Available on-line at www.prace-ri.eu

Partnership for Advanced Computing in Europe

An Energy-centric Study of Conjugate Gradient Method

Konstantinos Nikas[1,*], Dimitris Siakavaras[1], Vasileios Karakasis[1], Jan Christian Meyer[2], and Lasse Natvig[3]

[1]Greek Research & Technology Network (GRNET), Greece
[2]High Performance Computing Section, IT Dept., NTNU, Norway
[3]Dept. of Computer and Information Science (IDI), NTNU, Norway

Abstract

This whitepaper focuses on the study of the conjugate gradient method and how storage formats for sparse matrices have a significant impact on its performance and energy footprint. We perform the evaluation on a 32-core, NUMA platform that provides energy measurements for the processors and the main memory. Our study reveals interesting aspects of the execution of memory bound applications on state-of-the art multicore platforms which could be utilised by an automatic tuning process towards a more energy efficient execution.

1. Introduction

The most widely used storage format for non-special (e.g., diagonal) sparse matrices is the Compressed Sparse Row (CSR) format. CSR compresses the row indexing information needed to locate a single element inside a sparse matrix by keeping only *number-of-rows* 'pointers' to the start of each row (assuming a row-wise layout of the non-zero elements) instead of *number-of-nonzeros* indices. However, there is still a lot of redundant information contained in the column indices, which CSR keeps intact in favor of simplicity and straightforwardness. For example, it is very common for sparse matrices, especially those arising from physical simulations, to have sequences of continuous non-zero elements. In such cases, it would suffice to store just the column index of the first element and the size of the sequence. CSX [1] goes even further by replacing the column indices with the delta distances between them, which can be stored with one or two bytes in most of the cases, instead of the typical four-byte integer representation of the full column indices.

In our previous work [2, 3], we evaluated the energy and performance impacts of CSR and CSX. More specifically, we instrumented the execution of a conjugate gradient solver and extracted running power estimations from hardware performance counters. These were used to produce power/time curves, which, when integrated over particular intervals, estimate the energy consumption of the individual application stages. Our goal here is to gain a deeper insight on the energy consumption breakdown of the system; to do so, we extend our observations to a significantly greater number of cores than before, incorporated into a state-of-the-art NUMA machine.

The rest of the paper is organized as follows: Section 2. presents the platform, the test matrices and the instrumentation used in our experiments, Section 3. presents our findings and Section 4. concludes and highlights possible directions for future work.

2. Experimental Methodology

2.1. Hardware Platform

All experiments are executed on a quad-socket platform, which contains 4 Intel® Xeon E5-4620 processors. The details of this Sandy Bridge-based processor are presented in Table 1, while the platform architecture is depicted in Fig. 1. As it can be seen, our test platform is a NUMA machine with a total of 32 physical cores and 256GB of main memory.

The Sandy Bridge architecture offers the Turbo Boost technology [4], which converts thermal headroom to higher performance by enabling the processor to run above its base operating. However, in this work all the experiments are executed with Turbo Boost disabled and every processor clocked at the maximum allowed frequency (2.2GHz).

*To whom correspondence should be addressed. Email: knikas@cslab.ece.ntua.gr

CPU Model	Intel Xeon E5-4620
Model #	45
Stepping	7
Manufacturing process	32nm
Clock frequency (min-max)	1.20GHz - 2.20GHz
Number of physical cores	8
Number of logical cores	16
L1 cache	32KB, private
L2 cache	256KB, private
L3 cache	16MB, shared

Table 1: Processor specifications

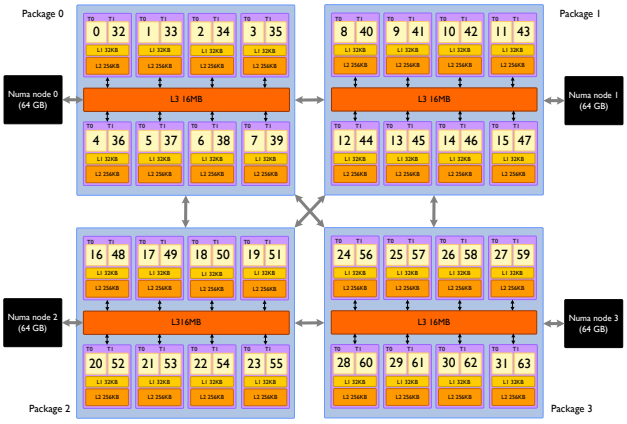


Fig. 1: Test platform

2.2. Test Matrices

For our tests, we use 10 matrices from the University of Florida Sparse Matrix Collection [5]. The chosen matrices have different sizes and can be classified as regular or irregular, based on the extent to which their nonzero components are regularly arranged in patterns making them suitable for compression. The characteristics of the matrices are summarized in Table 2.

Matrix	Rank	Type
af_5_k101	503.625	Regular
bmwcra_1	148.770	Regular
boneS10	914.898	Regular
kkt_power	2.063.494	Regular
ldoor	952.203	Regular
F1	343.791	Irregular
G3_circuit	1.585.478	Irregular
offshore	259.789	Irregular
parabolic_fem	525.825	Irregular
thermal2	1.228.045	Irregular

Table 2: Test matrices.

The selected matrices are used in a conjugate gradient solver with a fixed number of iterations (1024). This number of iterations may not suffice to bring our test system to convergence, but it reveals a steady-state power consumption in the iteration phase, as it is shown in the results presented in Section 3.

2.3. Instrumentation

Sandy Bridge based server platforms provide a running estimate of the energy consumption not only for each processor package, but also for the DRAM modules. Therefore, we extend the instrumentation used in [2] to read the appropriate Model Specific Registers (MSRs) and record the energy consumed by the following components:

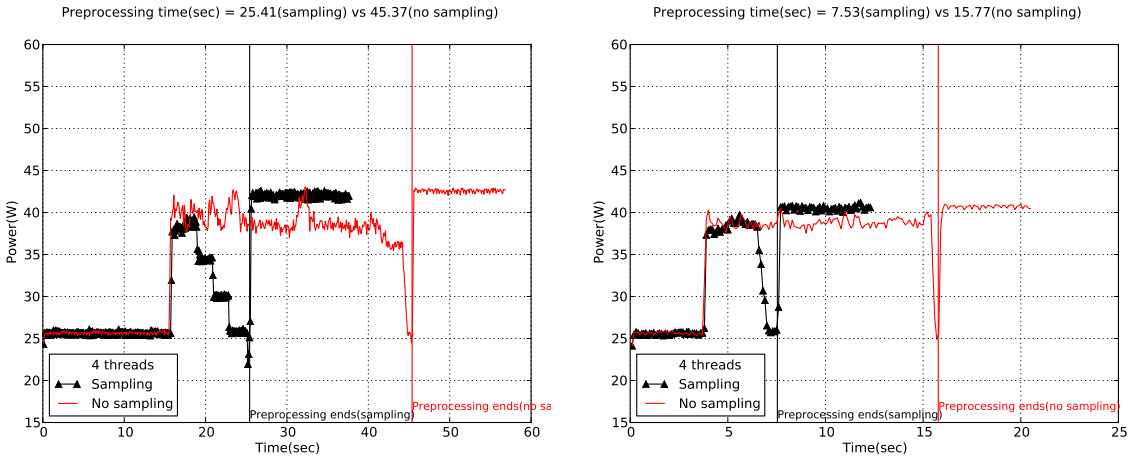
1. **Cores**: The energy consumed by the cores and their L1 and L2 caches. Unfortunately, the architecture tracks the energy usage on a package granularity. Therefore, we cannot acquire the actual energy consumption of each core.
2. **Package**: The energy consumed by a single physical package.
3. **DRAM**: The energy consumed by the DRAM modules of each package.

Moreover, from the package and cores' energy consumption we can deduce the energy consumed by the uncore part of the package, which, among other components, includes the Last Level Cache (LLC), the on-die memory controller and the QPI controllers.

3. Results and Discussion

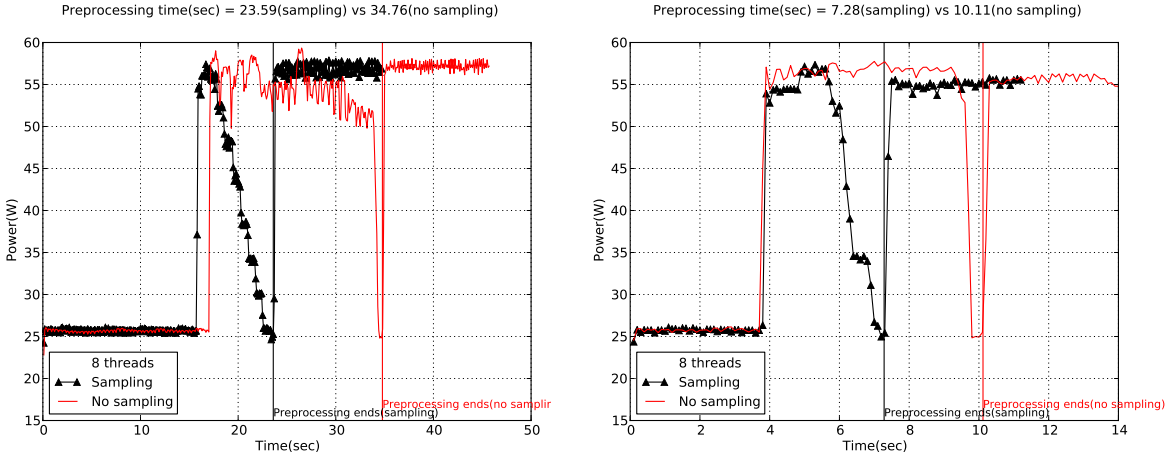
3.1. Statistical Preprocessing

CSX has a sampling preprocessing feature which can significantly reduce the substructure detection phase of the preprocessing stage. As previous work [2] was performed without enabling this feature, we evaluate it here.



(a) af_5_k101 (b) offshore

Fig. 2: Detailed view of CSX execution with 4 threads with and without statistical preprocessing



(a) af_5_k101 (b) offshore

Fig. 3: Detailed view of CSX execution with 8 threads with and without statistical preprocessing

Fig. 2 and 3 present the initialisation, preprocessing and iteration stages for two different matrices, with and without sampling, for 4 and 8 threads respectively. It is evident that statistical preprocessing makes CSX more efficient both in terms of performance and energy.

Table 3 presents the time, as well as the energy spent in the preprocessing stage for each one of the selected matrices for different number of threads. As the number of threads increases and the preprocessing effort is split between the parallel threads, the performance and energy gains of enabling statistical preprocessing diminish. In fact, there are 3 matrices, namely G3_circuit, parabolic_fem and thermal2, for which statistical preprocessing with 8 threads takes more time and energy compared to when no sampling is used.

3.2. CSX-CSR comparison

Fig. 4 depicts the energy consumption breakdown of CSX and CSR iteration phase for all matrices and threads from 1 up to 32, as well as their speedup with respect to the execution time of CSR running with 1 thread. All the runs employ what we call *packed placement policy*, i.e. we employ all the physical cores available in one physical package before assigning a thread to another physical package. So, when we use 8 threads, all of them are assigned to cores on physical package 0; when we use 16 threads they are assigned to physical packages 0 and 1, and so forth. The reported total energy is the total energy of the system including all the DRAM modules and all the packages, irrespective of whether they are actually used or they are idle.

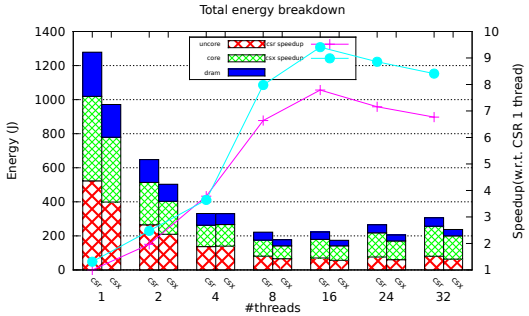
As it was also observed in [2], there is clear evidence that better performance translates to reduced energy consumption. For the majority of cases CSX achieves better performance than CSR and thus is more energy efficient. The breakdown of the consumed energy reveals that as more threads are utilised the majority of the energy is consumed by the cores and their caches, while the least energy is consumed by the DRAM modules.

Matrix	Threads	Time (sec)			Energy (J)		
		Sampling	No Sampling	%	Sampling	No Sampling	%
af_5_k101	1	31.8670	133.0689	76.05	821.5822	3438.9735	76.10
	4	25.4083	45.3704	43.99	715.0702	1543.1236	53.66
	8	23.5933	34.7640	32.13	738.5381	1386.1543	46.72
bmwcra_1	1	15.9139	78.3676	79.69	406.0751	2011.7216	79.81
	4	12.8248	26.7548	52.06	359.7530	906.1603	60.29
	8	12.9937	24.1671	46.23	403.1264	1009.4537	60.06
boneS10	1	87.0062	443.7592	80.39	2241.3116	11456.5964	80.43
	4	57.5111	133.1319	56.80	1636.5812	4615.0765	64.53
	8	55.0448	109.6157	49.78	1706.1398	4711.4802	63.78
kkt_power	1	38.1075	149.4080	74.49	974.8047	3845.5118	74.65
	4	32.0096	53.2843	39.92	1043.8734	1839.2230	43.24
	8	31.1458	34.7125	10.27	1296.4859	1498.7568	13.49
ldoor	1	65.6666	301.5339	78.22	1685.5137	7783.4536	78.34
	4	59.5436	106.4609	44.06	1701.1027	3611.5231	52.89
	8	57.8411	82.2420	29.66	1859.6330	3363.9663	44.71
F1	1	43.8309	215.3817	79.64	1119.9978	5535.3154	79.76
	4	34.0371	90.3744	62.33	993.4615	3088.7248	67.83
	8	33.1719	66.7586	50.31	1099.7078	2838.2399	61.25
G3_circuit	1	20.3693	61.7549	67.01	521.1307	1585.8950	67.13
	4	19.3826	22.2740	12.98	623.3594	745.3431	16.36
	8	19.2523	15.7539	-22.20	761.0053	649.6169	-17.14
offshore	1	9.8932	49.3502	79.95	251.3977	1266.4788	80.14
	4	7.5321	15.7714	52.24	227.9116	556.5390	59.04
	8	7.2774	10.1135	28.04	260.0583	439.7696	40.86
parabolic_fem	1	12.1224	43.0467	71.83	308.4652	1102.6851	72.02
	4	11.5843	17.0763	32.16	379.1447	577.7249	34.37
	8	10.8325	10.7845	-0.44	469.9097	454.3731	-3.41
thermal2	1	28.5477	106.6851	73.24	730.5855	2744.4493	73.37
	4	24.0816	33.7634	28.67	828.4820	1198.6552	30.88
	8	23.8453	22.8178	-4.50	1083.5644	1007.6155	-7.53

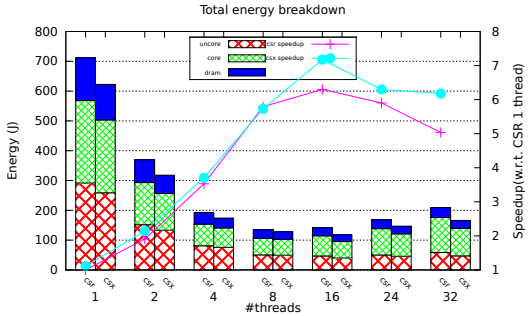
Table 3: Comparison of time and energy spent in the preprocessing stage with and without sampling

In general, all the aforementioned results indicate that, for the majority of cases, the most energy efficient setup is the one that utilises all the available resources, i.e. all the 32 cores. One could argue that this is anticipated, as the processor packages consume energy even when they are idle. Therefore, as long as idle power accounts for a significant part of the overall consumption, it is unconditionally beneficial to utilise them for performance improvements. However, things would be different if we were able to shut down the parts of the system that are not used. In this scenario, the extra resources needed to improve performance would be more costly in terms of energy than before, as they would not contribute to the total power consumption when they are not used.

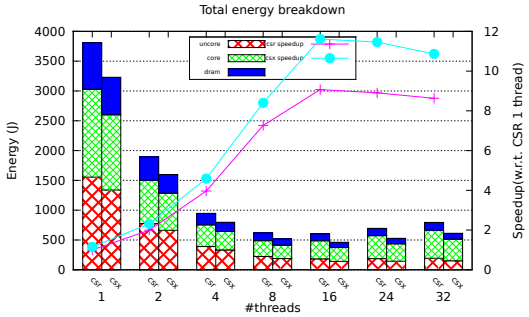
Fig. 5 plots again the energy consumption of CSX and CSR, but now without taking into account the energy consumed by the idle packages, thus emulating a system where idle packages are shut down. As it is evident in the graph, the setup that achieves the best performance is no longer the most energy efficient one. In fact, for all the matrices, we achieve the lowest energy consumption when we utilise only one of the available packages and even in some cases a subset of the package.



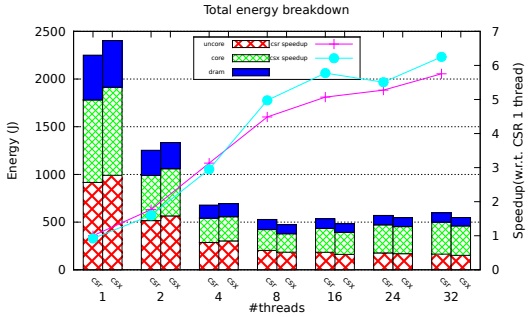
(a) af_5_k101



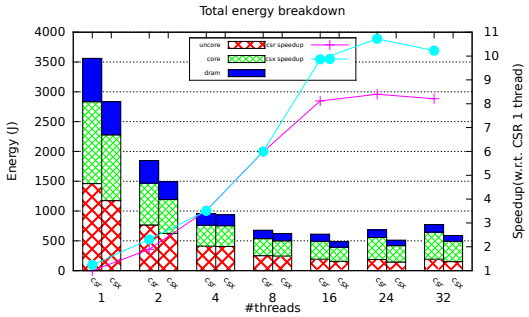
(b) bmwcra_1



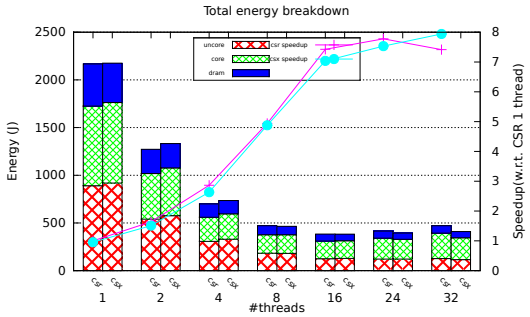
(c) boneS10



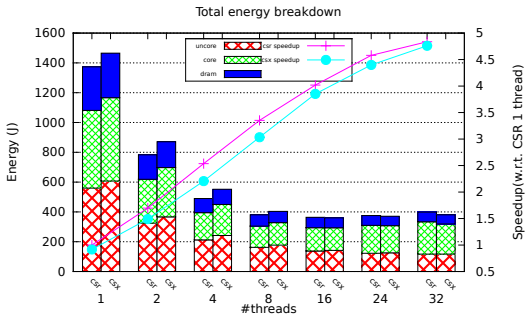
(d) kkt_power



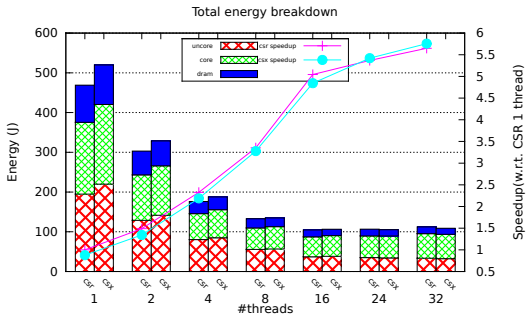
(e) ldoor



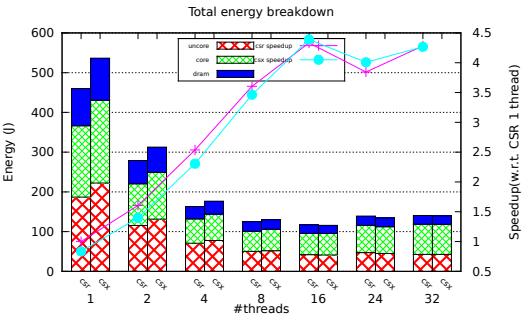
(f) F1



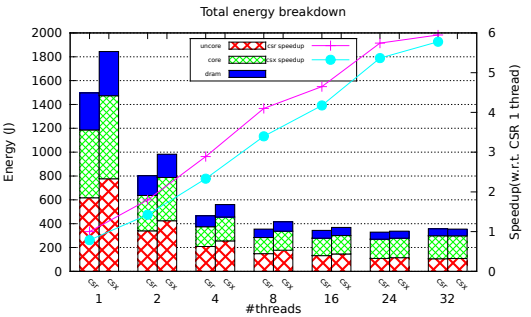
(g) G3_circuit



(h) offshore

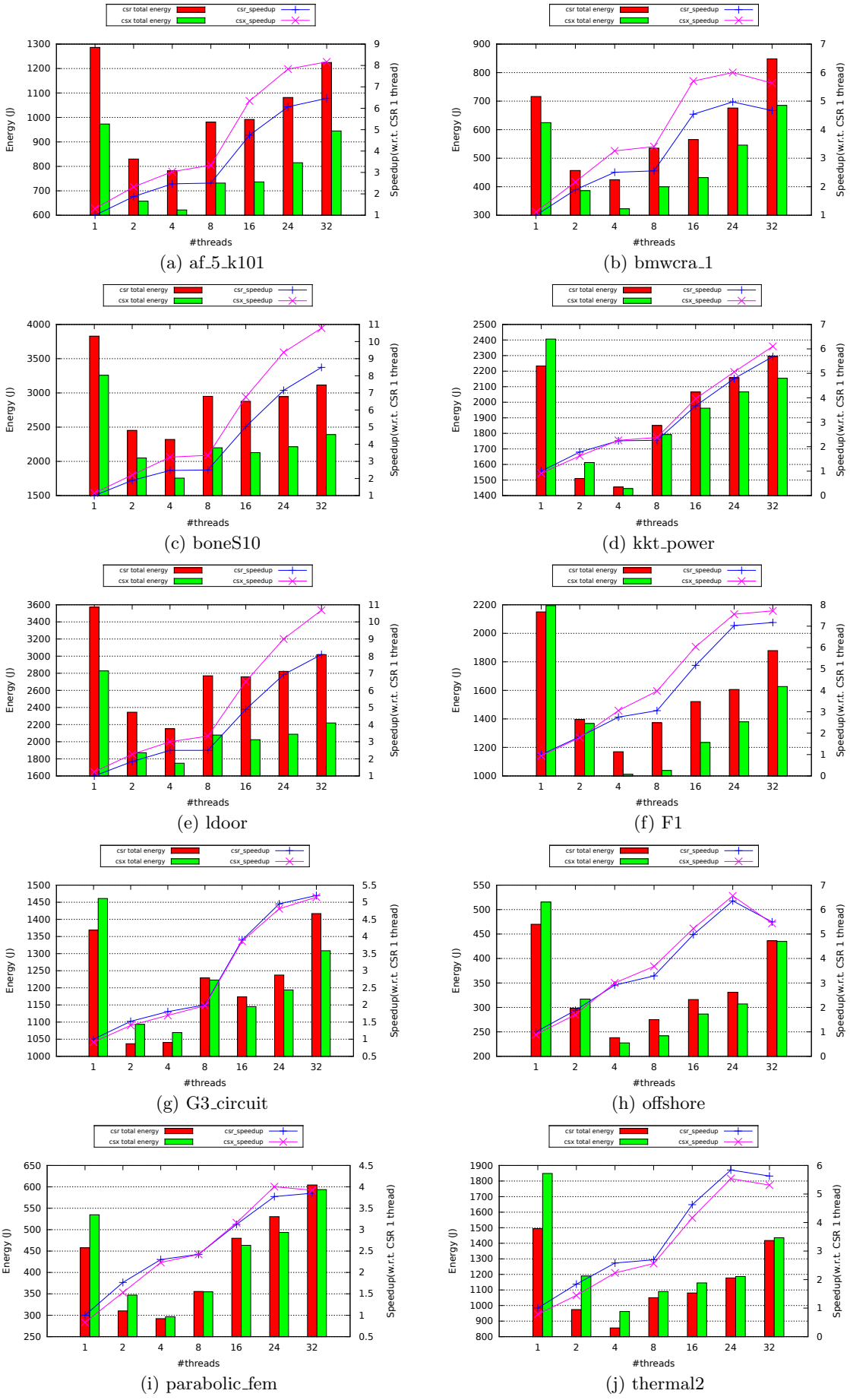


(i) parabolic_fem



(j) thermal2

Fig. 4: CSX and CSR iteration phase energy consumption breakdown and speedup



(a) af_5_k101 (b) bmwcra_1

(c) boneS10 (d) kkt_power

(e) ldoor (f) F1

(g) G3_circuit (h) offshore

(i) parabolic_fem (j) thermal2

Fig. 5: CSX and CSR iteration phase energy consumption and speedup taking into account only active packages

3.3. Power and Scalability

Fig. 7 plots the average power consumption of the different components for the iteration phase of CSX executed with various number of threads. The plotted values are calculated for each component by averaging the power recorded from all the active packages over the execution time, while the error bars show the minimum and maximum values.

The graph reveals three facts that remain true across all matrices. First of all, the power consumption of the uncore part of the package is around 15 Watts, stable throughout the execution, and does not vary with the number of active cores inside the package, as it can be seen for threads 1 up to 8. Second, the power consumption of the DRAM modules tops at around 10 Watts and, unlike the uncore component, depends on the number of cores utilised inside the package. As the number of threads increase from 1 to 8, the amount of memory requests in the unit of time increases, causing a rise in the power consumption. When more packages are utilised (for thread numbers greater than 8), the load is split between the different packages, causing a drop at the activity of the DRAM module of the package in the unit time and a lower power consumption.

The power profile of the core component is not as straightforward. For threads 1 up to 8, power consumption increases as more cores are becoming active, reaching a maximum of around 40 Watts when all the cores inside the package are used. However, as more packages are becoming active, the average power consumption drops, even by around 35% at some cases when all 4 packages are employed. At the same time, as it is evident from the error bars, the actual power consumption of the core component varies significantly from the average throughout the execution and across packages.

This drop and variation could be attributed to a load imbalance caused by either the conjugate solver or the structure of the matrix. The threads are assigned different loads, which causes some of them to quickly reach the barrier before the next iteration. When a thread reaches a barrier, it yields the processor and since there are no other runnable contexts, the OS puts the processor to a lower power state. Thus, in case of load imbalance, many threads will wait for the late one to reach the barrier, causing their assigned cores to operate on a lower power state and reducing the average power consumption of the core component of the package.

This is evident in Fig. 8, which plots the average time a thread spends working and waiting at a barrier. As more threads are employed, the amount of work performed by each thread is reduced, while the time spent inside the barriers is increased. In fact, for the majority of the matrices, when 3 or 4 packages are used, the waiting time is equivalent or even higher than the working time.

However, load imbalance does not seem to be the only reason for the observed drop in the power consumption of the core component. Fig. 6 plots the average power of the different components when a dense matrix is used. Still, when more packages are employed, the average power consumption is reduced again. As now there is no load imbalance between the threads, this behaviour could be attributed to the actual implementation of the barriers and their lack of scalability. This observation signals the need for the investigation of more efficient synchronisation mechanisms.

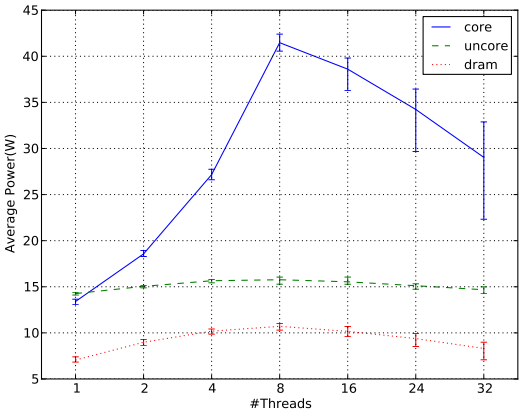
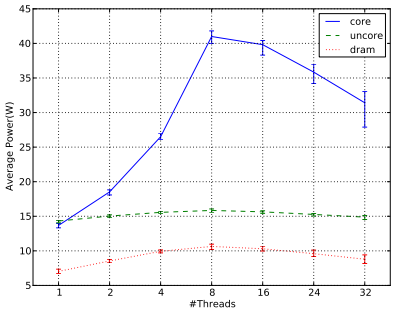
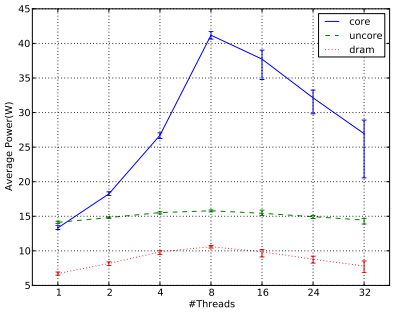


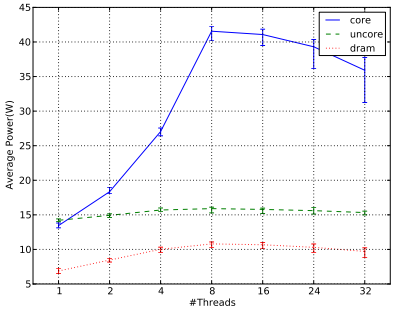
Fig. 6: CSX iteration phase average power for the core, uncore and dram components for a dense matrix



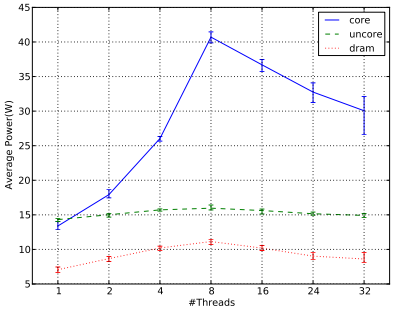
(a) af_5_k101



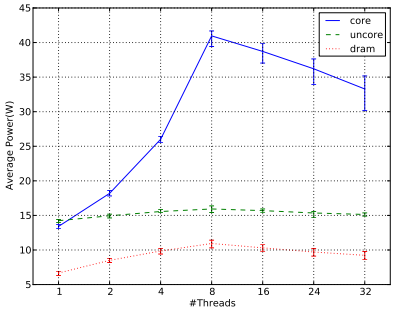
(b) bmwcra_1



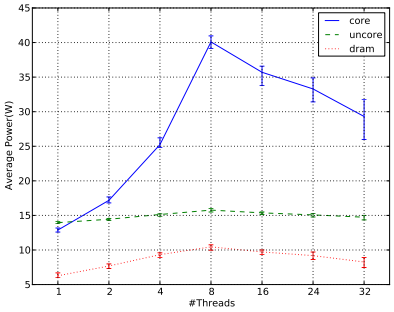
(c) boneS10



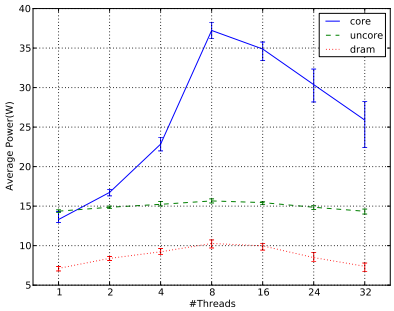
(d) kkt_power



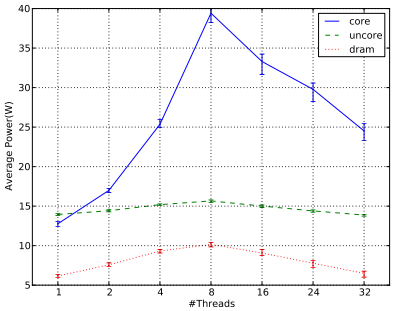
(e) ldoor



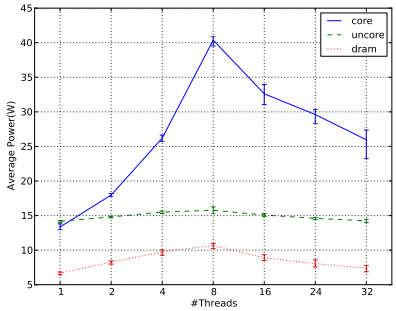
(f) F1



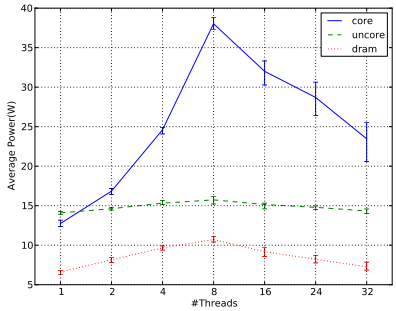
(g) G3_circuit



(h) offshore

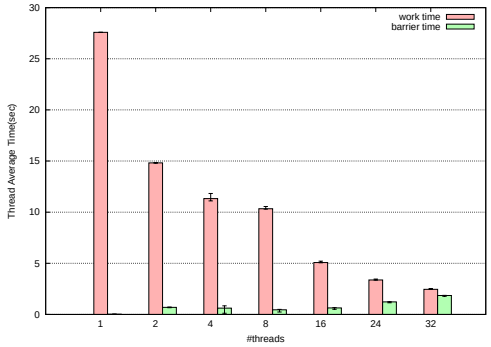


(i) parabolic_fem

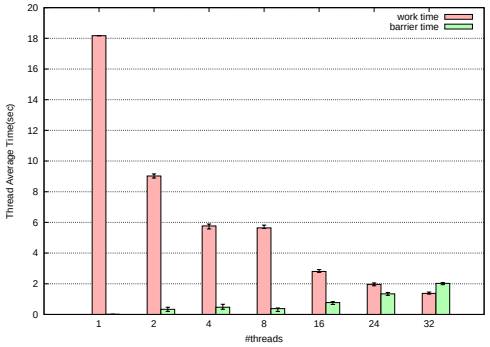


(j) thermal2

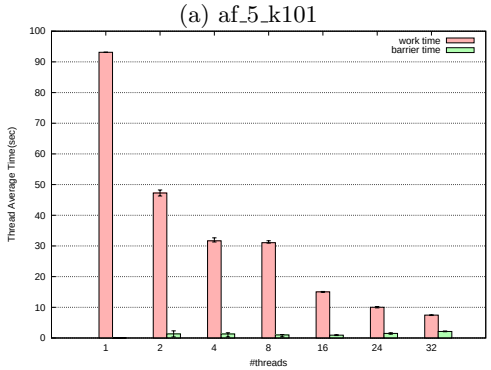
Fig. 7: CSX iteration phase average power for the core, uncore and dram components



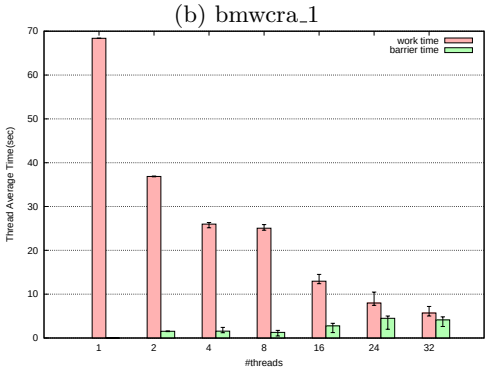
(a) af_5_k101



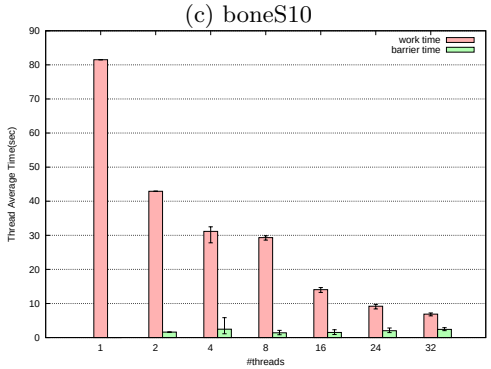
(b) bmwcra_1



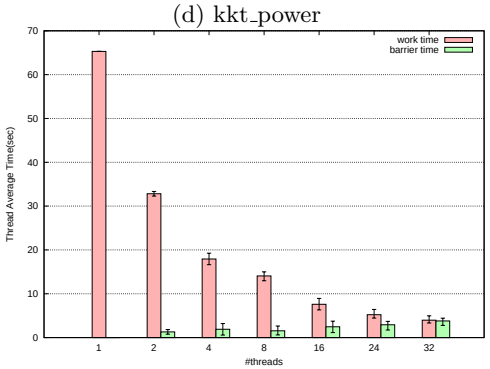
(c) boneS10



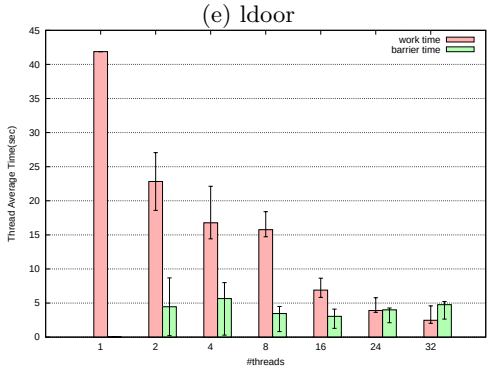
(d) kkt_power



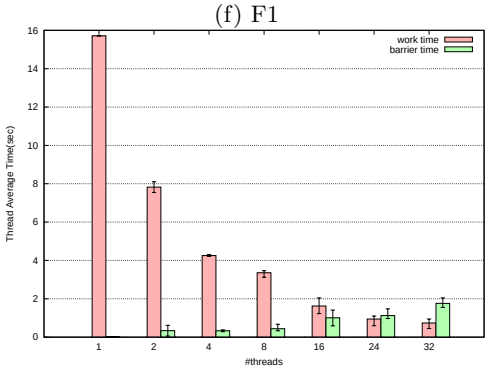
(e) ldoor



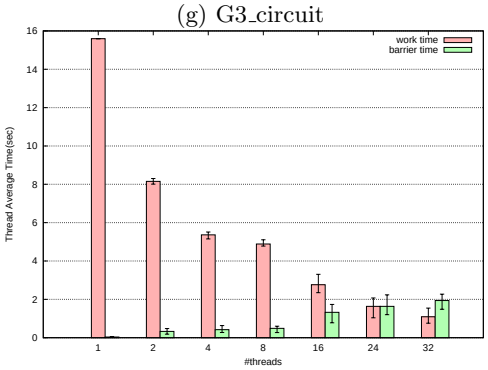
(f) F1



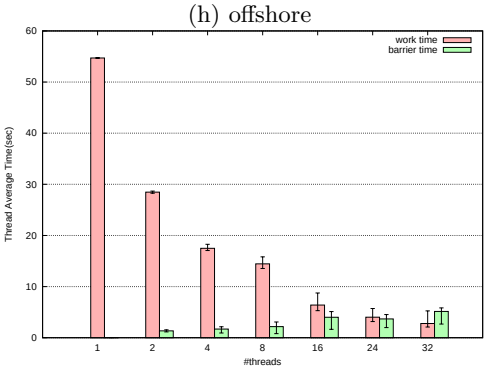
(g) G3_circuit



(h) offshore



(i) parabolic_fem



(j) thermal2

Fig. 8: Average working and waiting thread time for the CSX iteration phase

4. Conclusions

We have studied the execution of the conjugate gradient method and how the sparse matrix storage format can affect its performance and energy footprint. Our study reveals that CSX generally outperforms CSR both in terms of performance and energy consumption. However, the detection of the most energy efficient setup in terms of resources assigned to the solver, is not completely straightforward. In current platforms, where it is not possible to shut down the inactive parts of the system, the most energy efficient setup is usually the one that delivers the best performance. On the other hand, if future platforms do offer the option to shut down idle packages, then this detection will become more complex, as it will require the prediction of the idle power consumption in order to identify the most energy efficient setup.

Acknowledgments

This work was financially supported by the PRACE-2IP project funded in part by the EU's 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-283493.

References

1. K. Kourtis, V. Karakasis, G. Goumas, and N. Koziris, "CSX: An Extended Compression Format for SpMV on Shared Memory Systems," in *Proceedings of the 16th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP'11)*, (San Antonio, Texas, USA), pp. 247–256, ACM, 2011.
2. J. C. Meyer, L. Natvig, V. Karakasis, D. Siakavaras, and K. Nikas, "Energy-efficient sparse matrix auto-tuning with csx." PRACE Whitepaper, 2013. `http://www.prace-project.eu/IMG/pdf/wp57_energy_efficient_sparse_matrix_auto-tuning_with_csx.pdf`.
3. J. C. Meyer, J. M. Cebrian, L. Natvig, V. Karakasis, D. Siakavaras, and K. Nikas, "Energy-efficient sparse matrix autotuning with csx - a trade-off study," in *9th Workshop on High-Performance, Power-Aware Computing (HPPAC)*, 2013.
4. Intel Corporation, "Intel 64 and IA-32 architectures software developer's manual: Combined volumes: 1, 2a, 2b, 2c, 3a, 3b and 3c," 2013. `http://download.intel.com/products/processor/manual/325462.pdf`.
5. T. Davis, "The University of Florida sparse matrix collection." NA Digest, vol. 97, no. 23, June 1997. `http://www.cise.ufl.edu/research/sparse/matrices`.