A GPU Based Parallel Computing Mechanism Of an Etching Profile Evolution Model

Fan Wu, Yixu Song, Nan Wu, Hongjun Yang, Xiaomin Sun

State Key Laboratory on Intelligent Technology and Systems
Department of Computer Science and Technology
Tsinghua Universit
tChengfu Road, Tsinghua niversity
Beijing, 100084, China
lightning2-u.an@126.com

Abstract: Since the simulations for etching and deposition are increasingly time consuming, we explore the parallel computing mechanism of a typical profile evolution model on GPU platform in order to accelerate the process. It turns out the traditional parallelism will lead to either data unsafety or low efficiency, because of GPU's special architecture and a lack of safe protocols to protect a memory address from being simultaneously written by multi-threads. To overcome the dilemma, we propose an advanced parallel computing mechanism suitable for GPU platform by dividing each Thread's work into several parts. It is theoretically and experimentally proven that the multi-Threads writing the same address can be totally avoided. Experiment results show that the advanced parallel model is much faster than the traditional serial one.

1 Introduction

More and more efforts are made to explore the mechanism and principles of the plasma etching process during the manufacturing of microelectronic devices, and computer simulation is becoming an essential way to study, and even to predict this complicated process. Different models are used to simulate and to explore the process with some common species of gas and substrate. Since the Continuous Cellular Automata (CCA) was introduced[ZC00], many others extended and improved the model in different aspects by expanding the traditional 2D cellular model into a 3D model and introduced a time compensation algorithm [Ka99], combining complicated and specific physical and chemical principles into a 3D profile evolution model[GSV10], exploring more details about the electric field in the plasma etching process[MAC08]. To reduce the simulation time, we explore the potential mechanism of parallel computing for the traditional 2D profile evolution model on GPU (Graphic Processing Unit) by tracking movements of a swarm of particles simultaneously. However, the traditional way to translate a serial program into a parallel one fails on this model because of some special architectures and so called "warp" mechanism of GPU, which cannot provide a highly efficient way to protect a memory address being written by multi-Threads. So we propose an advanced

parallel computing mechanism that disintegrates the incidence-sputtering process into several subtasks to prevent multi-Threads from writing the same memory address and keep efficiency. The next Section briefly introduces the traditional profile evolution model; Section 3 discusses in detail about the problem with a common design of parallel computing using GPU, and then proposes an advanced mechanism; experiments and results of the advanced parallel model are reported in Section 4 with some analyses.

2 Traditional Profile Evolution Model

In this model, the substrate is assumed to be silicon just for convenience of latter introduction, and the particles are assumed to be the element inert to silicon to ensure only physical sputtering will occur so that more attention can be paid on the parallel computing mechanism. In order to represent the profile, the whole model is divided into lattice. Particles with energy and incidence angles from given distributions appear at the top-most of the lattice. Once a cell is hit by a particle, its value will decrease according to the energy and incidence angle of the particle. Once the value descends to 0, it will become a vacuity and some internal cells next to it will be exposed to other particles. A result of this traditional model shows that the middle bottom is higher than both side bottoms. That is because after being reflected from the side walls, particles are more likely to fall onto the side bottom of the silicon substrate than the center, resulting in the sputtering of more silicon atoms.

3 Parallel Mechanism for Profile Evolution and Detailed Discussion

A Graphic Processing Unit (GPU) is originally a specialized integrated circuit designed to rapidly build the images in a frame buffer intended to display on a screen in parallel. Now it is generalized to qualify some vastly repeating works and outperforms CPU. Owning independent Registers and Local Memory, every Thread can be seen as a little processor similar to CPU. The Threads from one Block can communicate and transfer data with each other through Shared Memory, to which a processor can access as fast as to Registers, but the Threads can only communicate with each other through Global Memory if they are from different Blocks. Each Thread runs a copy of the codes and all Threads run almost the same codes while they can run slightly different parts of the codes based on their unique orders. So GPU has great advantage in some highly repeated and little dependent works.

The first question to use parallel computing to accelerate a program is which parts of the program can separately run in parallel. According to the traditional serial model, an ordinary design is making different particles fall simultaneously, while there is always one particle in the serial model. As a consequence, it is possible that two particles simulated by two different Threads hit on the same cell nearly at the same time, which may cause data unsafety. In the serial model, particles fall one by one and the next one cannot fall until the effect of the previous one is finished. Conversely, in the multi-Threads design, two particles can affect a surface cell simultaneously, calculate the

results and write them to the same address regardless of other particles' effectiveness. Since there is no safe mechanism other than Atomic Operators for integers in GPU to protect the data, the eventual result simultaneously written by two Threads is uncertain and could be any one of the results from the two Threads. For instance, when a particle hit on a silicon cell, which should have become a vacuity cell after it has been sputtered by a previous particle but the sputtering process hasn't finished yet in that Thread, it could cause an invalid hit and sputtering or even some kind of unexpected fatal errors. In short, it is a typical Readers-Writers Problem. Even though it can be solved by using Spin-Lock or Monitor, the solutions cannot perform as well on GPU as on CPU because of GPU's special hardware architecture and controlling mechanism. In GPU, there is a concept of "warp", which is the most basic unit for instruction dispatching and controls 16 or 32 Threads' IPs (Instruction Pointer) to make sure these Threads are always synchronized. Whenever a Thread goes into a branch of the codes while others from the same warp don't, the others will wait for it to finish the branch and then run the same next instruction synchronously. It is this special mechanism that will cause deadlock in Spin-Lock or Monitor solution.

There is always only one Thread that can get the lock each time and if one Thread fails, it will keep trying and be blocked. Under GPU architecture, if two Threads from the same warp are getting the same lock, or if two particles simulated by the same warp are sputtering the same cell, only one of them can get authorized to access and write the effect to the cell while the other one has to wait for the lock to be released. On the other hand, even though one Thread can get the lock, it will still need to wait for the other one before it runs the critical codes since they belong to the same warp. As a consequence, when the one with the lock has no chance to release the lock and the other is waiting for the release of it, a deadlock appears. There is only one way to prevent this kind of deadlock: serializing the Threads from the same warp on the critical codes and preventing them from getting the same lock at the same time, which will inevitably lose a lot of efficiency. According to tests, the cost of the serializing Threads solution is unacceptably high, so an advanced design of parallel computing mechanism is required.

Whenever two Threads are about to write the same address at the same time, a lock mechanism must be added to protect the data, then that leads to Threads serialization and inevitably lose a lot of efficiency. So the dilemma can only be solved at the root, which is to say, to prevent any two Threads from writing the same address.

An applicable advanced mechanism is to separate the work of each Thread to several phases, where each phase ends whenever more than two Threads have the possibility of writing the same address. Before every phase ends, each Thread must store the data into a cache instead of writing it directly to the final address. Then at the beginning of the next phase each Thread will collect the data that is about to be written to a particular address from the cache, then it will calculate and write the final result to this address. For instance, in the advanced parallel computing mechanism, first phase of each Thread is to simulate the movement of a particle and to write the sputtering yield to cache after it hits on a silicon cell, and at the second phase each Thread will collect all the sputtering yields on a same cell and calculate the remains of the cell. In other words, each Thread simulates a particle in the first phase but simulates a cell in the second phase. In this

advanced mechanism, writing the same address by different Threads can be totally avoided by using cache, and only a few simple Atomic Operators can ensure that different Threads won't have access to the same address in cache. So the advanced mechanism for parallel computing can guarantee data safety. The process can be described as Figure 1. From the figure we can see the particles are coming swarm by swarm instead of one by one in traditional model.

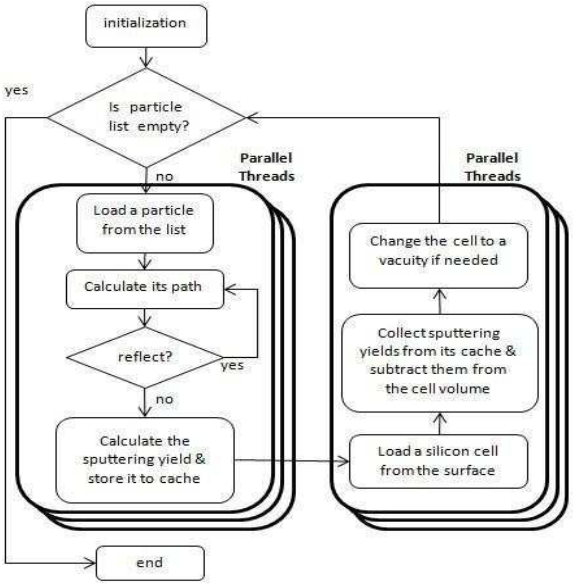

Figure 1 Flow chart of advanced parallel computing mechanism

4. Experiments and Results

We implement the traditional serial profile evolution model running on the Host with 8 Intel i7 CPU and 6GB RAM, and the advanced parallel computing model running on NVIDIA GeForce GTX 570 with 15 Blocks, each of which can run a maximum of 1024 Threads. In the first experiment, these two models run with the same particle list of 2M particles and the running times are recorded. The running time of advanced model involving 2~12 Blocks is shown in Figure 1. It is easy to understand that as the Block number increases, the number of particles that fall at the same time will increase so that the simulated process can be done within less time. And the running time is almost proportional to one divided by Block number. More specifically, no matter how many particles are in the last swarm of particles, the program should repeat one more time even though a lot of Threads are in leisure. So if PN and TN respectively indicate the total number of particles in the list and the total number of Threads in all Blocks, the running time should be proportional to [PN/TN]. When TN$\geqslant$PN, the running time will stop descending regardless of TN. And the fact is only when can the increase of the value of TN distinctively save the running time. The dashed line above all the data points in Figure 1 indicates the cost time of serial model running on CPU with the same particle

list. The result says that the advanced model can be approximately 6 times faster than the original model under the same input conditions.

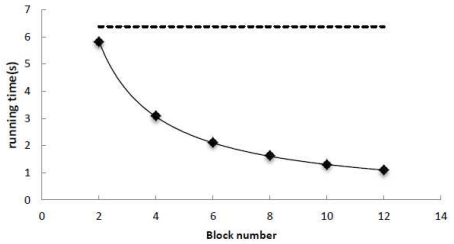

Figure 1 Running time of advanced model with increasing Block number, dashed line represents the running time of original model under the same conditions

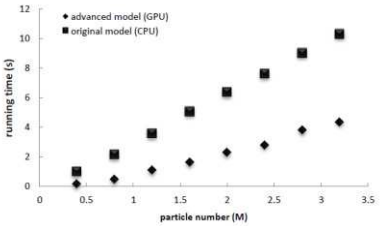

Figure 2 Increasing running time of two models with different particle numbers

According to the block diagrams of both original model and advanced model, the running times of these two models are expected to increase linearly as the particle number increases. Then it is confirmed by the test result shown in Figure 3. Since the time complexity of the original model is O(PN), there is not much prospective improvement in this aspect. But because of the increasing ratio being less than the serial model, the advanced model with only 6 Blocks can save at least half of the running time and the saved time will become more meaningful under a really large input particle list.

Another important result shown in Figure 3 indicates the difference between the final silicon profile in traditional serial model and that in advanced parallel model.Graph (a) in Figure 3 is the result from the serial model while (b) comes from the advanced parallel model with only one Block and under the same conditions with the serial model. In general, the etching depth of (b) is slightly less than that of (a). Before the cause can be analyzed, the sputtering yield compensation mechanism applied to the serial model must be introduced first. Because the silicon substrate is divided into lattice, and only one cell in the lattice can be sputtered each time. So it is unavoidable that some particles' sputtering yield is bigger than the remained volume of the cells they hit on, which will cause a little lost sputtering yield without the sputtering yield compensation mechanism. More specifically, after each sputtering that creates a yield exceeding the left volume of the cell, the remaining yield will be separated evenly to the adjacent silicon cell to make sure there is no lost sputtering yield. However, the mechanism applied in the traditional serial model cannot be inherited to the advanced parallel model because it naturally contains the possibility of writing the same address in different Threads, which is demonstrated that must be avoided in GPU programs. So the compensation has to be abandoned in the parallel model, which can cause some lost of sputtering yield that builds up the difference between (a) and (b). The more the total sputtering yield on a silicon cell within a swarm of particles is, the more yield will be lost when the remaining volume of the cell is nearly zero, and then the less deep the result will be. So we reckon that the advanced model, which runs on more Blocks or more Threads, may generate results that are less deep and probably less accurate.

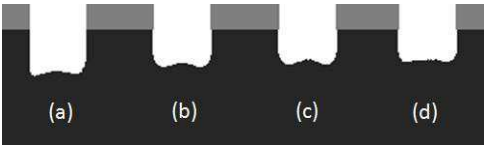

Figure 3 Results of traditional and advanced model (a) original model; (b) advanced model with 1 Block (c) advanced model with 8 Blocks; (d) advanced model with 12 Blocks

Graph (c) & (d) from Figure 3 provide an evidence supporting our theory while (c) is the result of advanced model running on 8 Blocks and (d) on 12 Blocks. This is the major reason why the Threads used by advanced model must be limited even though more Threads lead to more efficiency. However, it can be demonstrated that there is an instinct relationship between the lost yield and Threads number TN. So the compensation mechanism can somehow be interpreted into the function of sputtering yield $Y(E,\theta)$ to make it a GPU specialized function $Y(E,\theta,TN) = c(\sqrt{E} - \sqrt{E_{th}}) \cdot f(\theta) \cdot (1 + P(TN))$

where P(TN) represents a new compensation mechanism only depending on used Threads number TN in the GPU model. We are now exploring the expression of P(TN).

5 Conclusion

After thoroughly exploring the architecture and control mechanism of GPU and revising the whole process of the traditional serial model, we proposed an advanced parallel computing mechanism to overcome the dilemma caused by the "warp" mechanism of GPU. And the test results demonstrate that the advanced model outperforms the traditional one and has the potential to save more running time. It's quite important in today's increasingly complex models. In conclusion, GPU doesn't provide a safe protocol to protect the memory from being written by multi-Threads and the best solution is to totally prevent this kind of circumstance when designing the whole program. Exploring the sputtering yield compensation function P(TN) so that the results can be more consistent and convincing and complicating the model to test whether it can jeopardize the present parallel computing mechanism are interesting topics in the future works.

References

[ZC00] Zhu A.; Liu C.,: Micromachining Process Simulation Using a Continuous Cellular Automata Method, Journal of Microelectromechanical Sytems. 2000; 9(2), 252-261.

[Ka99] Karafyllidis I.: A three-dimensional photoresist etching simulator for TCAD, Modelling Simulation in Material Scicence and Engineering, 1999; 7, 157-168.

[GSV10] Guo W., Sawin H.H: Etching of Si O2 in C4 F8 /Ar plasmas. II. Simulation of surface roughening and local polymerization, Journal of Vacuum Science and Technology A: Vacuum, Surfaces and Films, 2010; 28(2): 259-270.

[MAC08] Madziwa-Nussinov T.G., Arnush D. and Chen F.F.: Ion orbits in plasma etching of semiconductors, Physics of Plasmas, 2008; 15, 013503 (2008); doi: 10.1063/1.2819681