Energy-Aware Software for "Greener" Simulations

E. Calore, A. Gabbana, S.F. Schifano and R. Tripiccione

Introduction

Recent high-end processor architectures offer advanced power-monitoring and power-saving features, previously available only on low-power devices [1]. In this work we exploit these features in order to tune energy consumption on a per function/application basis.
We present simple techniques to tune processors clock frequencies (for both CPUs and GPUs) and to measure the effects of these optimizations in terms of the consumed energy and execution time for a given application.

Lattice Boltzmann Simulations

As a typical HPC workload we adopt a Lattice Boltzmann simulation implemented in different languages, optimized for different architectures, such as Intel Haswell CPUs [2] and NVIDIA kepler GPUs [3].

Lattice Boltzmann methods (LB) are widely used in computational fluid dynamics, to describe flows in two and three dimensions. LB methods – discrete in position and momentum spaces – are based on the synthetic dynamics of *populations* sitting at the sites of a discrete lattice.

LB models in n dimensions with p populations are labeled as $DnQp$; we consider a $D2Q37$ model describing the thermo-hydrodynamical evolution of a fluid in two dimensions; this model has been used for large scale simulations of convective turbulence [3, 4, 5].

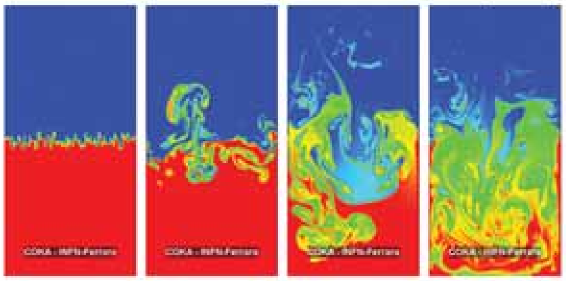


Figure: Four different steps, of a typical simulation, computed by our code.

At each iteration two critical kernel functions are executed:

- **propagate** moves populations across lattice sites involving a large number of sparse memory accesses; it is therefore strongly memory-bound;
- **collide** performs all mathematical steps in order to compute the population values at each lattice site for the next time step; collide is the floating point intensive step of the code.

Changing the CPU clock

Running on an Intel Xeon E5-2630 CPU, we set a specific clock frequency before executing each function of the Lattice Boltzmann simulation, paying a cost of $\approx 10\mu s$ for each frequency change. We concurrently measured energy consumption from Intel RAPL hardware energy counters [6].

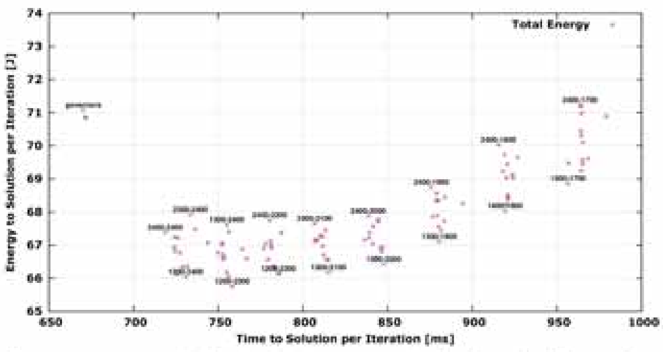


Figure: Average energy consumption for a simulation iteration; one point for each couple of frequency (reported in MHz on the labels, for some points. On the left the one used for propagate, while on the right for collide).

Changing the GPU clock

We did the same on an NVIDIA GPU hosted in a K80 board, but in this case the cost of a frequency change is $\approx 10ms$, thus identifying a single GPU frequency for the whole simulation seems a better choice:

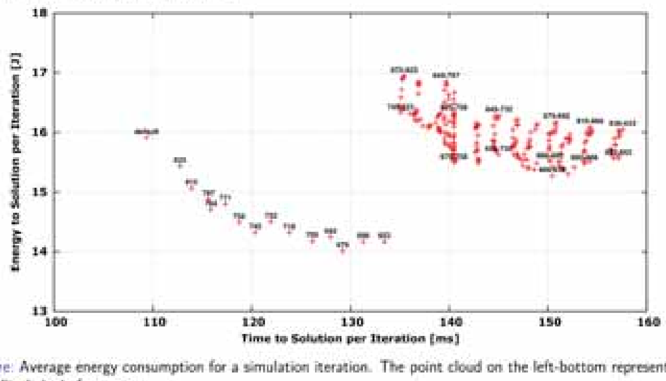


Figure: Average energy consumption for a simulation iteration. The point cloud on the left-bottom represent runs at a fixed single frequency.

Results for single processors

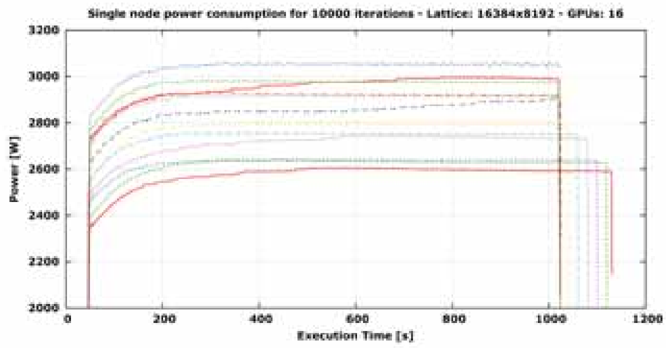
Taking into account, for both CPU and GPU processors, the frequencies that led the best energy efficiency, we estimated the energy saving wrt the performance penalty:

	GPU		CPU	
	E_S saving	T_S cost	E_S saving	T_S cost
propagate	18%	0%	9%	3%
collide	6%	10%	4%	4%
Full code	11%	10%	7%	8%

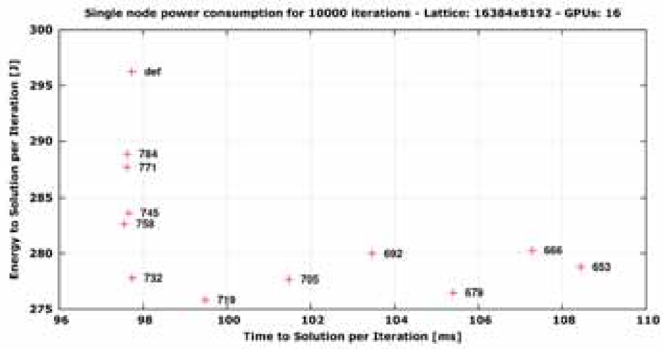
Table: Energy-to-solution (E_S) gains and the corresponding time-to-solution (T_S) costs.

Results for a GPU cluster

We applied the proposed method also on a full production code [3] running on 16 GPUs (8 x NVIDIA K80 Dual GPU boards) hosted on a single node. Power drain of the whole node was measured during code execution, for different GPUs clock frequencies, from the node's power supplies, through IPMI:



Then we computed the average energy in Joule, needed by each iteration, for the different clock frequencies:



At a specific frequency (i.e. 732MHz) $\approx$ 7% of the total consumed energy of the computing node can be saved without impacting performances. Taking into account the Italian average energy cost of 0.17€/kWh, the savings amount to $\approx$ 300€/year for each computing node.

Node No.	32	128	1024
k€/year	9.5	38.1	305

Table: Potential saving in k€/year of electricity bill for clusters of different sizes, not taking into account the savings related to the energy dissipated by the cooling system.

Conclusions

- default frequency governors do not seems to be energy aware;
- per function frequency optimization is not viable yet on GPUs, but it is on CPUs;
- per application frequency optimization can give interesting energy savings with minimal or no impact on performances on both CPU and GPUs;
- in general, for compute bound functions higher clock frequencies are desiderable for both energy efficiency and performances, while for memory bound functions clock frequency can often be reduced to minimize energy consumption minimally impacting on performances;
- the presented method is directly applicable on any other application and interestingly most applications are memory-bound.

References

[1] Enrico Calore, Sebastiano Fabio Schifano, and Raffaele Tripiccione.
Energy-Performance Tradeoffs for HPC Applications on Low Power Processors.
In *Euro-Par 2015: Parallel Processing Workshops*, volume 9523 of Lecture Notes in Computer Science, pages 737–748. Springer Berlin Heidelberg, 2015.

[2] Enrico Calore, Nicola Demo, Sebastiano Fabio Schifano, and Raffaele Tripiccione.
Experience on vectorizing lattice boltzmann kernels for multi- and many-core architectures.
In *PPAM 11th International Conference, Krakow, Poland, September 6-9, 2015*, Lecture Notes in Computer Science, pages 53–62. Springer, 2016.

[3] Enrico Calore, Alessandro Gabbana, Jiri Kraus, Elisa Pellegrini, Sebastiano Fabio Schifano, and Raffaele Tripiccione.
Massively parallel lattice boltzmann codes on large GPU clusters.
Parallel Computing, 58:1–24, 2016.

[4] Luca Biferale, Filippo Mantovani, Mauro Sbragaglia, Andrea Scagliarini, Federico Toschi, and Raffaele Tripiccione.
Second-order closure in stratified turbulence: Simulations and modeling of bulk and entrainment regions.
Physical Review E, 84(1):016305, 2011.

[5] Luca Biferale, Filippo Mantovani, Mauro Sbragaglia, Andrea Scagliarini, Federico Toschi, and Raffaele Tripiccione.
Reactive Rayleigh-Taylor systems: Front propagation and non-stationarity.
EPL, 94(5):54004, 2011.

[6] Enrico Calore, Sebastiano Fabio Schifano, and Raffaele Tripiccione.
https://baltig.infn.it/COKA/PAPI-power-reader