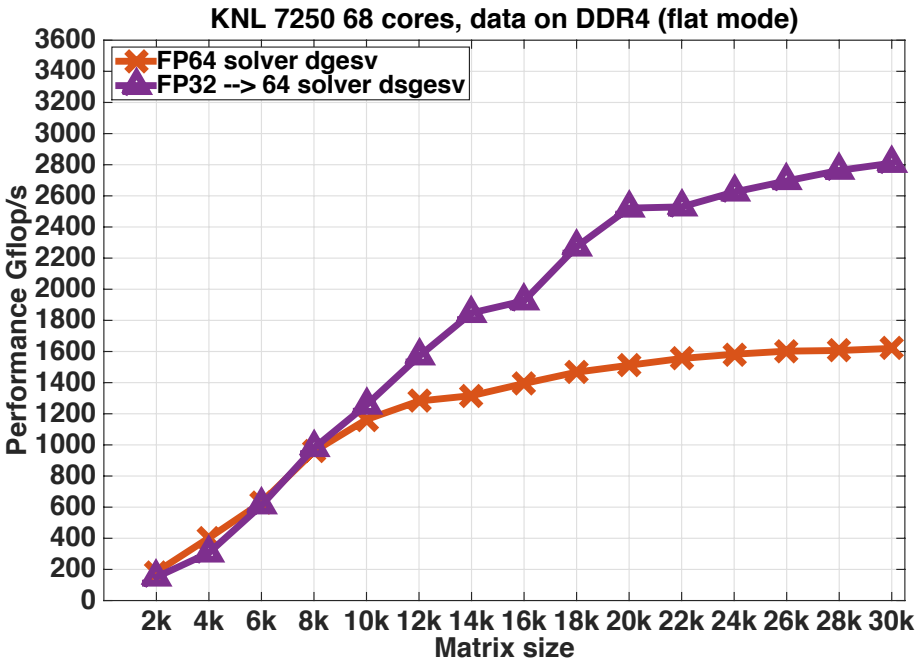



Fig. 2: Performance of three precision linear solvers $Ax = b$ on KNL 7250 68 cores when data is on DDR4.

Figure 2 and Figure 4, shows 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 was about 3 or 4 iterations to achieve the FP64 solution. In the first experiments displayed in Figure 2, the data are allocated on the DDR4 memory. The direct solver `dgesv` reaches an asymptotic performance of 1600 Gflop/s, while the IR solver `dsgesv` provides up to 2800 Gflop/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 Figure 3. 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 Gflops/Watts, the higher the better, that is the common energy efficiency metric used in the HPC community. IR solver has an energy efficiency of 12.7 Gflops/Watts, as opposed to 7 Gflops/Watts 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 drop 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.

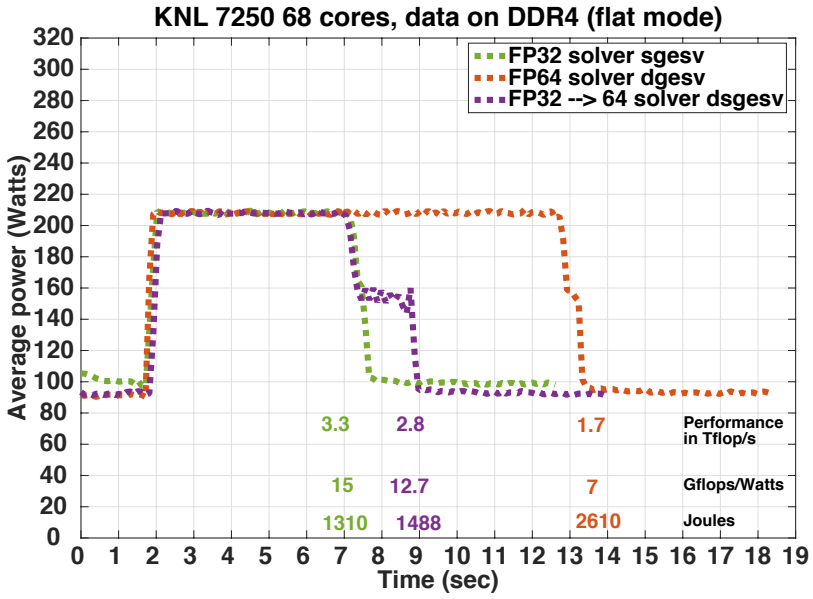



Fig. 3: Power Consumption of three precision linear solvers $Ax = b$ on KNL 7250 68 cores when data is on DDR4.

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 section 4.2, the IR method consists of the LU factorization and the iterative loop. In this regard, the following two points require special attention. One, the LU factorisation is known to be compute-intensive algorithm. Second, the IR loop consists of a sequence of matrix-vector product (e.g., `dgemv`), and a solution of linear system of equation (e.g.,using `Xtrsv`), thus this is a memory bound portion. Therefore one can expect that the IR loop will be faster when data is on DDR4, while the LU portion will achieve roughly same the performance. One can expect the `dsgesv` routine to provide slightly higher performance than the one when data is allocated on DDR4 because the IR iterations (usually 3 or 4 iterations) are faster. The performance and the energy efficiency results are displayed in Figure 4 and Figure 5, respectively. As expected, one can observe that the MCDRAM provides no performance gain for the standard solver `dgesv`, this because `dgesv` is compute-bound and does not benefit from the high bandwidth. However, the IR solver `dsgesv` has showed a performance improvement of 14%, reaching 3200 Gflop/s. As indicated above, this is due to the fact that the iterations of the iterative refinement consists of memory-bound kernels which are sensitive to the bandwidth.

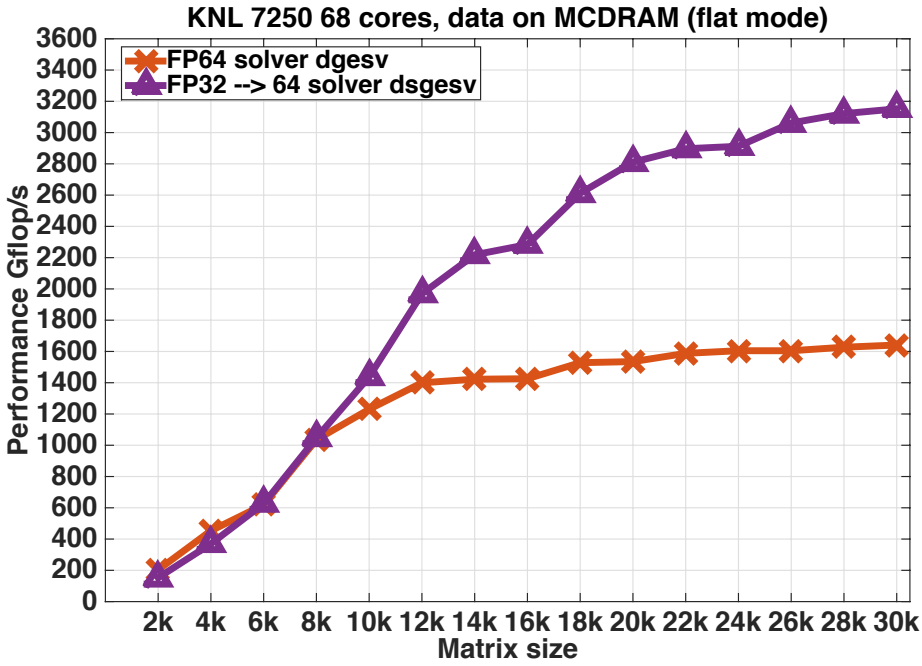



Fig. 4: Performance of three precision linear solvers $Ax = b$ on KNL 7250 68 cores when data is on MCDRAM.

Regarding the energy efficiency, the IR technique revealed successful. 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 factorisation using the lower `FP32` precision is about twice faster than its `FP64` counterparts meaning it consumes about half the energy of the `FP64` and 2) the IR required less than 5 iterations. Second, we remark that both `sgesv` and `dgesv` consumed about 5% less energy. This energy reduction is due to the DDR4 being idle, dropped its power consumption to 7 W, compared to 25 W in Figure 3 where the DDR4 have been used. `dsgesv` will also benefit from data on MCDRAM and will bring 5% energy economy as well compared to the one of Figure 3. In addition to that, since the MCDRAM provides higher bandwidth, the IR portion will be faster as shown in Figure 5 and thus will also offer further energy gain. Finally the `dsgesv` showed an energy improvement of 10% thanks the MCDRAM.

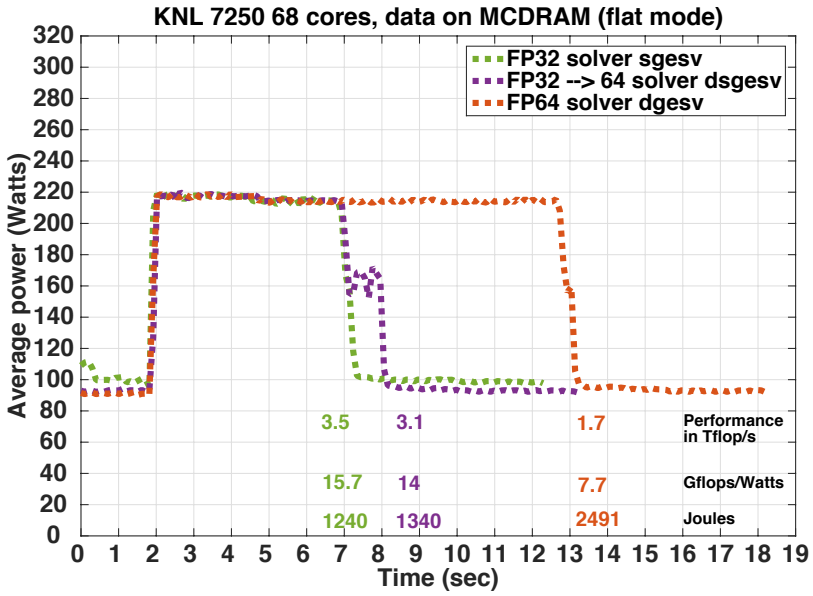



Fig. 5: Power Consumption of three precision linear solvers $Ax = b$ on KNL 7250 68 cores when data is on MCDRAM.

### 5.2 Study of the power efficiency on GPU V100

NVIDIA 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 TFlop/s in double precision, 14 TFlop/s in single precision, and 112 TFlop/s on tensor cores. It has 16 GB HB memory,

with a bandwidth of 900 GB/s. The interconnect bandwidth is 32 GB/s, and maximum energy consumption of the V100 is 250W.

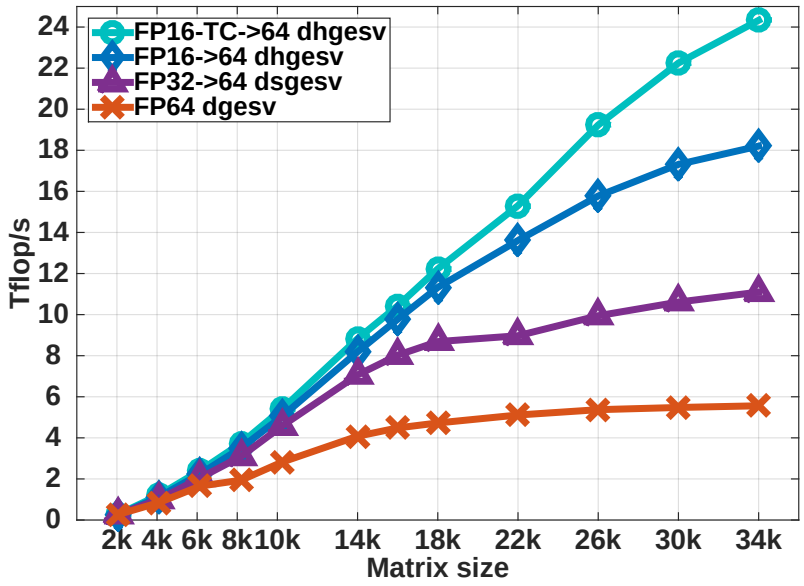



Fig. 6: Performance of four precision linear solver $Ax = b$ on Nvidia V100 GPU.

Figure 6, shows the performance obtained by the different IR solvers, as well as the reference `FP64 dgesv` solver for a 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 Figure 1a for the LU factorization. The `FP16-TC dhgesv-TC` solver is up to $4\times$ times faster than its `FP64 dgesv` counterpart. Similarly, the `FP16 dhgesv` and the `FP32 dsgesv` variants showed around $3\times$ and $1.8\times$ speedup over the `dgesv`, respectively. These observations endorse our findings that low precision techniques can be used to speedup linear solvers by a large factor and as a consequence one can expect similar gain in terms of energy consumption.

The energy efficiency results are displayed in Figure 7. We note that, here, since the GPU implementation is hybrid (meaning it uses the CPU and the GPU), we reported in Figure 7 the sum of the CPU, DRAM, and GPU power measurement. The standard `dgesv` solver provides an energy efficiency of 14 Gflops/Watts. Using the `FP32` IR `dsgesv` solver, helps in doubling the energy efficiency, that increased up to 27

Gflops/Watts. This follows our performance analysis described above, since the `dsgesv` is about twice faster and thus we can observe twice energy efficiency using the `dsgesv` routine. The results become more impressive with the `FP16` `dhgesv` that 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 Gflops/Watts, that is more than 5× improvement over the standard `dgesv` solver. These results demonstrate that the IR methods and half precision arithmetic will be decisive to help meeting the power constraints in large scale HPC systems. To make this description self content, we would also mention that similarly to the KNL observation, we can easily determine the portion of the IR loop in these graph. It is the portion with the lower power consumption (e.g., the portion draining 300W). 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.

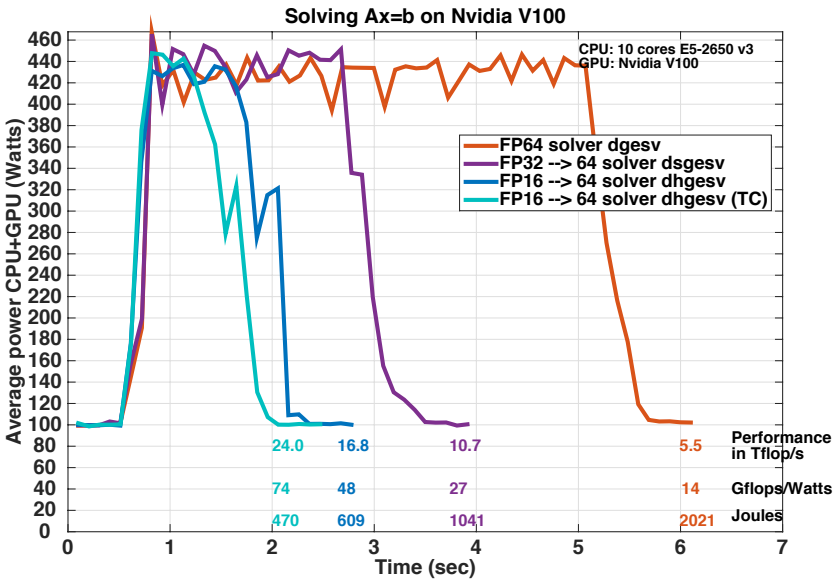



Fig. 7: Power Consumption of four precision linear solver $Ax = b$ on Nvidia V100 GPU.

## 6 Conclusion

This work is a direct response to the increase power efficiency concerns 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 the performance and the power efficiency by leveraging the knowledge of applications. For the solution of linear system 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 up to $2\times$ the execution time on Intel KNL architectures, and reducing their power consumption by upto half. The results on the new NVIDIA V100 PCIe GPUs are even more promising. We have achieved $4\times$ speedup, and more than 80% reduction on the power consumption, by exploiting the `FP16` features of the V100 GPU tensor cores.

In 2000's 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 in applications, the operations to be executed in reduced precision without compromising on the final accuracy.

## Acknowledgments

This research was supported by the Exascale Computing Project, 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. Computer Physics Communications 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 (Dec 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 Journal on Scientific Computing 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, pp. 394–412. Springer International Publishing, 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. Journal of Parallel and Distributed Computing 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 Transactions on Parallel and Distributed Systems 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 (Sept 2017)
9. Haidar, A., Wu, P., Tomov, S., Dongarra, J.: Investigating Half Precision Arithmetic to Accelerate Dense Linear System Solvers. In: SC16 ScalA17: 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems. ACM, ACM, Denver, CO (11/2017 2017)
10. Higham, N.J.: Iterative refinement enhances the stability of QR factorization methods for solving linear equations. BIT Numerical Mathematics 31(3), 447–468 (Sep 1991), https://doi.org/10.1007/BF01933262
11. 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 International Publishing, Cham (2016)
12. 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 (Sept 2006)
13. Langou, J., Langou, J., Luszczek, P., Kurzak, J., Buttari, A., Dongarraa, J.J.: Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (2006)
14. 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. pp. 460–469. ICS '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1542275.1542340
15. Skeel, R.D.: Iterative Refinement Implies Numerical Stability for Gaussian Elimination. Mathematics of Computation 35(151), 817–832 (1980)
16. Tomov, S., Dongarra, J., Baboulin, M.: Towards dense linear algebra for hybrid GPU accelerated manycore systems. Parellel Comput. Syst. Appl. 36(5-6), 232–240 (2010), DOI: 10.1016/j.parco.2009.12.005
17. Tomov, S., Nath, R., Ltaief, H., Dongarra, J.: Dense linear algebra solvers for multicore with GPU accelerators. In: Proc. of the IEEE IPDPS'10. pp. 1–8. Atlanta, GA (April 19-23 2010)
18. Wilkinson, J.H.: Rounding Errors in Algebraic Processes. Prentice-Hall (1963)