Distributed model calibration using Levenberg-Marquardt algorithm

Mark Lu[a], Liang Zhu[a], Li Ling[b], Gary Zhang[b], Walter Chan[c], Xin Zhou[*c]
[a]Grace Semiconductor Manufacturing Corp, 818 GuoShouJing Rd, Zhangjiang, Shanghai, China
[b]Anchor Semiconductor Inc., 666 East Beijing Rd, Shanghai, China
[c]Anchor Semiconductor Inc., 5403 Betsy Ross Drive, Santa Clara, CA 95054

ABSTRACT

The number of tunable parameters increases dramatically as we push forward to the next node of hyper-NA immersion lithography. It is very important to keep the lithographic process model calibration time under control, and its end result insensitive to either the starting point in the parameter space or the noise in the measurement data. For minimizing the least-squares error of a multivariate non-linear system, the industry standard is the Levenberg-Marquardt algorithm. We describe a distributed computing technique that is natural to the algorithm, and easy to implement in a cluster of computers. Applying this technique to calibrating lithographic process model, we can achieve robust optimization results in nearly constant calibration time.

Keywords: process model, calibration, distributed computing, Levenberg-Marquardt, computational lithography, numerical stability

1. INTRODUCTION

For computational lithography, calibrating the process model is the foremost and central task[1]. The measurement-prediction fit is a crucial criterion in determining whether the calibration has produced a workable model. The process model usually has a dozen or even scores of physical and empirical parameters. These parameters must be given reasonable initial values, but they are eventually determined by the model calibration process.

Using nested loop or mesh method to optimize a multivariate model is straightforward, but even when the number of parameters exceeds two, this method becomes too slow to be practical. Because the number of tunable parameters increases dramatically as we push forward to the next node of hyper-NA immersion lithography, it is very important to keep the lithographic process model calibration time under control, and its end result insensitive to either initial conditions or noise in the measurement data. There actually exists a powerful algorithm to travel and search in the high-dimensional space, namely the Levenberg-Marquardt algorithm.

We set out to start a project to combine the strength of the algorithm with a distributed computing environment. The stated performance goals of our project are shorten model-building time, improve coverage of parameter space, decrease sensitivity of the final model to data noise, and increase consistency of results among different users.

The following paper before conclusion is organized into five sections: brief description of the Levenberg-Marquardt algorithm; illustration of distributing the computational jobs; experimental results; discussion of numerical stability issues; further enhancements.

* xin.zhou@anchorsemi.com

Optical Microlithography XX, edited by Donis G. Flagello, Proc. of SPIE Vol. 6520, 65203C, (2007)
0277-786X/07/$18 · doi: 10.1117/12.711564

2. BRIEF DESCRIPTION OF LEVENBERG-MARQUARDT ALGORITHM

The famed Levenberg-Marquardt algorithm[2] is the industry standard when it comes to the optimization of a multivariate non-linear system posed as a least-squares problem. It can be considered a mix of steepest descent and Newton's method.

Let's define some basic variables first. Here we use vector $\boldsymbol{g}$ to represent the M computed gauges,

$$\boldsymbol{g} = (g_0, g_1, g_2, ..., g_{M-1}), \quad \text{Equation 1}$$

And we use vector $\boldsymbol{m}$ for the M measured gauges,

$$\boldsymbol{m} = (m_0, m_1, m_2, ..., m_{M-1}). \quad \text{Equation 2}$$

The difference between the measurements and the predicted gauges is denoted by the error vector $\boldsymbol{e}$,

$$\boldsymbol{e} = \boldsymbol{m} - \boldsymbol{g}. \quad \text{Equation 3}$$

Most importantly, the cost function our algorithm is trying to minimize is $\| \boldsymbol{e} \|^2$.

Suppose we have in total N parameters for our process model,

$$\boldsymbol{p} = (p_0, p_1, p_2, ..., p_{N-1}) \quad \text{Equation 4}$$

The entire pattern transfer process, from mask layout to wafer circuit, is completely characterized by this point $\boldsymbol{p}$ in the parameter space.

Our process model will determine the values of $\boldsymbol{g}$ vector when given a $\boldsymbol{p}$ vector. In other words, the lithography process model is a mapping from $\boldsymbol{p}$ to $\boldsymbol{g}$.

If we perturb the $\boldsymbol{p}$ vector by a tiny amount, we expect the $\boldsymbol{g}$ vector to also change a small amount. Hence we can define the derivative of $\boldsymbol{g}$ with respect to $\boldsymbol{p}$, and call it the Jacobian $\boldsymbol{J}$.

$$\boldsymbol{J}_{ik} = \frac{\partial g_k}{\partial p_i} \quad \text{Equation 5}$$

In our implementation, we use finite difference to compute the Jacobian.

The essence of the Levenberg-Marquardt algorithm is to approximate the non-linear system near the minimum with a quadratic system, yet the second derivative matrix is replaced with a simpler Hessian matrix constructed from only first derivatives, and its diagonal elements are augmented in a way that depends on how good the quadratic approximation is.

To make the algorithm more robust, there is a trial step that can be retracted and adjusted, when the fitting error is expected to rise instead of fall at the next step. This trial step is linked to and enabled by the augmentation of the diagonal elements. The original Gauss-Newton method does not allow this self-recovery mechanism.

In the following Fig. 1, we give a listing of pseudo-code that captures the major steps of our implementation in a distributed computing environment.

```
Initialize everything;

For (iter=0; iter<maxIter; iter++) {
   If (ρ > 0) {
      g = computeGauges(p);
      e = m - g;

      For (index=0; index<N; index++) {
         p[index] += δ; // δ is a positive small number
         g' = computeGauges(p); // Assigned to another computer
         p[index] -= δ;
         J[index] = (g' - g)/ δ;
      }
   }

// Normal equation
   A = J * J^T + μI; // μ is a positive number
   b = J * e;
   If (|b| is small) break;
   Solve Ax = b;
   If (|x| is small) break;

// Trial step
   p' = p + x;
    g' = computeGauges(p');
   Compute linear gain ρ according to reference 2;
   If (ρ > 0) {
      decrease μ;
      p = p';
   } else {
      increase μ; // but p stays the same
   }
}
```

Fig. 1 Outline of our implementation of the Levenberg-Marquardt algorithm in a distributed computing environment. The bold Italic font denotes a vector or a matrix, lower case for vector, upper case for matrix.

If we use first derivative information in a straightforward way such as the steepest-descent method, the optimization path in a high dimensional space may become extremely inefficient. In Fig. 2 we sketch the optimization paths of the two algorithms with artistic license.

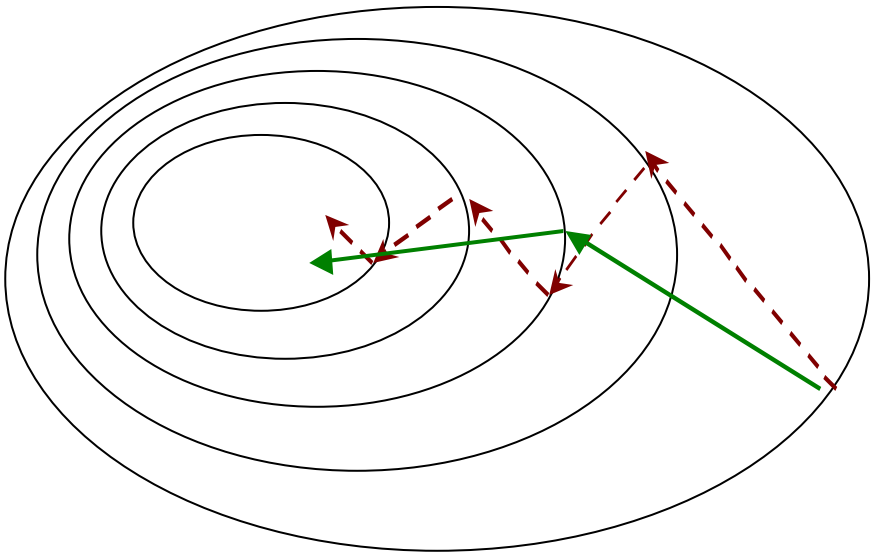

Fig. 2 Compare steepest descent (red dashed arrow) with Levenberg-Marquardt (green solid line). The ovals are different levels of minimization target function, e.g. the fitting error in the context of process model calibration.

3. DISTRIBUTE THE COMPUTING JOBS

The pseudo-code in Sec. 2 alludes to the use of distributed computing technique in the line

g' = computeGauges(***p***); // Assigned to another computer

The insight that leads us to parcel out the jobs to a cluster of computers has two parts: one is that the different rows of the Jacobian matrix can be computed independently of each other, the other is that model-construction is the most time consuming part of "nsmodel", the lithographic process modeling software of Anchor Semiconductor. These two parts can benefit the most from exploiting parallelism.

The benefit of parallelism comes naturally. Since each perturbed parameter vector represents a distinct process model, we can simply let a distinct computer to evaluate the system at that point in the parameter space. The selection of a computer in a cluster is linked to the index of the parameter that is being perturbed for computing a particular row of the Jacobian. The following figure illustrates our philosophy of parallel job assignment.

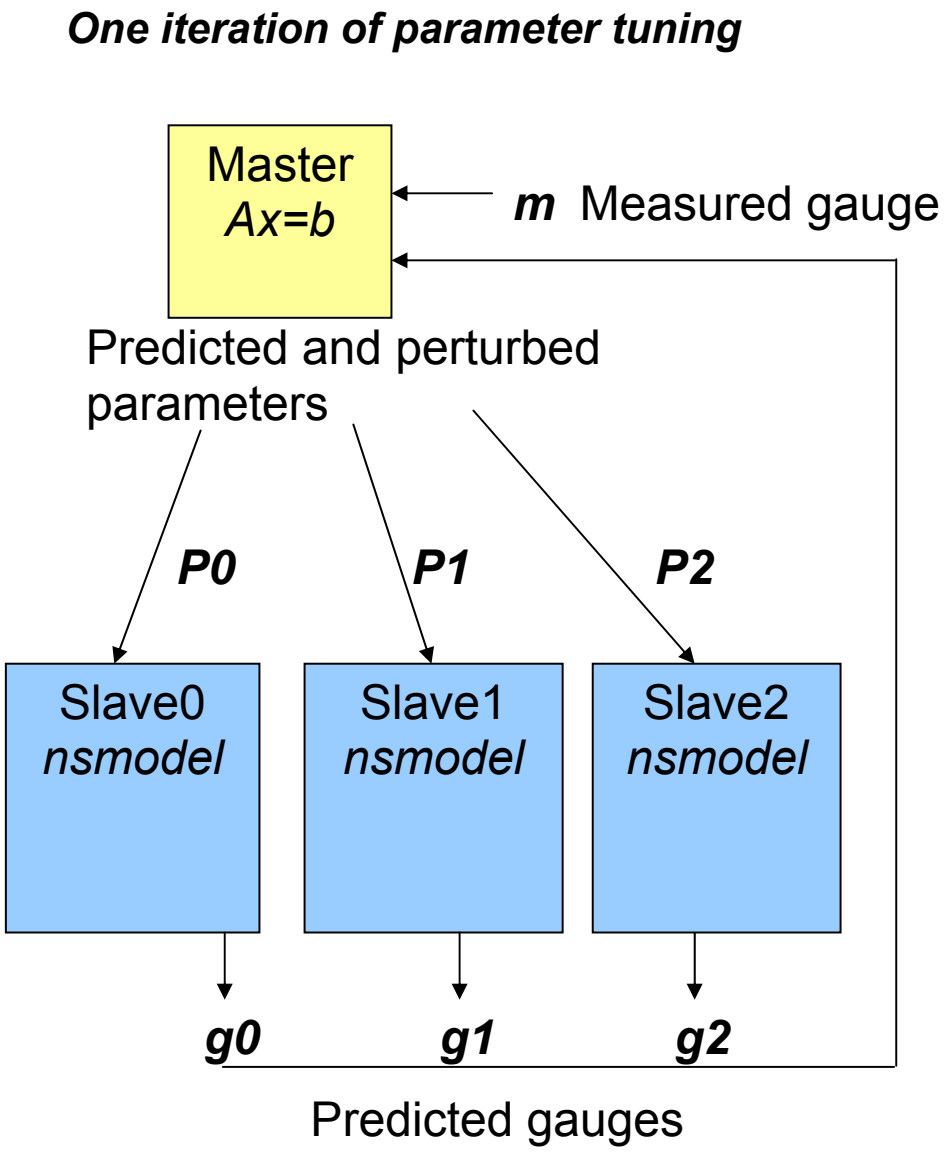

Fig. 3. The master process builds and solves the system of linear equations. The solution predicts the next change of the parameter vector that minimizes the fitting error. The slave processes receive the trial parameter vectors from the master, and proceed to construct models and compute gauges based on the given parameters. The slaves then report the gauge vectors back to the master for the next iteration. Here "nsmodel" is the name of the program of Anchor Semiconductor, Inc. that constructs and calibrates a process model.

4. RESULTS AND DISCUSSION

We have applied our distributed model calibration flow to numerous customer cases. Fig. 4 is an example taken from a real world experience building a contact layer model, which shows a remarkably rapid approach to the final point in the parameter space from an initial point that is very far away.

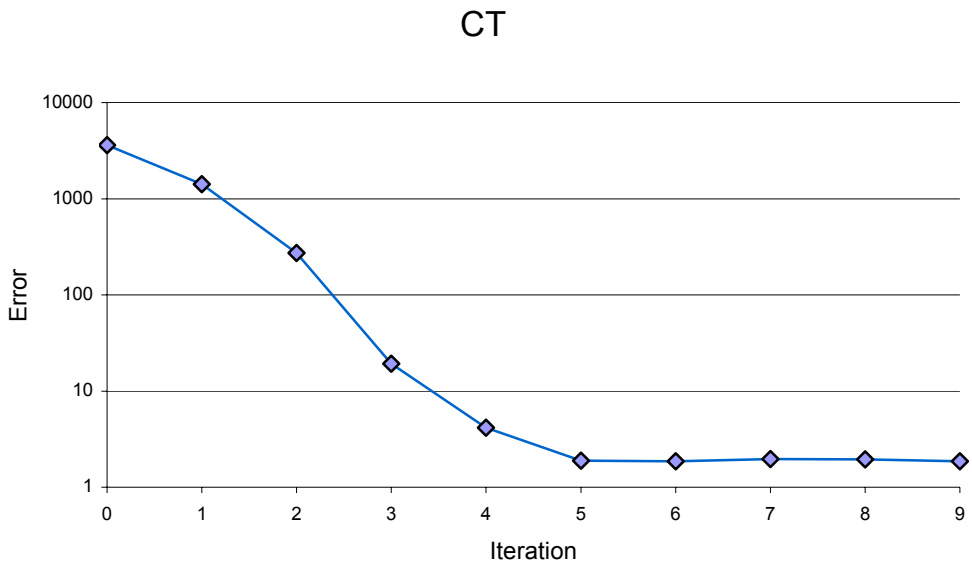

Fig. 4 Error of model fitting vs. iteration number.

What happens if we start from two different beginnings, do we get to the same end? The following Fig. 5 answers it.

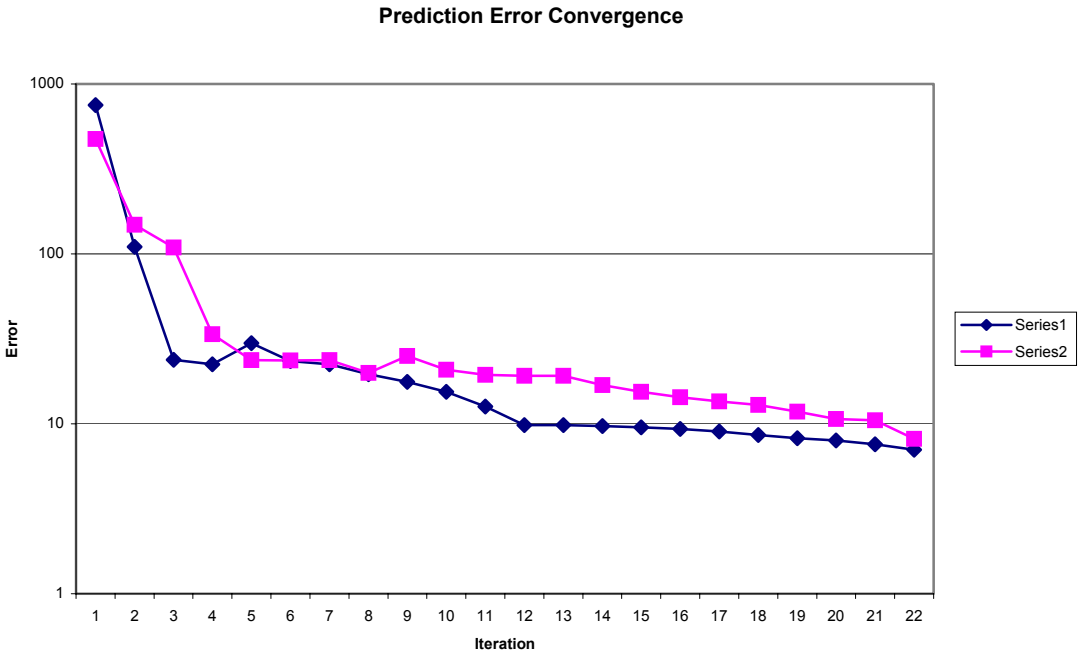

Fig. 5 Two sequences of fitting error vs. iteration number. Their initial parameters are very different.

The two curves start from substantially different initial conditions, yet they reach essentially the same limiting condition after only a few iterations. In a 20-CPU cluster, (in our case each CPU is an AMD64 2GHz or better) this whole process takes less than two hours. The bumps at #5 for curve 1 and #9 for curve 2 are indicative of the self-recovery capability of the Levenberg-Marquardt algorithm. A straightforward Newton's method would have sent the calibration path off the chart.

The following SEM micrographs overlaid with simulated contours demonstrate the goodness of the final result.

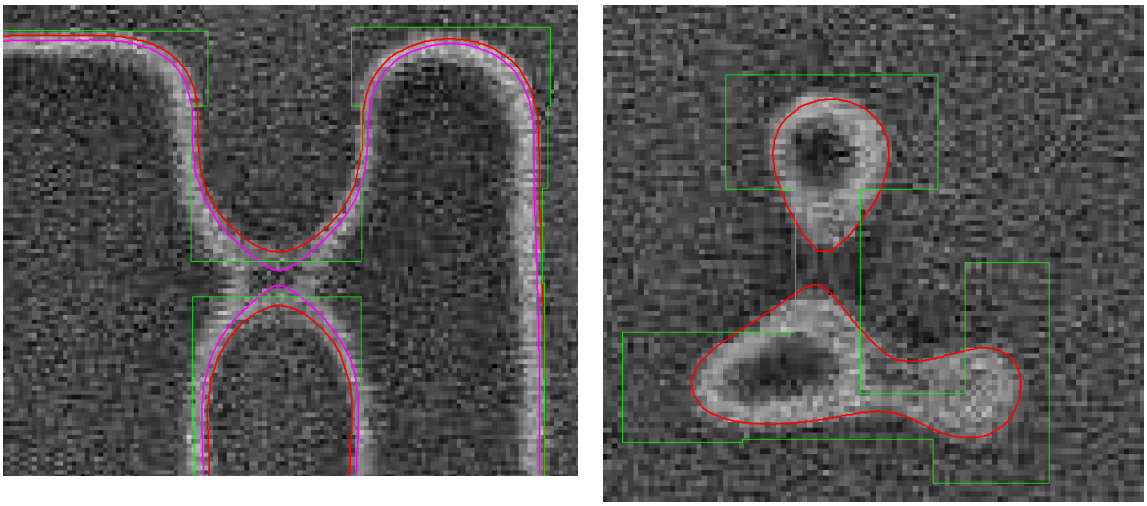

Fig. 6 The left panel shows a potential break in one process layer, where the green line is mask pattern, red is contour at nominal condition, pink is 10% over-dose. The right panel shows an actual break in the same layer, where green is mask pattern, red is nominal condition contour.

5. NUMERICAL STABILITY ISSUES

In solving non-linear least squares problems, we often encounter singular or very ill-conditioned matrices. For the Levenberg-Marquardt method, since the diagonal elements of the Hessian are augmented, we will never encounter a singular matrix when solving Ax=b.

The matrix of our system could still be ill-conditioned, and we can scale the physical parameters appropriately to resolve this issue. The scaling of individual parameters is a simple form of pre-conditioning[3] in numerical linear algebra.

Another simple way of pre-conditioning the system in order to speed up the optimization convergence is by a change of variable. For example, directly using σ the Gaussian width as a tuning parameter for a Gaussian function is not as effective as tuning another variable that is closely tied to σ.

There are computational issues unique to distributed computing that are not found in a single thread application of the Levenberg-Marquardt algorithm. Because the slave computers could return their results in different order even when we start the program from exactly the same initial conditions, the master computer will have a slightly different mathematical problem to solve. This uncontrollability will be exaggerated if some of the computers in the cluster are of a different architecture than others. For example, a few computers have 64bit CPUs, a few have 32bit CPUs, and they run different operating systems. While our flexible software architecture allows this kind of mix-and-match cluster, the optimization sequence will take a different path from run to run. The end result is also dependent on the exact computing environment.

Real world situation is not so dire as the above suggests. Insofar as our experience with thousands of model calibration runs under vastly different hardware configurations, we have not seen the end results of distributed model calibration differ so much as to contradict each other. The differences are well within normal tolerances, e.g. CD error variation is much smaller than 1nm. However, we know of no theoretical estimate as to the upper bound of end result variation due to the nature of distributed computing environment. This is certainly a fascinating subject to study.

6. FURTHER ENHANCEMENTS

When there are a multitude of local minimums suspected, we need to devise ways to search for the global minimum. it is desirable to start from a few sets of initial parameters that are substantially different, distribute the calibration jobs among a cluster of computers, and automatically present the finalists for review. The technical difficulty is in coming up with a set of programmable objective criteria that judges which result represents the most meaningful process model. Model validation is a topic in itself. The usual method of SEM-contour overlay can be automated and linked up with the whole model calibration flow.

Most parameters are only physically meaningful within a certain range. For example, the NA of a stepper can not be adjusted as freely as the defocus amount, for a stepper can set its NA very accurately but cannot set its absolute focal plane relative to wafer surface. We need to control the range of movement for different parameters. A simple way of damping the sensitivity of a parameter is to artificially add a positive diagonal term in the Hessian corresponding to that parameter and let the penalty grow proportional to the deviation of the parameter from its intended natural value.

Some parameters have a hard physical limit. For example, if the mask corner rounding is used as free parameters, the optimization process could demand its value to be negative at the next step. This type of cases has to be treated properly to avoid optimization break-down, because all the other parameters are adjusted based on the assumption that this negative corner rounding would go through to reduce total error. Setting parametric limits in the Levenberg-Marquardt algorithm makes the implementation more complicated.

7. CONCLUSIONS

Using Levenberg-Marquardt algorithm in a distributed computing environment naturally extends the power of the algorithm. As our example shows, from different initial conditions, we can reach essentially the same destination in the parameter space using only a few iterations. Each iteration can handle as many parameters as the size of the computer cluster, hence enabling model calibration in nearly constant time even if the total number of parameters explodes. This is a powerful tool in our quest to build the best lithographic process model to help the IC industry march down the technology node today and tomorrow.

REFERENCES

1. Edward W. Conrad, Daniel C. Cole, David P. Paul, and Eytan Barouch, "Model considerations, calibration issues, and metrology methods for resist-bias models", Proc. SPIE Vol. **3677**, 940-955 (1999).
2. K. Madsen, H.B. Nielsen, O. Tingleff, *Methods for Non-Linear Least Squares Problems*, 2nd edition, IMM(2004) http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3215/pdf/imm3215.pdf.
3. Lloyd N. Trefethen, and David Bau, *Numerical Linear Algebra*, SIAM (1997).