Toward computationally efficient Multi-Subband Monte Carlo Simulations of Nanoscale MOSFETs

Patrik Osgnach, Alberto Revelant, Daniel Lizzit, Pierpaolo Palestri, David Esseni and Luca Selmi
DIEGM, University of Udine, Udine, Italy
Via Delle Scienze 206, 33100 Udine
Email: patrik.osgnach@uniud.it

***Abstract*—We show how intense exploitation of multi-core architectures allowed us to cut by up to an order of magnitude the execution times of a Multi-Subband Monte Carlo (MSMC) simulator. The result brings simulations with the MSMC method out of the strictly academic domain and close to the execution time threshold for effective use in R&D departments of semiconductor research centres and industries.**

I. Introduction

Monte Carlo (MC) techniques to solve the semi-classical Boltzmann Transport Equation (BTE) have been for long time regarded as excessively demanding from a computational point of view and too time consuming for the daily use in the R&D departments of semiconductor industry. Thanks to continuous increase of computing resources at decreasing costs, and to improved algorithms for efficient collection of carrier statistics [1], the use of MC transport simulators is today well accepted for device analysis and design. In fact, MC is a perfectly integrated section of standard TCAD tools [2]. Aggressive scaling of CMOS devices and the introduction of technology boosters, such as high-κ gate stacks, strain and crystal orientation engineering promoted the adoption of *multi-subband* Monte Carlo models (MSMC) where a system of coupled BTEs for the inversion layer subbands is solved [3], [4], [5], [6], [7]. These models are computationally heavier than conventional MC models for the 3D carrier gas. MSMC has demonstrated its ability to enable the understanding of complex nanoscale CMOS device physics [8], but it is still mainly an academic research tool with execution times ranging from hours to tenths of hours per bias point on single core architectures.

In order to bring MSMC to the same level of acceptance that conventional 3D MC has today, a significant reduction of the execution times is mandatory. Code optimization is one way to achieve this goal but, as will be shown in the following, its benefits are often of modest entity and vary greatly from one simulation to another. The limitations of optimization are evident when one considers how modern CPUs are evolving nowadays. Significant CPU performance improvements do not come from a more efficient microarchitecture but from the integration of multiple cores on the same die. Having many cores has a price, however. Top notch CPUs (from Intel [9]) have a maximum TDP (Thermal Design Power) of 130-150W, and an increase in the number of cores corresponds to a decrease of the clock frequency of each core. Thus, unless there are very few processes running on a given CPU, the performances of single-threaded processes are reduced. From these considerations it is clear that, in order to achieve our goal, a massive exploitation of available multi-core architectures must be sought by means of code parallelization.

II. Simulator Overview

MSMC simulators [3], [4], [5] solve the multi-subband BTE by looping through 4 main steps (Fig. 1) until convergence is reached. Firstly the device is partitioned in sections along the transport (x) direction and the 1D Schrödinger equation is solved in each section to obtain the subband energies and wavefunctions (step "1", SE). The scattering rates used in the MC transport are computed in each section (step "2", SC) according to the subbands energies and wavefunctions [10]. The subband profile along x is then constructed by connecting the subbands of each section and it is used by the Monte Carlo transport algorithm (step "3", MC) to move the particles inside the device. Transport is divided in time-steps with duration of about 1fs. Statistics collection occurs periodically (e.g. every 10 time steps). After the MC transport step, a new potential profile is computed by solving the 2D Poisson equation (step "4", PE). Since the Monte Carlo method relies on a statistical sampling algorithm, its performances and accuracy depend on how the carrier statistics are collected and on the quality of the random numbers generator, as we show in section IV.

III. Optimization

The first step of any optimization task is to find the regions where most time is spent during the program's execution, the so called hot spots, and estimate their relative contribution to the duration of the job. This is typically done using a profiler. To this purpose we used Intel Vtune Amplifier XE 2013 [11] and the results are shown in Fig. 2. The simulation time is dominated by the SC step, followed by the MC step. The other steps add a negligible computational burden. This technique identified many sections of the original MSMC code where improvements were possible. Among these improvements, two are are worthy of notice.

The first regards one of the data structures used during the MC step (step 3 in Fig.1). In order to take properly into account the Pauli's exclusion principle we need to keep track of the particles state, which results in the occupation function in the $\vec{k}$ space. For each particle, the occupation is described by the device section ("x") where the particle is located, its valley (conduction band minimum), its subband (eigenvalues of the Schrödinger's equation) and its wave-vector in the transport plane ($\vec{k}$). A five-level tree is thus required to record the

978-1-4673-5736-4/13/$31.00 ©2013 IEEE

occupation of the electron states (Fig. 3a). Trees are very sparse data structures and the sparseness enforces improper memory access patterns. These patterns have a profound influence on the performance of an application. CPUs use the cache memory to reduce the costs of accessing the main memory, so the data structures must be designed in order to exploit the time and space locality principles [12]. In order to obtain this, we converted the tree into an array with a fixed size record-like structure (Fig. 3b), thus achieving a more cache-friendly processing. By construction, each section has the same number of valleys but each valley can have a different number of subbands. The discretization of the wave-vector requires the same amount of elements for each subband. If we assume that each valley has a number of subbands equal to the maximum among all the valleys, each element can be located by performing very simple math. The same idea applies to other branched structures, such as the ones containing the scattering rates. There is, however, a price to pay. The linearization induces some memory waste (depending on specific simulation parameters), waste partly balanced by the removal of the internal nodes of the trees, which accounted for about 1% of the tree memory occupation.

The second optimization regards both the SC and MC steps (steps 2 and 3 in Fig. 1). We are often interested in finding the total scattering rate from state k to a final state k' and the corresponding wave vector variation ($\vec{q}$) that conserves the energy. For every such $\vec{q}$ we need to compute a matrix element $M(\vec{q})$. In order to compute the scattering rates in an acceptable amount of time, the range of all possible $\vec{q}$ values is chosen in advance and is discretized. A matrix element $M(\vec{q}_i)$ is computed for every such discretized $\vec{q}_i$. During the MC step it is very likely that we need the matrix element corresponding to a value of $\vec{q}$ which was not in the discretized set of $\vec{q}_i$. Therefore we must interpolate between two known $M(\vec{q}_a)$ and $M(\vec{q}_b)$ such that $\vec{q}_a \leq \vec{q} \leq \vec{q}_b$. The original code performed a binary search on the list of $\vec{q}_i$, which is the default choice if the list contents are unknown but ordered. We found that a distribution of the $\vec{q}_i$ that follows a geometric progression is accurate enough and allows us to obtain $\vec{q}_a$ and $\vec{q}_b$ (and the corresponding $M(\vec{q}_a)$ and $M(\vec{q}_b)$ analytically and without a time consuming binary search.

IV. Parallelization

To start with we note that the SE and the SC steps can be easily executed concurrently section-by-section. Parallelization of the MC step instead requires more work and due care.

The MC step implements an ensemble Monte Carlo procedure: the motion of a set of particles is simulated for a number (N) of time steps (Δt) [10]. The two main choices to make are: how to divide the ensemble of particles over the various threads? How to synchronize the motion in the different threads? About the first question, performance scaling with the number of threads can be impaired if the number of particles processed by one thread is too different from the number of particles processed by others. Particles assignment to the threads based on the section where the particle belongs (similar to what is done in [13], [14]) minimizes the amount of data structures accessed by each thread, but requires a significant overhead to trace the particles exiting the domain of one thread to enter the domain of another thread. We thus decided to evenly distribute the particles of each section to all threads (similar to what is done in [15]). This criterion applies also to the particles injected at the contacts (see [16] for the description of how ohmic contacts are implemented in the MSMC). This approach keeps the number of particles processed by each thread roughly the same.

Since we are interested in steady-state solutions, we allow the threads to "drift apart", meaning that, at a given time, the motion of particles in a chunk may have been computed over a longer time with respect to the particles of other chunks. However, when enforcing the Pauli's exclusion principle, we need to know the occupation function f to reject scattering events [17]. To reduce the synchronization points, each thread builds its own estimate of f but, when computing the state-after-scattering, it computes the (particle weighted) average of the f functions of all threads (including its own), for improved accuracy and reduced statistical noise.

Before the statistics collection phase, all threads are synchronized (by using an explicit barrier [18]). This in done in order to avoid mixing together information from particles processed for a too different amount of times when computing f. Fig. 4 shows that increasing the number of time steps (Δt) between two synchronization points affects the efficiency of the parallelization process, that is however negligible above a given number of steps. On the other hand, since we are simulating a steady-state process, the error associated to poor synchronization is essentially negligible. Notice that in the parallelization of the MC motion, a reentrant [19, sect. 12.3.8] random number generator must be used, otherwise much time is spent in serializing the accesses to the global state of a traditional random number generator. Implementing parallel steps required a careful inspection and rewriting of the code to remove any race condition. To parallelize the steps we have used OpenMP API [18], thus implementing a shared memory parallelism. The work done so far allows for a fairly simple conversion to distributed memory parallelism (using MPI), should it come useful in the future.

V. Methodology and Benchmarks

To benchmark the improved MSMC simulator we measured the execution times of the four steps in Fig. 1 using a profiler [11]. Simulations of three template Si and SiGe MOSFETs from [20], with the parameters given in Table I were analyzed. For the sake of a fair comparison, we set the appropriate number of iterations of the loop in Fig. 1 to reach a given relative error computed using equation 18 from [1]: at the end of each iteration (except the first $\mathrm{N_{tran}}$ ones, to discard the initial transient phase) we calculate the channel current I_D by averaging the current over the sections in the channel region. Then we compute $\bar{I}_D$ as the average of I_D over all the previous iterations (except the first $\mathrm{N_{tran}}$) and $\sigma_{\bar{I}_D}$ as its unbiased standard deviation. The simulation is stopped when $r_{err} = \sigma_{\bar{I}_D}/\bar{I}_D$ falls below a chosen threshold. All devices have been divided in 102 sections. $\mathrm{N_{tran}} = 10$ was deemed sufficient to avoid propagating errors from the initial transient. The phonon and surface roughness scattering mechanisms were active in the simulations. For device #3, alloy scattering was also considered. All benchmarks were performed on a workstation equipped with two Xeon X5690 and 192GiB of DDR3 main memory (1GiB=2^{30}B).

VI. Results

As a first sanity check, we verified that the same currents were obtained with the original and the improved codes. Fig. 5 shows the drain currents vs the gate voltage for the three devices. The I_D computed by the improved code has a maximum discrepancy of 2.2% with respect to the original code. Fig. 6 shows that increasing the number of iterations lowers the relative error and that the process is slower in the sub-threshold region because no statistical enhancement techniques are used.

Many factors determine the execution time of the steps in Fig. 1. The most relevant is the number of sections, since the duration of the SE and SC steps increases proportionally. The SC step duration also depends on the number of activated scattering mechanisms. Roughly 50% of the SC time is spent computing the dielectric function [21]. The duration of the MC step is proportional to the number of particles and time steps (typically between 1000 and 1500). The time spent on solving PE depends on the size of the mesh. Table II compares the absolute execution times of the original and the improved code. We have chosen a relative error r_{err} of 4% for V_{GS} = 0.0V and 1% for the other cases, which is more stringent than the ITRS roadmap specifications for accuracy in modeling [22]. Device 3 requires longer times because of the additional alloy scattering mechanism, but it is the one which benefits the most from our work, being now 19 times faster to simulate. For a given device and a given number of threads [19, sect. 2.2], the relative amount of time spent in each step is roughly constant because the executed code is the same at each iteration. For this reason, in Fig. 2 we show only the time distribution for V_{GS} = 0.5V. Parallel execution should keep constant the percentages of Fig.2, when the parallel steps scale equally well, while the optimization should changes the relative contribution.

Finally, Fig. 7 shows how the parallel code scales with the number of threads. We compare the execution times taking the original code as a reference. The scaling is not ideal and tends to saturate when the number of threads is increased. This is because of both the implicit synchronization points (at the end of the parallel regions [18] in the SE, SC and MC steps) and the explicit ones (added to the Monte Carlo code when statistics are collected). A strategy for particle removal and injection at the contacts that keeps their number uniform over the threads mitigates the detrimental effects of these explicit synchronization points, as explained in section IV. For device #3 in Table I, the code optimizations alone (seen when using only one thread) account for a 66% execution time reduction, but this figure reduces to about 10% for other devices, thus confirming that code optimization is not enough to achieve the desired execution time savings. After all the work done, what remains strictly serial is the PE solver and all the initialization, management and support code. This code accounts for about 1.5% of the total execution time. By applying the well known Amdahl's law [23] we obtain a maximum theoretical average scaling of 7.23 when using 8 threads (see Fig. 7), which is higher than the actual average scaling of 6.4. The difference is due to the synchronization points in both the SC and MC steps. Regarding MC, we have synchronized every 10 time-steps in all figures, whereas Fig.4 shows that increasing the time steps between synchronization may help.

VII. Conclusion

The improved version of the MSMC simulator delivers in few hours (or less) results which required about one day of computation time. Speed-up factors are close to those predicted by Amdahl's law. 8-threaded simulations are almost 6.5 times faster (on average) with respect to the single-thread simulation, which in turn can be up to an additional factor of 3 times faster due to code optimization. The differences between the theoretical and measured speed-ups are due to the unavoidable synchronization points. Regarding the SC and SE blocks, these points are placed at the end of the parallel regions in order to prevent the subsequent blocks from using partial data. Regarding the MC block, the synchronization points are needed to prevent the threads from drifting too much apart. These results are very promising in the perspective of enabling MSMC simulations in R&D departments of research centers and semiconductor industries.

Acknowledgements

This work was partially funded by the EU FP7 project STEEPER via the IUNET consortium and by the project "Futuro in Ricerca" (RBFR10XQZ8).

References

[1] C. Jungemann et al. In *Proceedings of SISPAD*, p. 209–212, 1997.

[2] http://www.synopsys.com/Tools/TCAD/CapsuleModule/news_dec04.pdf p. 7.

[3] L. Lucci et al. *IEEE TED*, p. 1156 –1164, 2007.

[4] M. De Michielis et al. In *IEEE TED*, p. 2081–2091, 2009.

[5] M. V. Fischetti et al. *Phys. Rev. B*, p. 2244–2274, 1993.

[6] F. Gamiz et al. *IEEE TED*, p. 1122–1126, 1998.

[7] J. Saint-Martin et al. *Semiconductor Science and Technology*, p. L29, 2006.

[8] F. Conzatti et al. *IEEE TED*, p. 1583–1593, 2011.

[9] http://ark.intel.com/.

[10] D. Esseni, P. Palestri, and L. Selmi. *Nanoscale MOS Transistors*. Cambridge University Press, 2011.

[11] http://software.intel.com/en-us/intel-vtune-amplifier-xe.

[12] P. J. Denning. *ACM Communications*, p. 19–24, July 2005.

[13] A. Kepkep et al. In *Proceedings of IWCE*, p. 21–22, 2000.

[14] Wei Zhang et al. In *Proceedings of IWCE*, p. 1–4, 2009.

[15] A. Hiroki et al. In *Proceedings of VPAD*, p. 18–19, 1993.

[16] P. Palestri et al. *Semiconductor Science and Technology*, 25(5), 2010.

[17] P. Lugli et al. *IEEE TED*, p. 2431–2437, 1985.

[18] B. Chapman et al. *Using OpenMP*. The MIT Press, 2007.

[19] A. Tanenbaum. *Modern Operating Systems 3rd edition*. Pearson, 2007.

[20] D. Lizzit et al. *IEEE TED*, p. 1884–1891, 2013.

[21] P. Toniutti et al. *IEEE TED*, p. 3074–3083, 2010.

[22] http://www.itrs.net/Links/2012ITRS/2012Tables/Modeling_2012Tables.xlsx.

[23] G. Amdahl. In *Proceedings of AFIPS*, p. 483–485, 1967.

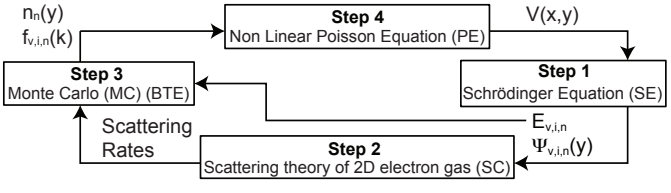

Fig. 1. Flowchart of a typical MSMC simulator [3]. Transport is along x direction, quantization is along y direction.

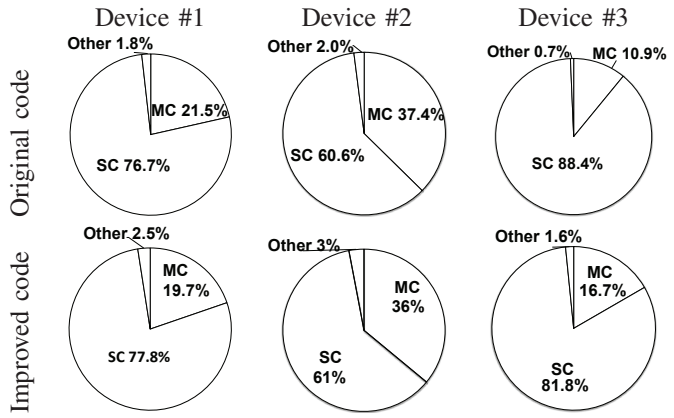

Fig. 2. Execution time distribution measured with the original code for the devices in Table I. V_{DS} = 0.5V. The percentages do not depend on the applied voltages or on the number of iterations.

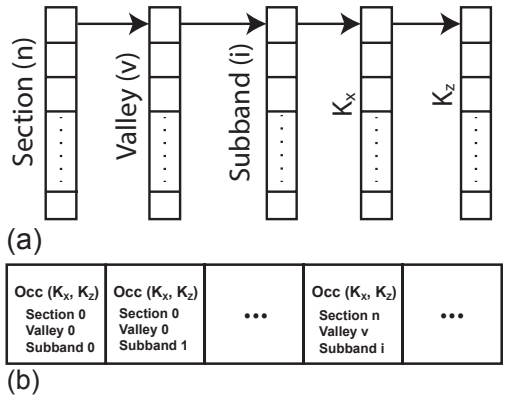

Fig. 3. Linearization of the branched structure of the electron states occupation. (a) represents the original tree-shaped data structure. The square matrices in (b) (one for each subband) represent the occupation in the (K_X, K_Z) plane.

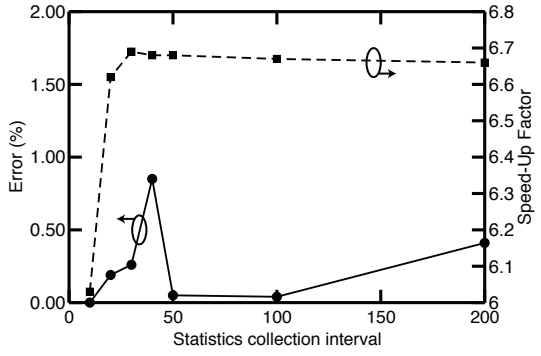

Fig. 4. Speed up using 8 threads (with respect to single-threaded simulation) and simulation error vs synchronization interval for device #3, V_{GS} = 0.5V. The case with synchronization interval equal to 10 time steps is taken as reference to compute the error.

TABLE I. MAIN PARAMETERS DESCRIBING THE UTB-SOI DEVICES CONSIDERED IN THIS STUDY. STRAIN IN DEVICES #2 AND #3 IS INTRODUCED BY A $Si_{0.25}Ge_{0.75}$ SUBSTRATE.

Device	T_{semi}	EOT	L_{ch}	$L_{S/D}$	Channel material
#1	7nm	0.7nm	14nm	20nm	Relaxed Si
#2	7nm	0.7nm	14nm	20nm	Strained Si
#3	7nm	0.7nm	14nm	20nm	Strained $Si_{0.5}Ge_{0.5}$

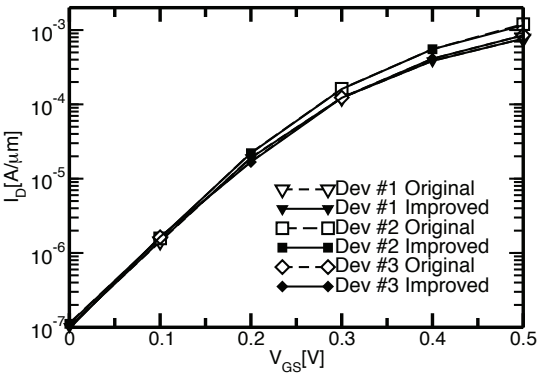

Fig. 5. Drain current for the three devices of Table I vs gate voltage. V_{DS} = 0.5V.

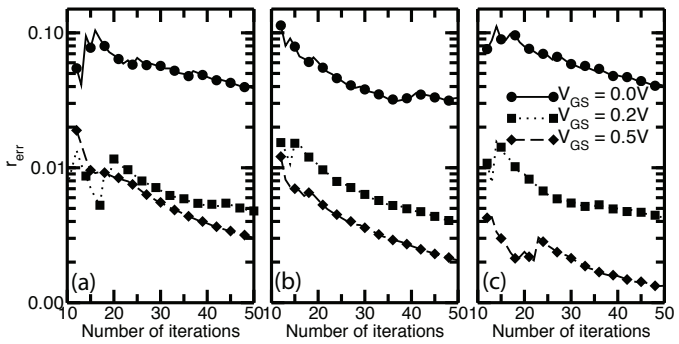

Fig. 6. Relative error vs iterations number for device #1 (a), #2 (b) and #3 (c) in Table I. V_{DS} = 0.5V. See [20] for more details about the device structure and the scattering rates.

TABLE II. EXECUTION TIME FOR THE DEVICES OF TABLE I.

Dev.	V_{GS}	Iterations required	Original exec. time (s)	Improved exec. time(s) (8 threads)	Speed-Up factor
1	0.0V	45	54679	8033	6.8
	0.2V	22	27411	3907	7.0
	0.5V	15	18651	2626	7.1
2	0.0V	27	25346	3603	7.0
	0.2V	20	19002	2656	7.1
	0.5V	13	12244	1743	7.0
3	0.0V	50	240746	12740	18.9
	0.2V	18	88734	4539	19.5
	0.5V	12	59956	3179	18.9

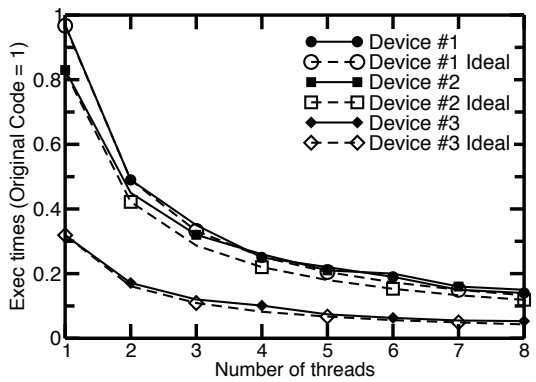

Fig. 7. Execution times reduction versus the number of threads. Ideal curves refer to the scaling expected by applying the Amdahl's law. V_{GS} = 0.5V, V_{DS} = 0.5V.