A priori power estimation of linear solvers on multi-core processors

Dimitar Lukarski[1], Tobias Skoglund[2]
Uppsala University
Department of Information Technology
Division of Scientific Computing[1]
Division of Computer Systems[2]
{dimitar.lukarski;tobias.skoglund}@it.uu.se

May, 2013

ABSTRACT

High-performance computing (HPC) centres simulate complex scientific models which provide vital understanding of our world. In the recent years, power efficiency has become a critical aspect in the new HPC facilities because of high energy consumption costs. In this work, we present our study on power consumption of linear solvers on modern multi-core CPUs which are widely used in many scientific applications. We focus on both dense and sparse linear solvers – parallel direct solvers for the dense, and parallel iterative methods for the sparse problems.

We provide answers to the questions – what is the energy efficiency using multi-core parallel algorithms for linear systems and do we save energy using more cores? Furthermore, we propose a methodology for estimating total power consumption. Based on benchmarks which achieve high accuracy we estimate total power usage of the whole solution phase on multi-core CPUs.

1 INTRODUCTION

A recent study has shown that processors are accounted for one-third of the power budget in cluster computers [1]. A key contributing factor of total consumption is idle time which translates to static power leakage in the chip. However, an idle processor also wastes time which translates into performance. As such, the link between performance of the processor and power consumption is strong and though many tools for evaluating performance exist, acquiring power consumption numbers is not as easy.

Among many supercomputer scientific applications, linear solvers are one of the most widely used algorithms [2, 3]. In this work, we want to provide information about power consumption and efficiency in multi-core processors using parallel algorithms for solving linear systems. Here, we study two types of problems - dense and sparse. For the dense problem, we consider parallel direct solvers (based on PLASMA software, [4]) and for the sparse problem we use iterative methods (based on PARALUTION software, [5]). Our goal is to devise a scheme for predicting the power consumption of these methods. We use benchmarks find the power dynamics of matrix and vector operations found in typical parallel linear solver algorithms.

In this work, we have developed a methodology for estimating power consumption based only on the distinction between compute bounded and bandwidth bounded matrix and vector algorithms. Iterative solvers are bounded by bandwidth since they perform relatively few floating-point operations (FLOPs) per element. In contrast, dense direct solvers have relatively high number of FLOPs per element which deme them compute bounded.

In this paper, we present our method of deriving power predictions for linear solvers. This method was

developed for linear solvers, however it can be generalized for any type of matrix or vector algorithm and is not limited to the applications considered in this work. Practically, our method has the following properties: (i) independence of input, (ii) independence of architecture and (iii) simplicity.

Our results show that the method is $86 - 95\%$ accurate at estimating our benchmarks. Based on these results we compute the speed-up (performance) and green-up (energy efficiency) of dense and sparse linear solvers using a multi-core processor.

2 RELATED WORK

In this work we have focused on capturing the power characteristics of one particular processor model. Others have also reported power consumption in various linear algebra operations [6], there using the PLASMA library. The authors also observe similar characteristics of power over time, but their goal in that work was different. The observation in their work is that long-running programs, such as linear algebra operations, consume power steadily.

In modelling, Choi and Vuduc [7] recently developed architecture independent methods for analysing performance and energy (power). Their approach is to describe performance and energy arising from floating-point operations and memory operations. The architecture that they use was based on heuristics to abstract processors and different levels of memory. Their goal was to predict limitations of future HPC- centres based on algorithmic design. Their work is an important first step in modelling energy.

Another approach is to use the fundamental energy equations of switching circuits [8, 9] to achieve higher accuracy. In this work, the authors acknowledge power as a combination of dynamic, static and leakage power. Under this assumption they measure power in different parts of the processor and train a model to achieve very accurate power estimations. The authors goal is to model instant power consumption accurately and they rely on the existence of performance counters in the processor to report execution statistics.

Software methods for estimating power [10, 11, 8] are based on the assumption that hardware performance counters are correlated with power. These techniques focuses primarily on accurate instant estimates and the information is used to create power efficient policies that can be applied in OS kernels or at hardware-level.

Our work focuses only on the performance and power efficiency (green-up) for linear solvers. The direction that we take is to base our predictions on observations of power consumption in linear algebra operations.

This approach is more native to large and complex HPC-applications which includes linear solvers, and using our methodology we achieve close estimates at no overhead cost.

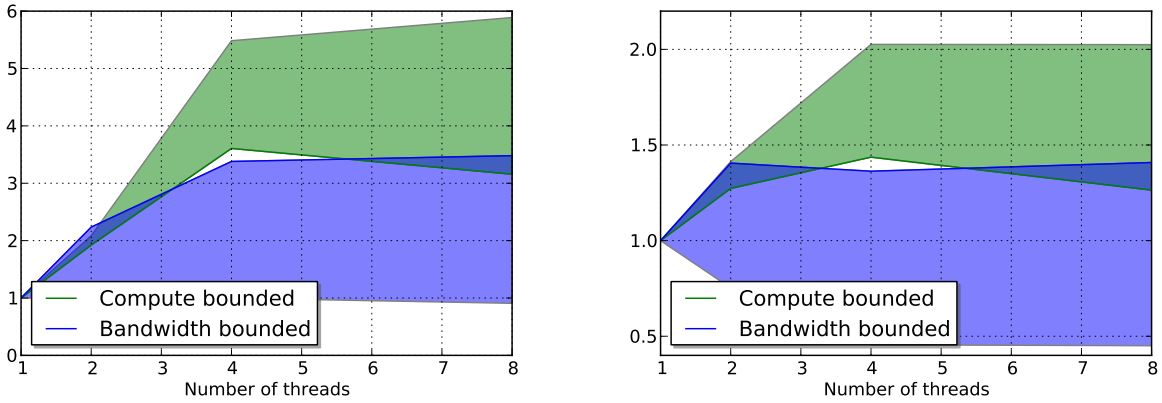
3 BENCHMARKS AND SOLVERS

We classify each routine or algorithm as either being asymptotically bounded by computations or bounded by bandwidth. The result of being either compute- or bandwidth bounded is reflected in the runtime, as the number of threads increase to the number of physical cores in the processor. We compute the classification taking the ratio between the number of operands and operations. An operation having relatively less operations per operand is classified as bandwidth bounded and vice versa for the compute bounded classification. For modern CPUs, all sparse operations are bandwidth bounded and all dense matrix operations are compute bounded.

For the dense operations and solvers we use PLASMA 2.4.6 [4] which is a multi-threaded library that used a high-performance BLAS-backend. The matrix multiplication and factorization of PLASMA are tile-based which gives a high level of task-parallelism. And for the sparse ones we use PARALUTION (version 0.0.1b) [5] which provides several backends and can express fine-grained level of parallelism. In this experiment we use the OpenMP backend of the library.

3.1 Sparse Benchmarks

The palette for sparse operations which we use in this work includes: vector updates of type $x = x + \alpha y$ and



(a) Speed-up (b) Green-up

Figure 1: Envelopes show the minimum and maximum speed-up (left) and green-up (right) of compute bounded (green) and bandwidth bounded (blue) algorithms.

$x = \alpha x + y$, where x and y are real vectors and α is a scalar parameter; dot product; and sparse matrix-vector multiplications. The performance of the sparse matrix-vector multiplication depends heavily on the sparsity pattern. Because of that, we test three different matrices which test structured and non-structured patterns – `FEM_3D_thermal2`, `apache2` and `ecology2`, available from [12]. For sparse matrix inputs we use four common storage formats (CSR, HYB, ELL, MCSR).

3.2 Sparse Solvers

Almost all of the iterative methods for solving sparse linear systems can be seen as combinations of various vector operations and matrix-vector multiplication. Thus, without lose of generality, we test Conjugate Gradient method (CG), which one of the most famous algorithm for solving symmetric and positive definite matrices. The convergence properties of the solvers depends on the spectrum of the matrix. In all tests, we user relative tolerance of 10^{-6} and zero initial values.

3.3 Dense Benchmarks

For the dense benchmarks, we use only square dense matrix multiplication. We choose sizes which fit in and exceed the cache size of our system. The sizes are 6000, 7000, 8000 and 9000 and the elements are generated as random numbers.

3.4 Dense Solvers

The dense solvers are based on LQ, QR and Cholesky decompositions. These algorithms have the same complexity. Therefore, we present our analysis only for the performance of LQ factorization.

4 METHODOLOGY

In this section we present our methodology for estimating the total power consumption of scientific workloads based on mean power. We will use standard dense and sparse matrix and vector operations to derive these estimations.

The steps to derive the estimations are: first, to measure the power dynamics of the processor using a number of threads and a set of compute and bandwidth bounded benchmarks. Second, to calculate the

mean of each such thread and benchmark combination, and finally to combine means of all compute and bandwidth bounded measurements respectively into one estimator ω. The final estimator can be computed using any optimal condition. Throughout this work we have used the least-mean-square error to compute the final estimator for 1, 2, 4 and 8 threads running dense and sparse workloads that can be characterized as either compute or bandwidth bounded.

4.1 Power measurement setup

We use an instrument setup previously presented in [8] to accurately measure the instant power consumption in the processor. This is done by measuring voltage in the off-chip voltage regulator residing on the motherboard at millisecond resolution. The resulting data is the instant energy consumption in Watt (W) measured every millisecond. The total energy consumption is computed as the integral of the instant energy consumption in Joule (J).

We run our experiments on a Ubuntu 12.04 OS, kernel 3.5.0. The processor used is a Intel Core i7, which is a quad-core CMP processor. Each core supports multi-threading which we kept switched on during the measurments. To avoid start-up anomalies in the recorded power, we warm-up the system with two unrecorded runs for each benchmark before the initiating the measurement.

5 RESULTS

Here, we present recorded power consumption of the benchmarks and our a priori estimations. Our evaluation is in terms of speed-up and a notion of scaling in power which we call green-up. The definition of green-up (G_n) is similar to that of speed-up

$$G_n = \frac{E_1}{E_n},$$

where E_n is the total energy consumed by n threads in the experiment and E_1 is the energy consumed by 1 thread.

Since performance is closely tied to energy we present our results of performance and energy in a an envelope that shows the maximum and minimum achieved results. The figure should be interpreted as indicating the behaviour of this particular system, however our methodology is able to capture the long-term dynamic behaviour of similar CMP as most of processors used in high-performance computing fall into the same category of hierarchical memory systems with a few complex cores.

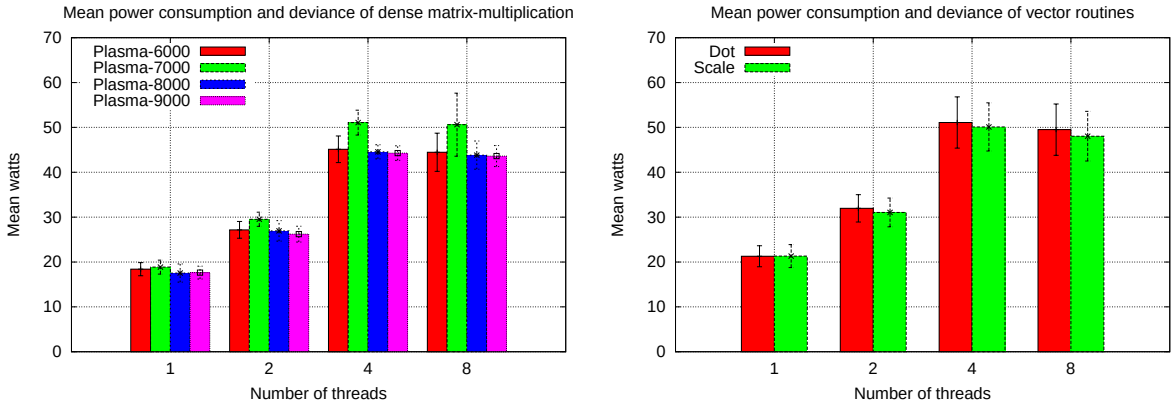
5.1 Speed-up and Green-up

With sparse and dense benchmarks we recorded very different results in power consumption. Processors of the type that study here typically have the property of *race to halt*. This means that power consumption is proportional to runtime. We can see the same pattern for our results in Figure 1(a) and 1(b). The speed-up envelop has a characteristic linear increase for compute bounded benchmark while the bandwidth bounded display a decaying maximum value with increasing threads. Bandwidth bounded benchmarks also suffer from very low speed-up as seen in 1(a) and we make the observation that the degree of boundedness is very important for performance and consequently also power consumption of bandwidth bounded operations.

When we look at energy in Figure 1(b) we see that we reach peak green-up of the bandwidth bounded benchmarks already with two threads. Also, the minimum green-up with this configuration consume more energy than the baseline measurement using one thread because according to Figure 1(a) the least performing benchmark has no speed-up using two threads.

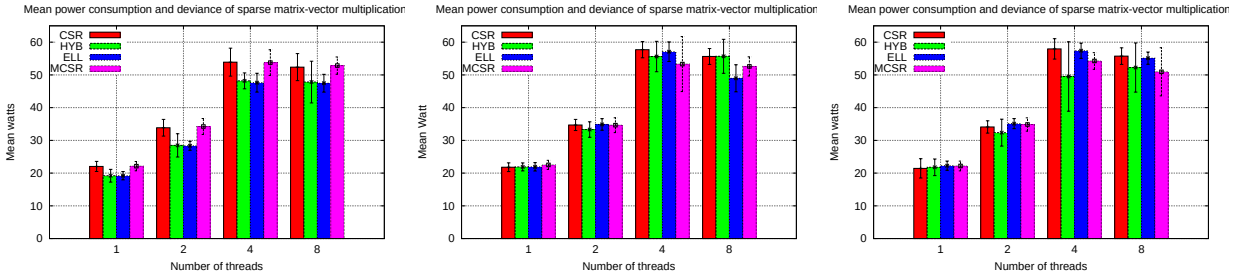
5.2 Mean power measurements

We now study the performance and power consumption in more details. Figure 2(a) presents the recorded mean power consumption and deviation of our matrix multiplication benchmark. The equivalent results of the vector and scalar multiplication operation benchmarks are presented in Figure 2(b). Note the slightly higher mean consumption and deviation of the vector operations. We see the same increase in consumption for the sparse matrix vector multiplication presented in Figure 3(a), 3(b) and



(a) (b)

Figure 2: Mean power consumption and deviation (error bars) in benchmarks matrix-multiplication (left), dot product and scalar vector multiplication (right).



(a) `FEM_3D_thermal2` (b) `apache2` (c) `ecology2`

Figure 3: Mean power consumption and standard deviance of matrix-vector operations for many sparse matrix formats CSR (red) HYB (green) ELL (blue) MCSR (purple)

3(c). Note however that the mean and deviation very much depends on the choice of sparse matrix storage format. In all figures we include the results with eight threads to show the impact of multi-threading. We do not expect the performance to increase beyond the number of physical cores (four), however, some performance can be gained if the processor can interleave tasks and there exist available jobs for a pending thread.

5.3 Estimations and comparison

In the previous section we outlined our methodology for taking mean power consumption measurements and compute a best-fit estimator for the compute and bandwidth bounded benchmarks respectively. In the tables we present next we refer to this best-fit value by $\hat{\omega}$. As stated earlier, we use the least-square-error technique to obtain this value. This approach is based on the assumption that each of the two classes of benchmarks have different dynamic power behaviour because they achieve different processor utilization.

We present the $\hat{\omega}$ estimator and the error between total power consumption and our prediction. The error is presented in terms of both mean square error (MSE) and relative error (R.Error). Again our $\hat{\omega}$ value which we derive using our methodology described in the previous section is referred to by $\hat{\omega}$.

Similar to the envelopes in Figure 1(a) and 1(b) we first present Tables 1 and 2 of our predictions of compute and bandwidth bounded benchmarks. Table 1 shows our $\hat{\omega}$ achieved with 1, 2, 4 and 8 threads for the bandwidth bounded benchmarks (refer to figure 2(a), 2(b) for comparison). In the second and third column our evaluation shows the errors obtained. Comparing with the compute bounded benchmarks in Table 2, we see that both values of $\hat{\omega}$ are fairly similar while the error increases to at most 13.7% compared with bandwidth bounded predictions. In our experiments we have observed that the error for these detailed predictions can vary with 8% depending which can be attributed to the implementation. However as we shall see next these errors have little impact when we consider the algorithm as a whole.

We now provide a prediction for the CG algorithm.

Table 1: Evaluation of **bandwidth bounded** predictions.

Bandwidth bounded estimates			
Threads	$\hat{\omega}$	**MSE**	**R.Error**
1	21.495	9416	8.9
2	32.992	9447	8.8
4	53.395	9496	9.1
8	51.785	9500	9.5

Table 2: Evaluation of **compute bounded** predictions.

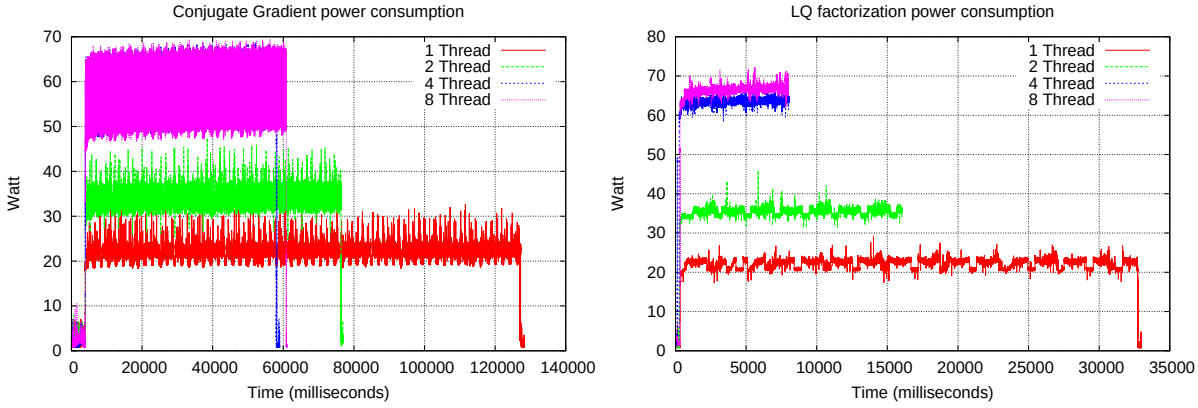
Compute bounded estimates			
Threads	$\hat{\omega}$	**MSE**	**R.Error**
1	19.499	2426	8.1
2	30.451	2137	10.0
4	52.149	2134	11.2
8	53.434	2426	13.7

Table 3 shows the predictions of total power consumption using the sparse matrix `ecology2`. For reference, we obtained similar (or better) results with `apache2` and `FEM_3D_thermal2` matrices. A figure of the instant power consumption during the whole execution of 1, 2, 4 and 8 threads is presented in Figure 4(a). Table 3 shows how the prediction improves when we consider this algorithm as a whole.

Furthermore, we provide a similar study for the factorization of dense matrices. The size of the square matrix is 4000. In Table 4 again the results improved considerably over the detailed predictions which we computed for the benchmarks. We provide also the instant power consumption during execution of a LQ factorization presented in Figure 4(b). Note in particular the much higher mean power $\hat{\omega}$ for the factorizations in Table 4.

6 CONCLUSIONS

In this paper we have shown that mean power consumption can estimate the total power consumption of dense and sparse linear solvers. Furthermore, we have shown that the green-up and the total power consumption of the solvers can be estimated a priori



(a) (b)

Figure 4: CG (left) and LQ (right) power consumption over time

Table 3: Evaluation of CG predictions using $\hat{\omega}$ derived from bandwidth bounded benchmarks.

Conjugate Gradient estimates			
Threads	$\hat{\omega}$	**MSE**	**R.Error**
1	21.495	4190	2.4
2	32.992	10192	4.0
4	53.395	42436	6.5
8	51.785	109246	10.0

Table 4: Evaluation of LQ, QR and Cholesky predictions .

Factorizations			
Threads	$\hat{\omega}$	**MSE**	**R.Error**
1	21.349	196	2.5
2	34.439	156	3.9
4	60.001	251	4.0
8	63.841	141	3.2

by the use benchmarks. Based only on runtime we can predict total consumption for both compute and bandwidth bounded applications within 5-14% error. We have extended the results from benchmarks to dense and sparse linear solver with the same small error.

The dense linear solvers are based on compute bounded algorithms and our results show that they provide close to linear speed-up as expected with 1,2 and 4 threads. These solvers also green-up when executed on a common multi-core processor, their power consumption improves up to 2 times. In contrast, the iterative sparse methods are bandwidth bounded and due to the utilization of the bandwidth of the modern multi-core processors they do not speed-up linearly. Therefore, they are faster but not always energy efficient when performed on multiple cores. Fluctuations can occur due to the efficiency and structure of the implementation for solvers and benchmarks we have tested. To provide comprehensive results we have performed various tests and we present these results using speed-up and green-up envelops.

This work is a general survey of the power consumption of linear solvers on the modern multi-core processors. We hope that this work can be used for the chip manufactures and the software developers to gain intuition of their hardware and software to deliver higher power efficiency as this has become the limiting factor for future processor designs.

ACKNOWLEDGEMENTS

This work was supported by the Swedish Research Council and the EU Commission FP7-ICT project LPGPU Contract no. 288653. It was carried out within the Linnaeus centre of excellence UPMARC, Uppsala Programming for Multicore Architectures Research Center.

REFERENCES

[1] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, (New York, NY, USA), pp. 13–23, ACM, 2007.

[2] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003.

[3] J. W. Demmel, *Applied Numerical Linear Algebra*. Philadelphia, PA: SIAM, 1997.

[4] `http://icl.cs.utk.edu/plasma`, 2007. Last visited 2013.

[5] D. Lukarski, "PARALUTION project." `http://www.paralution.com`, 2012.

[6] H. Ltaief, P. Luszczek, and J. Dongarra, "Profiling high performance dense linear algebra algorithms on multicore architectures for power and energy efficiency," *Computer Science - R&D 2012*, vol. 27, no. 4, pp. 277–287, 2012.

[7] R. Vuduc and J. W. Choi, "A roofline model of energy," December 2012.

[8] V. Spiliopoulos, A. Sembrant, and S. Kaxiras, "Power-sleuth: A tool for investigating your programs power behavior," in *Int. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2012.

[9] V. Spiliopoulos, S. Kaxiras, and G. Keramidas, "Green governors: A framework for continuously adaptive dvfs," in *Green Computing Conference and Workshops (IGCC), 2011 International*, pp. 1 –8, july 2011.

[10] R. Joseph and M. Martonosi, "Run-time power estimation in high-performance microprocessors," 2001.

[11] G. Contreras, "Power prediction for intel xscale processors using performance monitoring unit events," in *In Proceedings of the International symposium on Low power electronics and design (ISLPED*, pp. 221–226, ACM Press, 2005.

[12] T. A. Davis, "The university of florida sparse matrix collection," *NA DIGEST*, vol. 92, 1994.