SC14

Machine Learning Algorithms for the Performance and Energy-Aware Characterization of Linear Algebra Kernels on Multithreaded Architectures

A. Cristiano I. Malossi, Yves Ineichen, Costas Bekas, Alessandro Curioni @ **IBM Research – Zurich**
Enrique S. Quintana–Ortí @ **Universidad Jaime I**

Motivations

- *Power* and *Energy* are now primary design principles in supercomputing
- Hardware power-cap and energy saving mechanisms are only sub-optimal without re-thinking algorithms for *green-computing*
- Modeling and understanding the performance-energy interaction of the Berkeley "dwarfs" [2] represents a fundamental step toward this direction, with immediate benefits for all the existing scientific computing applications

Summary

We present two semi-automatic machine learning algorithms to measure, model, and characterize the time-power-energy *performance triangle* of linear algebra kernels:

1. A *bottom-up approach* that decomposes dense linear algebra kernels into sub-cost components to highlight power-hungry ingredients [4]
2. A combined *parameterization-classification technique* to tackle and analyze the irregular memory access pattern of sparse linear algebra operations [3]

Model applications

1. *System management* – for HPC- and cloud-systems run-time optimization
2. *Software development* – for algorithm performance analysis and modeling
3. *Hardware development* – for architectural design toward energy-efficiency

Experimental setting

Table 1: Range of tested architectures: L4-cache size is given per DIMM, other sizes are given per core.

Architecture	Cores	GHz	SMT	L1-cache [kB]	L2-cache [kB]	L3-cache [MB]	L4-cache [MB]	DDR [GB]
IBM Blue Gene/Q (BG/Q)	32×16	1.60	4	16	2048	-	-	1
IBM POWER7	4×8	3.30	4	32	256	4	-	8
IBM POWER8	2×6	3.89	8	64	512	8	16	8

We run our tests on three IBM architectures as reported in Table 1. All the routines are processed by the IBM XL C/C++ compiler.

Kernel example: inner product data acquisition

Kernel 1: benchmark implementation of the inner product kernel with unrolling factor equal to 4. ● Orange lines delimit instrumented code; ● yellow lines are present only in the register version; ● green lines only in the memory version; all other lines are always present.

```
double *x, *y; // Vectors of size N_vector, with entries in the range [-0.5,0.5]
...
omp_set_num_threads(N_threads); // Set number of threads
 #pragma omp parallel private(k,i,iStart,iStop,tID,xs0,xs1,xs2,xs3,ys0,ys1,ys2,ys3)
 #pragma omp parallel private(k,i,iStart,iStop,tID)
{
    tID    = omp_get_thread_num(); // Thread ID
    iStart = tID * N_vector / N_threads;   iStop  = (tID + 1) * N_vector / N_threads;

     xs0 = x[tID*4];     xs1 = x[tID*4+1];   ys0 = y[tID*4];     ys1 = y[tID*4+1];
     xs2 = x[tID*4+2];   xs3 = x[tID*4+3];   ys2 = y[tID*4+2];   ys3 = y[tID*4+3];

     // Start routines for time and power measurement
    for (k = 0; k < repetitions; ++k) {
        dot0 = 0.0; dot1 = 0.0; dot2 = 0.0; dot3 = 0.0;
        #pragma omp parallel reduction(+:dot0,dot1,dot2,dot3)
        for (i = iStart; i < iStop; i+=4)
            dot0 += xs0 * ys0;   dot1 += xs1 * ys1;   dot2 += xs2 * ys2;   dot3 += xs3 * ys3;
        dot = dot0 + dot1 + dot2 + dot3;
        dot = 0.0;
        #pragma omp parallel reduction(+:dot)
        for (i = iStart; i < iStop; ++i)
            dot += x[i] * y[i];
    }
     // Stop routines for time and power measurement
}
```

- Kernel execution is repeated many times to obtain enough samples
- Impact of memory accesses is isolated comparing register and memory versions
- Cost of memory access at different levels is obtained tuning the vector size
- Cost of reduction is isolated removing the "#pragma omp parallel reduction"
- Power measures are obtained from hardware sensors

Dense linear algebra kernels: decomposition and regression analysis

The energy E consumed by a generic operation is given by

$$E = \int_0^T P(t)\,\mathrm{d}t = T\bar{P} \qquad t \in [0,T]$$

where $\bar{P}$ is the average power in the execution time period T. Given a generic kernel "ker", we decompose its execution time and average power consumption into base *cost components*, i.e.,

$$T^{\text{ker}} = \sum_{k\in\mathcal{C}} \left[T^{\text{ker}}\right]_k \qquad \bar{P}^{\text{ker}} = \sum_{k\in\mathcal{C}} \frac{\left[T^{\text{ker}}\right]_k \left[\bar{P}^{\text{ker}}\right]_k}{T^{\text{ker}}}$$

where each component in the $\mathcal{C}$ list is expressed as function of a few parameters. For example, to model the CG method (with dense matrix) we employ six components:

1. *arithmetic* – for the floating-point multiply-add (FMA)
2. *memory access* – for load/store from/to cache or RAM
3. *repetition* – for an auxiliary loop that embraces the kernel
4. *multithreading* – for accounting more threads per core
5. *reduction* – for parallel reductions across threads
6. *barrier* – for parallel synchronizations across threads

These components are in turn parameterized with respect to:

1. the size of the kernel data set (e.g., in a vector operation, the number of elements in the vector)
2. the number of cores and OpenMP threads
3. the number of times that the kernel is executed (e.g., in the CG method, the number of iterations to converge)

Cost model example: inner product parallel reduction

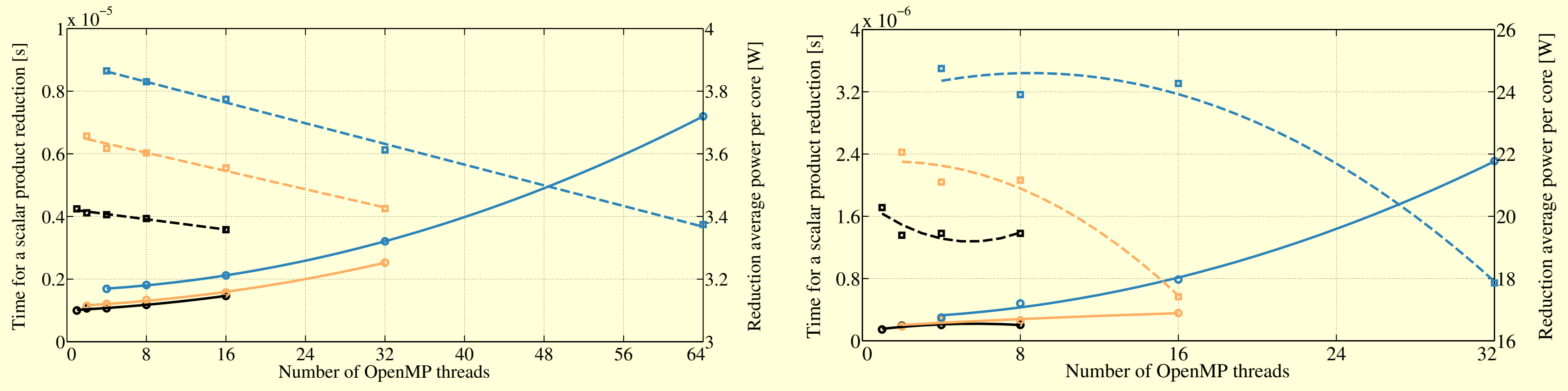


Figure 1: Linear and quadratic regression polynomial models for the reduction time (solid lines) and average power per core (dashed lines) on the IBM BG/Q (left) and POWER7 (right). Measures are identified by bold markers. Results for 1, 2, and 4 threads per core are in ● black, ● orange, and ● blue color, respectively.

Dense linear algebra kernels: measures, predictions, and validation

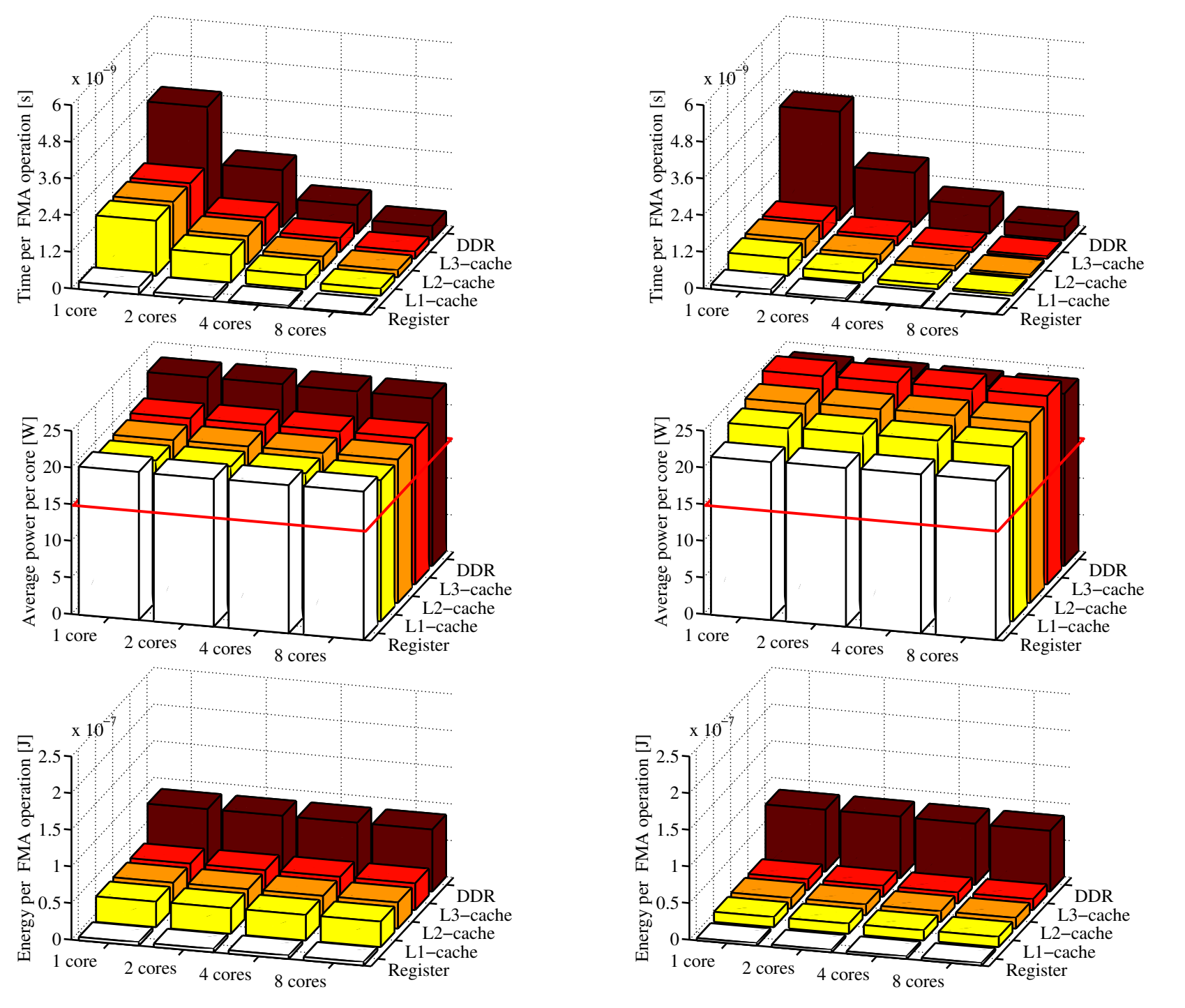


Figure 2: IBM POWER7 inner product kernel measures *without reduction* using 1 (left) and 4 (right) threads per core. Red lines indicate idle power.

- Accurate time prediction (average error ≈ 2%) and total power prediction (max. error ≈ 7%), see Table 2
- Highlight cost of components vs. parameters
- Detect lower-level power-hungry ingredient

Table 2: IBM POWER7 measures and model relative errors for the inner product kernel *with reduction* using 1 and 4 threads per core. The overall average relative errors for time and power quantities are 2.32% and 0.95%, respectively.

	Measures								*Prediction relative error*							
	T_{FMA} $[\text{s}\cdot 10^{-9}]$				$\bar{P}_c$ [W]				T_{FMA} [%]				$\bar{P}_c$ [%]			
Cores	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8
1 thread per core																
Register	0.23	0.11	0.06	0.03	20.4	20.4	20.4	20.4	0.1	0.1	0.3	0.4	0.0	0.2	0.1	0.1
L1−cache	2.07	1.10	0.55	0.28	19.4	19.3	19.3	19.3	1.8	1.4	1.0	0.1	0.1	0.3	1.4	2.2
L2−cache	1.86	0.94	0.47	0.23	19.6	19.5	19.5	19.5	0.3	0.5	0.1	0.1	0.0	0.0	0.0	0.0
L3−cache	1.83	0.91	0.46	0.23	19.9	19.9	19.9	19.9	0.2	0.3	0.3	0.4	0.9	0.8	0.7	0.9
DDR	3.71	1.85	0.93	0.46	23.0	23.0	22.9	22.8	0.0	0.0	0.0	0.1	0.1	0.1	0.2	0.1
4 threads per core																
Register	0.15	0.08	0.04	0.02	21.5	21.6	21.7	21.7	0.3	0.2	1.0	3.8	1.9	1.4	1.0	0.3
L1−cache	1.09	0.68	0.55	0.73	24.0	23.3	20.5	18.0	11.0	3.6	1.2	13.0	1.6	3.8	2.2	0.3
L2−cache	0.71	0.36	0.18	0.12	24.3	24.3	24.2	22.6	4.9	4.4	8.0	22.0	0.6	0.6	4.4	6.7
L3−cache	0.59	0.30	0.15	0.07	25.8	25.9	25.8	25.7	2.2	2.4	1.5	2.5	0.5	0.8	1.2	1.9
DDR	3.55	1.77	0.88	0.44	23.7	23.7	23.6	23.4	0.0	0.0	0.1	0.0	0.5	0.5	0.4	0.3

CG validation and power consumption insight

Table 3: IBM POWER7 measures and model relative errors for the CG kernel using 1 and 4 threads per core. The overall average relative error is 2.49%.

	Measures [W]								*Prediction relative error* [%]							
	1 thread per core				*4 threads per core*				*1 thread per core*				*4 threads per core*			
Cores	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8
L1−cache	19.7	19.3	18.9	18.8	22.4	22.0	21.0	20.8	1.0	0.6	0.2	0.6	0.2	1.2	0.6	0.6
L2−cache	19.5	19.3	19.2	19.0	23.3	22.7	21.7	20.8	0.4	0.3	0.4	1.0	0.1	0.0	0.3	0.4
L3−cache	19.7	19.4	19.4	19.3	24.4	24.3	24.0	21.7	0.6	1.9	2.0	2.3	2.3	2.1	1.2	1.3
DDR	23.7	23.6	23.5	23.4	25.2	25.3	25.2	25.0	7.0	6.9	6.5	6.4	5.4	5.6	5.7	5.7

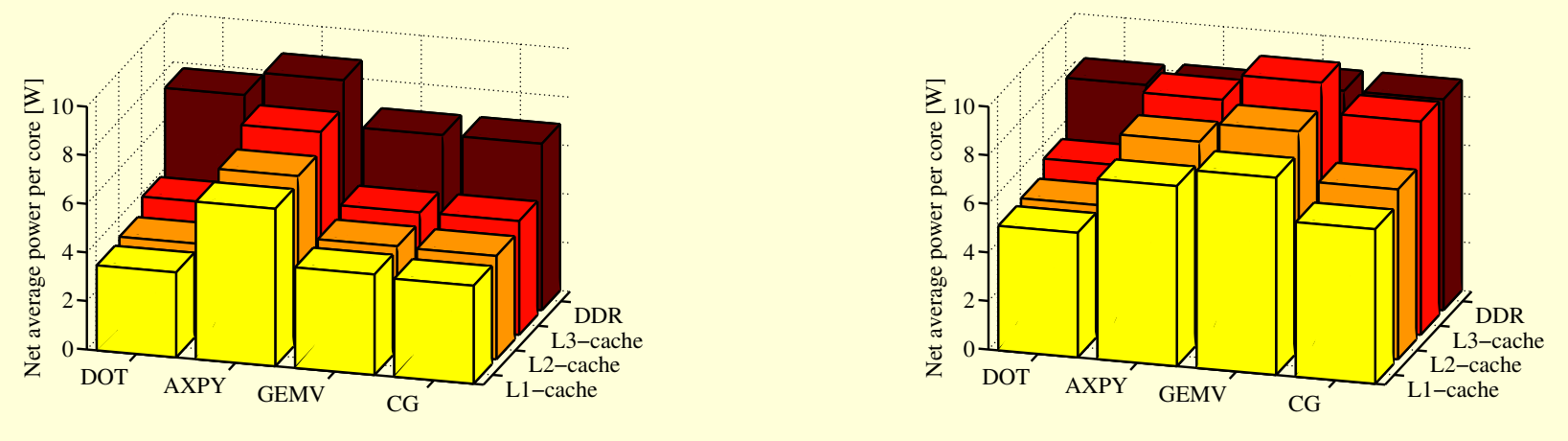


Figure 3: IBM POWER7 prediction of average net power consumption (per core) for the CG method and its sub-kernels using 1 (left) and 4 (right) threads per core. The model highlights power-hungry sub-kernels.

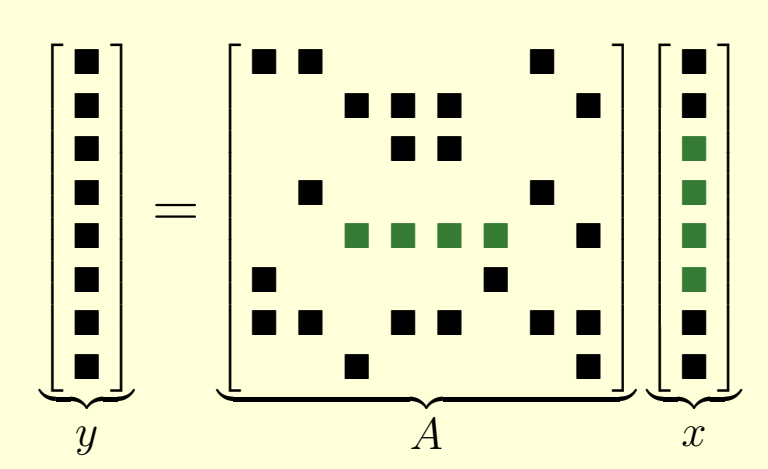
How to tackle sparse linear algebra operations?

Sparse linear algebra kernels are typically characterized by *indirect* and *irregular memory access*:

- performance is driven by sparsity pattern and system memory bandwidth
- modeling the sparse memory access is mandatory to capture the performance features of these operations

Example: parameterization of compressed sparse row (CSR)-based implementation of SpMV

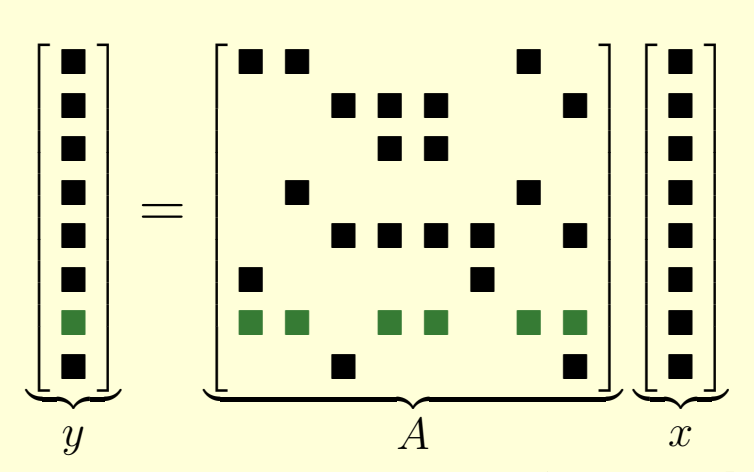
To account for the matrix sparsity pattern we introduce *three complementary parameters*:



The *block size* b_s is the number of non-zeros that are clustered into a block.

Importance:

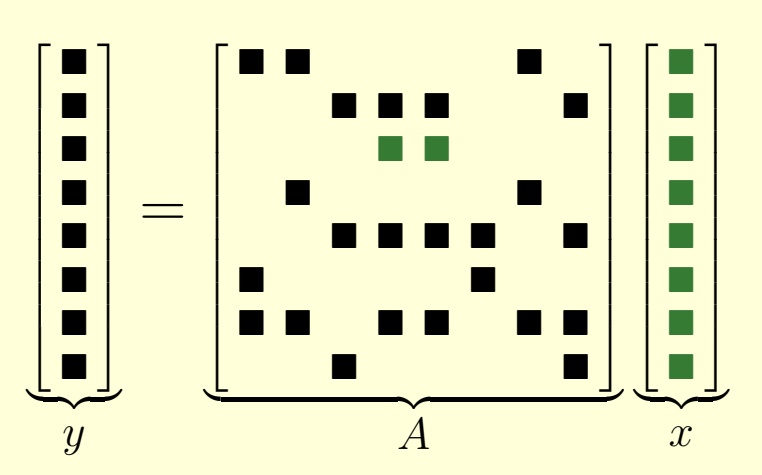
- Quantifies number of x-entries that are accessed with unit stride
- Captures prefetching and cache line reuse



The *block density* $b_d = b_s/n_{zr} \in [0,1]$ is the fraction of row non-zeros in a block.

Importance (with b_s fixed):

- Characterizes reuse factor in the access of vector y
- Exploitation of prefetching and cache memory reuse



The *row density* $r_d = n_{zr}/n \in [0,1]$ is the ratio between entries and row size.

Importance (with b_s and b_d fixed):

- Inversely proportional to problem size
- Expresses probability of finding x-entries in a higher memory level

SpMV classification: parameters vs. performance

Classification algorithm

1. define a reference *training set* composed of sparse matrices with a "uniform" (though sparse) non-zero structure
2. classify training set matrices into 2 groups based on memory requirement: cache vs. DDR
3. for each group, apply a k-means clustering algorithm for time, power, and energy ($k = 2$ in our case)

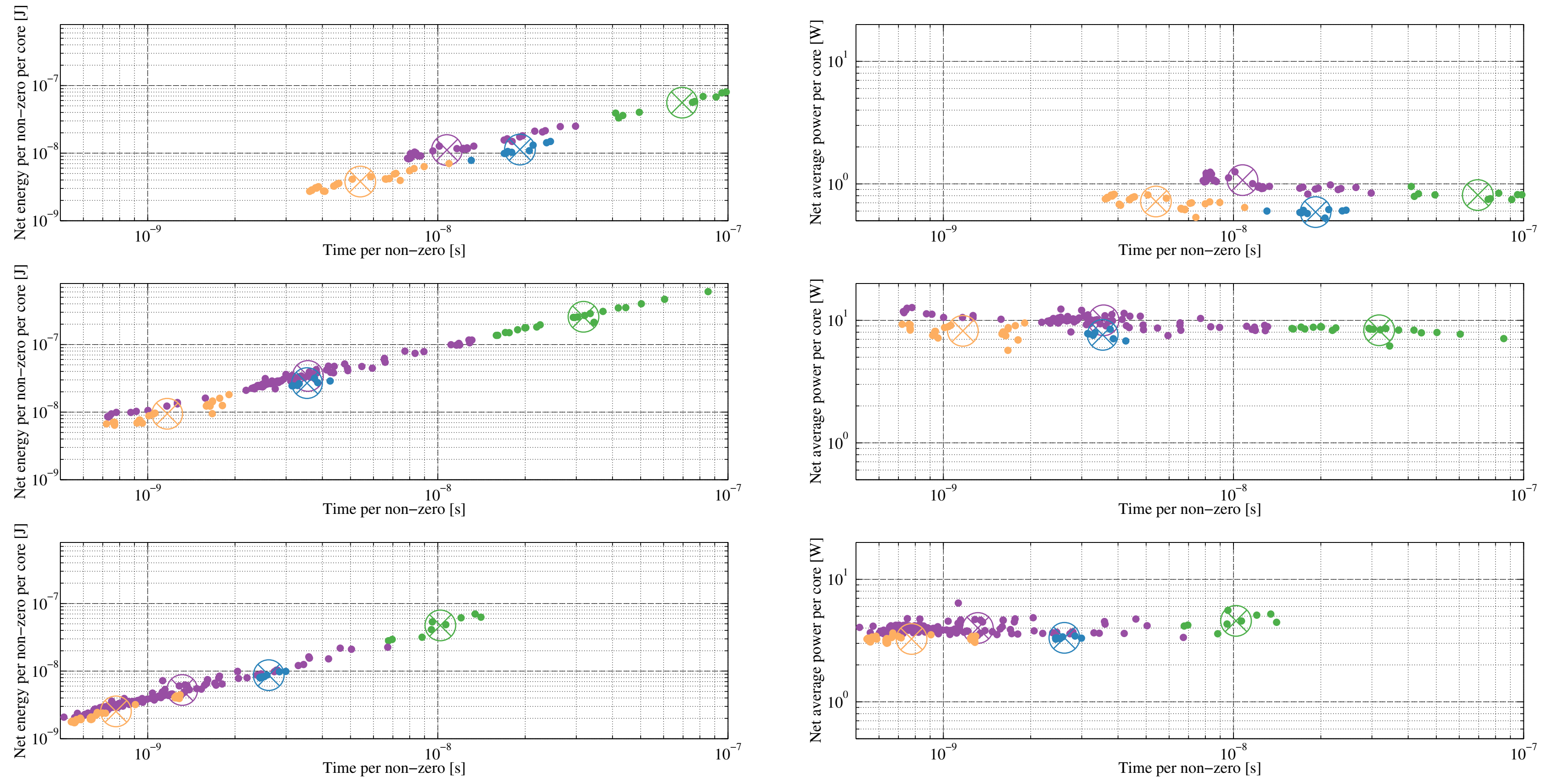


Figure 4: Training set classification on IBM BG/Q (first row, SMT4), IBM POWER7 (second row, SMT4), and IBM POWER8 (third row, SMT8).

- Classes are clustered into precise regions in the 3-D space defined by the (b_s, b_d, r_d) parameterization (see Figure 5)

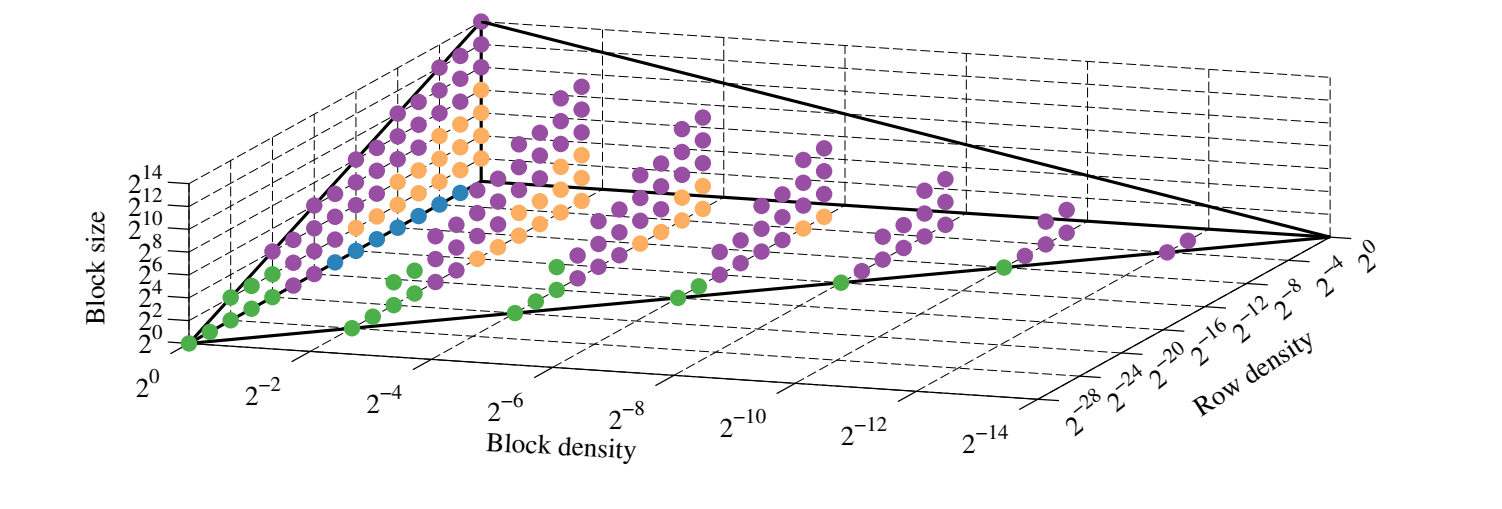


Figure 5: Matrix classification mapped in the 3-D (b_s, b_d, r_d) domain.

- Straight correlation exists between (b_s, b_d, r_d) parameterization and performances (see Table 4)

Table 4: Qualitative performance of each class of matrices.

Class	Color	Memory	Time	Power	Energy
1	●	Cache	Low	Low-Medium	Low
2	●	Cache	Medium	Low	Medium
3	●	DDR	Low-Medium	Medium-High	Low-Medium
4	●	DDR	High	Medium	High

SpMV modeling and validation

Model workflow

1. *(Optional) Row-width filter [W]* – redefine r_d to account for r_w, i.e., the average distance between the first and the last entries in the rows:

$$r_d = \frac{n_{zr}}{n} \quad \longrightarrow \quad r_d = \frac{n_{zr}}{r_w}, \quad \text{with} \quad r_w = \frac{1}{n}\sum_{i=1}^{n}\left[r_w\right]_i$$

2. Inspect the matrix graph to average the coordinates (b_s, b_d, r_d):
   - *Single estimate [S]* – for global averages over the entire matrix graph:

$$b_s = \frac{1}{n_b}\sum_{k=1}^{n_b}\left[b_s\right]_k \qquad b_d = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\left[n_b\right]_i} \qquad r_d = \frac{n_z}{n^2}$$

   - *Block estimate [B]* – for local averages on each j-block of rows:

$$b_s^j = \frac{1}{n_b^j}\sum_{k=1}^{n_b^j}\left[b_s\right]_k \qquad b_d^j = \frac{1}{n^j}\sum_{i=1}^{n^j}\frac{1}{\left[n_b\right]_i} \qquad r_d^j = \frac{n_z^j}{n^j n} \qquad j = 1,2,\ldots,N$$

3. *(Optional) Memory filter [M]* – filter the cloud of point in Figure 5 depending on the known memory requirements (see Figure 6)

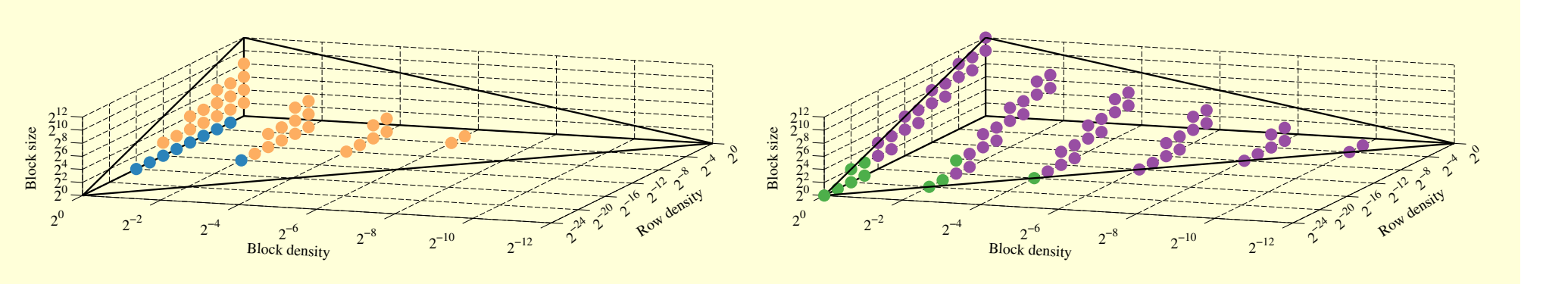


Figure 6: Filtered training set cloud of points: cache (left) vs. DDR (right).

4. Build a 3-D interpolation grid on top of the (filtered) cloud of points
5. Interpolate time, power, and energy at the given coordinates

Table 5: Model percent normalized error for the 50th, 75th, and 95th percentile of the entire UF collection [1] on IBM BG/Q. The error is normalized over the range of admissible measures Δ (minimum and maximum values in Figure 4). Time and energy predictions are very accurate. Power predictions are less accurate: this can be accounted to the limited accuracy of the power measurement system.

	Time $[\Delta = 9.50\cdot10^{-8}\ \text{s}]$						*Net power* $[\Delta = 7.29\cdot10^{-1}\ \text{W}]$						*Net energy* $[\Delta = 7.76\cdot10^{-8}\ \text{W}\cdot\text{s}]$					
Percentile	S	B	S-M	B-M	S-W	B-W	S	B	S-M	B-M	S-W	B-W	S	B	S-M	B-M	S-W	B-W
50th	2.33	2.08	1.50	1.39	1.90	1.02	7.73	8.31	5.76	5.37	9.63	9.27	1.91	1.61	1.26	1.16	1.59	0.96
75th	5.58	4.73	4.21	4.00	3.20	2.53	24.82	23.30	16.75	16.42	32.12	29.96	4.06	3.96	2.95	2.69	4.04	3.65
95th	16.28	16.32	22.20	22.19	6.79	5.49	62.44	59.06	42.55	43.46	68.44	69.57	15.03	15.13	20.10	20.06	7.48	7.48

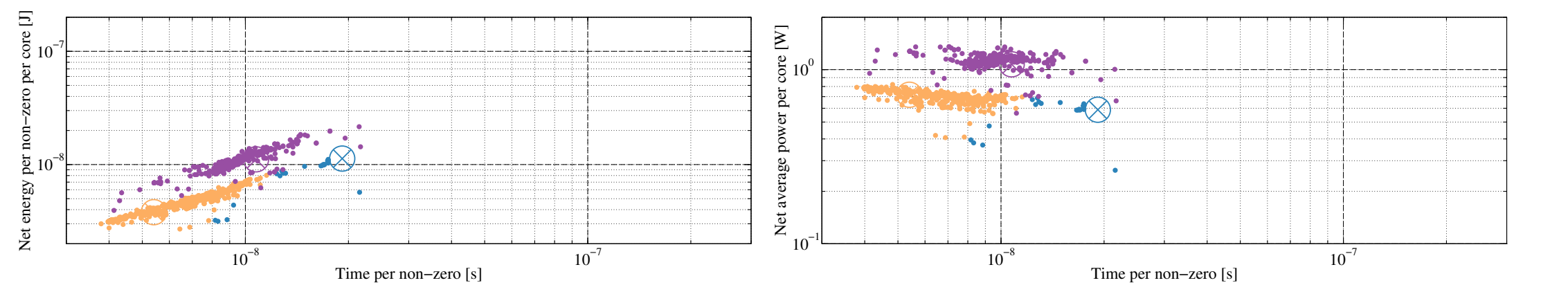


Figure 7: Classification of the entire University of Florida Sparse Matrix Collection [1] on IBM BG/Q.

References

[1] The University of Florida Sparse Matrix Collection, April 2014. `http://www.cise.ufl.edu/research/sparse/matrices`.

[2] K. Asanovic *et al.* The landscape of parallel computing research: a view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Dept. University of California, Berkeley, Dec. 2006.

[3] A. C. I. Malossi, Y. Ineichen, C. Bekas, A. Curioni, and E. S. Quintana-Ortí. Performance and energy-aware characterization of the sparse matrix-vector multiplication on multithreaded architectures. In *Proc. of 43rd ICPP*, Minneapolis, MN, USA, Sep. 2014.

[4] A. C. I. Malossi, Y. Ineichen, C. Bekas, A. Curioni, and E. S. Quintana-Ortí. Systematic derivation of time and power models for linear algebra kernels on multicore architectures. Submitted, 2014.

Acknowledgment

Exa2Green
energy-aware numerics

Project website: `http://exa2green-project.eu`

SEVENTH FRAMEWORK PROGRAMME