An Improved Parallel Implementation of 3-D DRIE Simulation on Multi-core Processors*

Zhang Fan, Wang Gang, Liu Xiaoguang, Sun Guangyi, Zhao Xin, Liu Jing, Lu Guizhang
College of Information Technical Science, Nankai University, 300071, Tianjin, China
zhangfan555@gmail.com

Abstract

Deep reactive ion etching (DRIE) technique is a new and powerful tool in Micro-Electro-Mechanical Systems (MEMS) fabrication. A 3-D DRIE simulation can help researcher understand the time-evolution of Bosch process used in DRIE. Due to the high complexity of the algorithm used in the simulation, it is necessary to develop an algorithm that can speedup the simulation. This paper presents a parallel implementation of the 3-D DRIE simulation based on multi-core processor. The algorithm is based on data partition. We examine four different data partition strategies and find two-dimensional block-cyclic distribution can obtain perfect load balance. The experimental results show that the parallel algorithm obtains a substantial speedup over the serial algorithm on an Intel quad-core computer.

1. Introduction

In Micro-Electro-Mechanical Systems (MEMS) fabrication, Deep Reactive Ion Etching (DRIE) is a new and powerful tool for the etching of very deep trenches (up to 500 μm) with nearly vertical sidewalls. The most popular silicon DRIE technique is Bosch process patented by Robert Bosch GmbH [1], of Stuttgart, Germany, in which etch and polymerization cycles alternate in an ICP-RIE system.

Since the procedure of Bosch process is more complex than other etchings', an accurate and fast simulation is necessary to help researchers understand the time-evolution of the topography.

Sun Guangyi et al. [2] has proposed a new approach for 3-D simulation of Bosch process with arbitrary 2-D mask shape. Its surface evolution is based on the morphological operations and the visualization is based on volume rendering. The etching and polymerization rate distribution is obtained from macroscopic point advancement models. Based on this method, it is possible to simulate the alternation of etching and polymerization.

Due to the high complexity of the algorithm ($O(n^6)$) proposed by Sun Guangyi, it is necessary to develop a parallel algorithm which can speedup the simulation using the novel multi core technology. We use four different mapping techniques to balance the load among processors.

In the next section, we describe the principle of the DRIE and introduce the serial algorithm used in 3-D DRIE simulation. In section 3 we give the design and analysis of the parallel algorithm. Finally, in section 4, the experiment result is presented.

2. Algorithm of 3-D DRIE simulation

2.1. Principle of the DRIE

2.1.1. Bosch Process The principle of the Bosch process is schematically shown in Fig. 1. The typical etch cycle lasts 5 to 15s, uses SF6 to etch silicon. In the next cycle, a fluorocarbon polymer, about 10nm thick, is plasma deposited using C4F8 as a source gas. In the following etch cycle the energetic ions (SFx+) remove the protective polymer at the bottom of trench, but the film remains relatively intact along the sidewalls. The repetitive alternation of etch and passivation steps results in high directional etch at rates between 1.5 and 4um/min, high aspect ratio up to 30:1, high selectivity to mask (75:1 to photoresist).

The computer simulation of DRIE uses a 3-D physical model which is simplified on some reasonable assumption, while maintains the main characteristics of the Bosch process. The following section gives the

* This paper is supported partly by the National High Technology Research and Development Program of China (2008AA01Z401), NSFC of China (90612001), RFDP of China (20070055054), Program for new century excellent talent in university(NCET-07-0464),and Science and Technology Development Plan of Tianjin (08JCYBJC13000) (60674068) (2006AA04Z304)

978-0-7695-3352-0/08 $25.00 © 2008 IEEE
DOI 10.1109/HPCC.2008.23

etching model and polymerization model in the 3-D DRIE simulation.

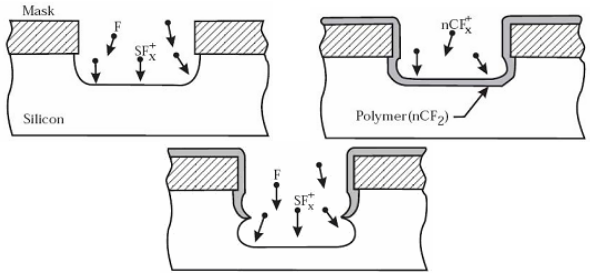


Figure 1. **Principle of Bosch process.**

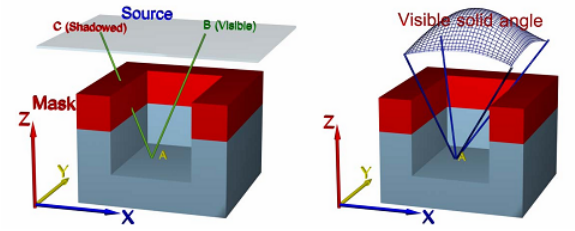


Figure 2. **Schematic illustration of shadow test and calculation of visible solid angle.**

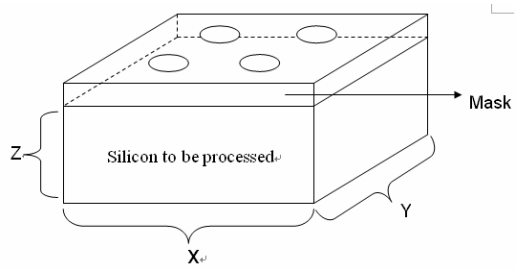


Figure 3. **The voxel data layout.**

2.1.2. Etching Model According to the analysis in [3], the etching in Bosch process can be seen as the combination of isotropic etching and ion-assistant etching. The isotropic etching leads to a uniform etching and the etch rate is described as a constant:

$$ER_{uni} = C_{uni} \qquad (1)$$

Where ER_{uni} is the uniform etching component and C_{uni} is a constant.

The ion-assistant etching is completed by the cooperation of ions and reactive radicals. The etching rate is assumed to be proportional to the number of incident ions and affected by the unshadowed solid angle and the surface normal. Fig. 2 gives a schematic illustration of shadow test and the calculation of solid angle. The shadow test is performed along a given direction following a discrete line of voxels from the exposed voxel to the source point. If any voxel on this line is a material voxel, the exposed voxel is shadowed. Then the calculation of visible solid angle is reduced to a series of shadow test.

The etch rate along the surface normal is expressed as:

$$ER_i = C_i \int_{\Omega} (n \bullet i) F(n \bullet i) d\Omega = C_i \int_{\Omega} \cos\alpha F(\alpha) d\Omega \qquad (2)$$

$$F(\alpha) = \frac{n+1}{2\pi} \cos^n \alpha \qquad (3)$$

C_i is a constant, n is the normal unit vector, i is the unit vector of the incident direction, α is the angle of incidence with respect to the surface normal. $F(\alpha)$ is the distribution function of ion flux which is described by a cosine power series. Ω is the visible solid angle. The linear combination of two etch rate components described above leads to

$$ER = ER_{uni} + ER_i = C_{uni} + C_i \int_{\Omega} \cos\alpha F(\alpha) d\Omega \quad (4)$$

ER is the total etching rate. This etch rate expression can be used for both silicon and polymer, but the parameters for silicon etching and polymer etching are different.

2.2. Serial Algorithm of the 3-D DRIE Simulation

In the 3-D DRIE simulation, the silicon is regarded as a set of points in a cuboid. Because the etching part is more complex than the polymerization process, the optimization mentioned blow is all about the part of etching process.

At each step of etching, first find all the points exposed to the air, and then the program will run a ray cast algorithm to calculate the amount of rays each point will receive. Then the etch rate of each point can be calculated from a series of physical formulas according to (1)(2)(3)(4). After the etch rate has been calculated, etch the silicon with a sphere whose center is the surface points and the radius is the etch rate. Fig.3 illustrates the data layout of the DRIE simulation. Algorithm-1 illustrates the serial algorithm of a single step of etching in 3-D DRIE simulation.

Algorithm 1. A single step of etching in 3-D DRIE simulation

```
Begin
For i = 1 to z do
  For j = 1 to x do
    For k = 1 to y do
      Begin
       If the point(i,j,k) of the silicon
is exposed to the air calculate the etch
rate using ray cast.
        etch the silicon with point(i,j,k)
as the sphere center and etch rate as
radius.
      End
End
```

The complexity of the algorithm is $O(n^6)$. So it is necessary to parallelize it.

3. Parallel algorithm

3.1. Basic Idea

Algorithm 1 provides a wide range of possibilities to implement the parallel algorithm. During the Bosch process, the entire silicon is divided into different regions which are etched independently with one or more masks. The etching processes of regions are same. So the religions can be distributed over nodes of multi-nodes system. Each node independently performs all the computations associated with its own region. Then we get a perfect coarse-grained parallel algorithm. It is easy to implement it on the message-passing platforms [4]. So we focus on fine-grained parallel algorithms on multi-core platform in this paper.

Dividing a computational task into smaller subtasks and assigning them to different processes for parallel execution are the two key steps in designing parallel algorithms. Once a computation has been decomposed into several tasks, these tasks are mapped onto processes with the objective that all the tasks complete in the shortest amount of elapsed time. In order to achieve a small execution time, the overheads of executing the tasks in parallel must be minimized. For a given decomposition, there are two key sources of overhead. The time spent in the inter-process interaction is one source of overhead. Another important source of overhead is the time that some processes may spend being idle. Both interaction and idling are often a function of mapping. Therefore, a good mapping of tasks onto processes must strive to achieve the twin objectives [5]: Reducing the amount of time processes spend in interacting with each other. Reducing the total amount of time some processes are idle while the others are engaged in performing some tasks.

As we implement the algorithm on a multi-core platform which is typically a shared-memory model, we do not need to consider the overhead of data transmission through the network. The parallel algorithm mainly focuses on how to decrease the synchronization time and achieve a good load balance.

Because successive etching steps are indeed data dependent, they can not be processed in parallel. But during an etching step, each surface point is processed independently. So we parallelize each etching step internally using multiple cores, and synchronize the cores between two successive etching steps. So the key objective of the parallel algorithm is balancing the load of each core, i.e., evenly distributing the computation tasks in each etching step to the cores.

The surface points associated with computations is in a 3D space. But they compose a curved surface. Namely, the computations are distributed over the curved surface instead of the whole 3D space. Moreover, as the number of etching steps is not very large, and passivation steps are applied, the shape of the "computation curved surface" is close to the shape of the mask on the top surface. So we don't consider three-dimensional data partition strategy, only distribute the tasks through the plane perpendicular to the z-axis which is defined as xy-plane in this paper. For the purpose of load balance, we examined several data partition strategies. In the following section, we will discuss these strategies.

3.2. Mapping Techniques for Load Balancing

3.2.1 Block Distributions Block distributions are some of the simplest ways to distribute an array and assign uniform contiguous portions of the array to different processes.

One-Dimensional Block Distribution

In one-dimensional block distribution, consider an n*n two-dimensional array A with n rows and n columns. We can now select one of these dimensions, e.g., the first dimension, and partition the array into p parts such that the kith part contains rows kn/p ... $(k+1)n/p-1$, where $0 \le k < p$. That is, each partition contains a block of n/p consecutive rows of A. Similarly, if we partition A along the second dimension, then each partition contains a block of n/p consecutive columns.

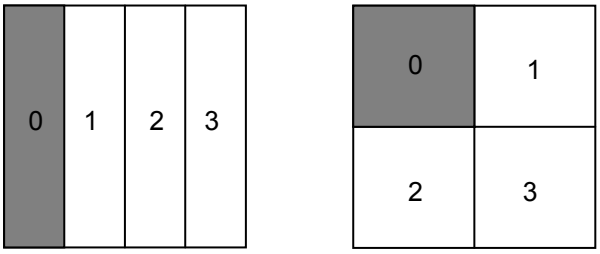
So we can parallelize the algorithm like this: divide the xy-plane through x or y direction into p equal-size areas and assign each process one area. An example is illustrated in Fig.4.a. Process 0 takes charge of an area which is gray, so do the other processes.

Two-Dimensional Block Distribution

If the shape of mask is not distributed uniformly along one dimension, one-dimensional block distribution results non-optimal load distribution. Unfortunately, practical masks often are not one-dimensional uniformly distributed. Instead of selecting a single dimension, we can select multiple dimensions to partition. For instance, in the case of array A we can select both dimensions and partition the plane into blocks such that each block corresponds to a $(n/p_1)*(n/p_2)$ section of the matrix, with $p = p_1*p_2$ being the number of processes.

Fig.4.b illustrates an example of two-dimensional block distribution, the plane is simply divided through x and y direction to 4 equal-size blocks. The advantage

of this method is that as long as the mask is center-symmetric, each part will have the same load.



a. one-dimensional b. two-dimensional

Figure 4. **Block distributions.**

3.2.2 Block-Cyclic Distributions If the load differs for different regions of the surface points of the silicon, a block distribution can potentially lead to load imbalance. The block-cyclic distribution is a variation of the block distribution scheme that can be used to alleviate the load-imbalance and idling problems. The central idea behind a block-cyclic distribution is to partition an array into many more blocks than the number of available processes. Then we assign the partitions to processes in a round-robin manner so that each process gets several non-adjacent blocks.

One-Dimensional Block Distribution

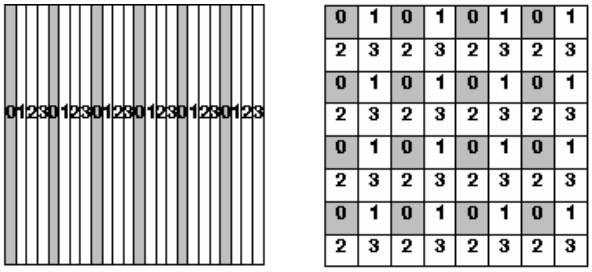
In a one-dimensional block-cyclic distribution of a matrix among p processors, the rows (columns) of an n*n matrix are divided into αp groups of $n/(\alpha p)$ consecutive rows (columns), where $1 \le \alpha \le n/p$. Here we introduce a term "granularity" which equals to α. Now, these blocks are distributed among the p processes in a wraparound fashion such that block b_i is assigned to process $P_i\%p$. This distribution assigns α blocks of the matrix to each process, but each subsequent block that gets assigned to the same process is p blocks away.

Fig.5.a illustrates an example of one-dimensional block cyclic distribution, each number represent a process. Process 0 takes charge of the area uniformly distributed on the plane, and the granularity is 6. So the plane is divided into 6*4 blocks with each process own 6 smaller blocks.

Two-Dimensional Block Distribution

Two-Dimensional Cyclic Block Distribution combines the two-dimensional block distribution and one-dimensional block cyclic distribution. We can obtain a two-dimensional block-cyclic distribution of an *n*n* array by partitioning it into square blocks of size $\alpha\sqrt{p} \times \alpha\sqrt{p}$ and distributing them on a hypothetical $\sqrt{p} \times \sqrt{p}$ array of processes in a wraparound fashion.

As Fig.5.b shows, each process takes charge of a set of smaller blocks uniformly distributing on the plane and the granularity is 4. Thus, there are 16 blocks on the plane overall. And each block is divided into 4 sub-blocks to be distributed to each processor. Algorithm 2 gives the pseudo code of a single step of etching in 3-D DRIE simulation using two-dimensional cyclic block distribution as mapping techniques.



a. one-dimensional b. two-dimensional

Figure 5. **Block-Cyclic distributions.**

The reason why a block-cyclic distribution is able to significantly reduce the amount of idling is that all processes have a sampling of tasks from all parts of the surface points of the silicon. As a result, even if different parts of the surface points of the silicon require different amount of work, the overall work on each process balances out. It has high possibilities that each process get equal amount of work. Also, since the tasks assigned to a process belong to different parts of the matrix, there is a good chance that at least some of them are ready for execution at any given time.

4. Experiment results

We used a computer with an Intel Core 2 Quad Q6600 CPU [6] with four 2.40GHz cores inside and 2 GB of DRAM running Windows XP as the experiment platform. The complier is MS Visual C++ 8.0 with options "/MD /arch: SSE2 /fp:fast". The program is based on multi-threads programming.

4.1. Performance of different mapping techniques

We first tested a single step of etching with the mask shown in Fig.6.a. The mapping techniques used with this mask is one-dimensional block distribution. The plane is divided into 4 blocks through the x-axis. We can learn from Tab.1 test.1 that, the 4 processors' execution time have little differences.

If the shape of the mask is not distributed uniformly along one dimension, the one-dimensional block distribution leads to load imbalance. Two dimensional block distribution can help solve this problem if the mask is center-symmetric. The mask showed in Fig.6.b is a mask of this kind. Tab.1 test.2 and test.3 show the etching time of mask #2 using the two strategies respectively. One-dimensional block distribution leads

to serious load imbalance, while two-dimensional block distribution obtains a better load balance.

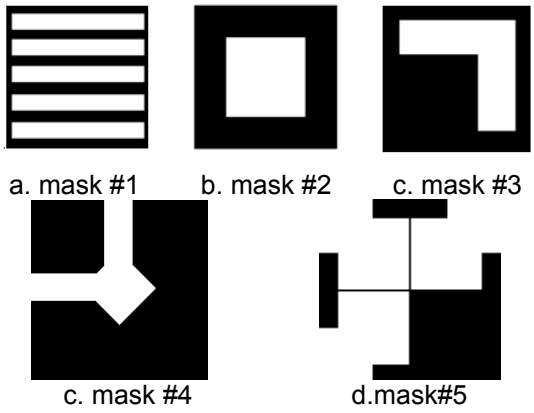


Figure 6. **Mask used in the simulation.**

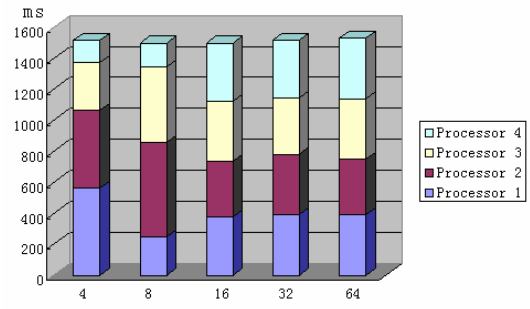
When the mask is irregular, e.g., mask #3, the block distribution doesn't work too. We will use block-cyclic distribution instead. Tab.1 test.4~test.7 gives the execution time of the four kinds of mapping techniques applied to mask #3. Two block-cyclic distribution strategies obtain load balance, and the run time is much less than two block distribution methods. It agrees well with the analysis in section 3.2.2.

Table 1. **Single step etching time using different mapping techniques.**

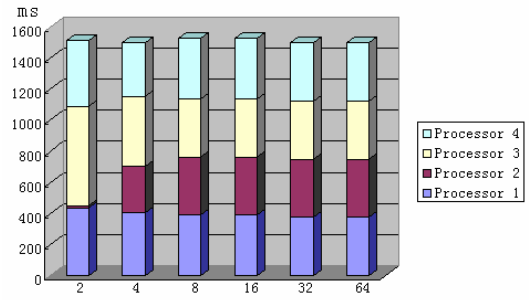
Processor / Test	1	2	3	4
1	575	625	625	575
2	141	703	719	156
3	422	422	422	437
4	172	281	547	531
5	422	0	625	422
6	375	375	344	375
7	375	375	375	375

4.2. Impact of granularity

Next, we tested the cyclic distribution methods with different granularities. We used mask #3. From the figure illustrated in Fig.7; we can learn that granularity is an important factor which can impact load distribution. In fact, block-cyclic distribution will degenerate into block distribution when granularity=1. As granularity increases, the computation tasks are distributed more evenly. For mask #3, one-dimensional block-cyclic distribution achieves perfect load balance when granularity is greater than 16 and the critical point of two-dimensional block-cyclic distribution is 8.



a. one-dimensional block-cyclic distribution



b. two-dimensional block-cyclic distribution

Figure 7. **Impact of granularity.**

4.3. Speedup to the serial algorithm

We then compared the parallel algorithm and the serial algorithm. We use mask #3, and the parallel algorithm used two-dimensional block-cyclic distribution, the granularity is set to be 16. Fig.8 clearly shows that parallel algorithm can significantly reduce the run time. We can learn that the serial algorithm's execution time is about 3.95 times as long as the two-dimensional block-cyclic distribution algorithm from Fig-16. Because the there are 4 cores in our system, the speedup is very good.

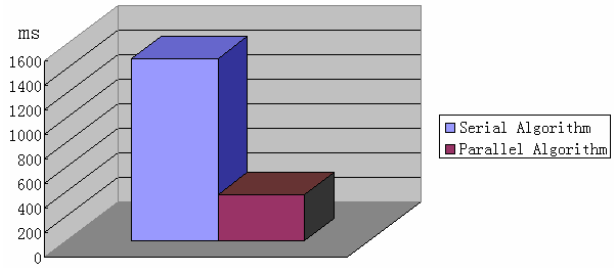


Figure 8. **Speedup.**

Besides, we tested the speedup of the parallel program with compiler option "/arch: SSE 2" to the one without it. With this option, the compiler will turn on **S**treaming **SIMD** **E**xtensions (SSE) [7] which is a SIMD (Single Instruction, Multiple Data) instruction

set extension to the x86 architecture. The complier will automatically optimize the code to fit the x86 architecture. As there are lots of floating-point calculations in the ray cast step and the etching step, the optimization we made is to align the data at 128bit which will let the SSE processor run at full spped [8]. Tab.2 shows the comparison between the two kinds (with and without the compiler option), the mask used is mask #4 shown in Fig. 6.d. The original mask (256*256) and the zoomed in mask (512*512) are tested. It shows that there is a substantial speedup to the trivial one.

Table 2. **Impact of SSE.**

Mask	With /arch: SSE2 option (ms)	Without /arch: SSE2 option (ms)	Speedup
256*256	219	156	1.40
512*512	641	984	1.53

4.4. Multi-step etching

At last, we test multi-step etching. We alternately etch and passivate the silicon several times to get a deep trench. The objective of this experiment is to test whether or not two-dimensional block-cyclic distribution still works in multi-step etching. If not, we have to use three-dimensional distribution. The mask used in this experiment is mask #5 shown in Fig. 6.e. Fig.9 shows the experiment result. The figure shows the load of each processor of 10 continuous steps of the etching.

From the experiment result, we can learn that when etching several steps, each processor get about 25% of the total work. So the algorithm still works in multi-step etching. Though the etching is in 3D space, the computation is distributed on the surface of the silicon which is close to a 2D plane.

6. Conclusion

In this paper, we examined several parallel approaches to accelerate the 3-D DRIE simulation. The experiment result shows that two-dimensional block-cyclic distribution approach can lead to perfect load balance in most cases and can utilize fully the computational power of new multi-core CPUs. Besides the x86 architecture's SSE feature can substantially speedup the computing process.

The mapping techniques we mentioned in this paper are all static task mapping. In the future work, we will try some methods such as randomized block distributions and dynamic task mapping which include centralized schemes and distributed schemes. Besides, the hybrid model which uses MPI, OpenMP and multithreading programming to distribute the computing task to the processors and GPU programming are also good choices.

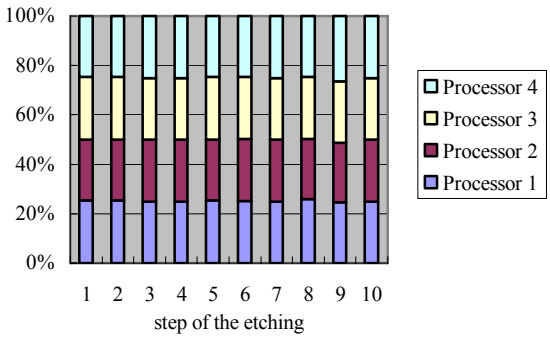


Figure 9. **Load balance when multi-step etching.**

References

[1] L¨armer F and Schilp A Method of anisotropically etching silicon, US Patent Specification 5501893, German Patent Specification DE4241045.

[2] Guangyi Sun, Xin Zhao, Haixia Zhang, Lei Wang, and Guizhang Lu. 3-D Simulation of Bosch Process with Voxel-Based Method, Proceedings of the 2nd IEEE International Conference on Nano/Micro Engineered and Molecular Systems, Bangkok, Thailand,2007：pp. 45-49, Jan 2007.

[3] Rongchun Zhou, Haixia Zhang, Yilong Hao, and Yangyuan Wang, "Simulation of the bosch process with a string-cell hybrid method," IOP J. Micromech. Microeng, vol. 14, pp. 851-858, 2004.

[4] Michael J.Quinn, Parallel Programming in C with MPI and OPENMP, McGraw-Hill Companies,Inc.2004

[5] Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, Introduction to Parallel Computing (Second Edition), Pearson Education, 2003.

[6] R.M. Ramanathan, Intel Multi-Core Processors Making the Move to Quad-Core and Beyond, White Paper Intel(R) Multi-Core Processors, 2007

[7] http://en.wikipedia.org/wiki/Streaming_SIMD_Extension

[8] Aart Bik, Milind Girkar, Paul Grey, and Xinmin Tian, Programming Guidelines for Vectorizing C/C++ Compilers, http://www.ddj.com/cpp/184401611?pgno=1, 2003