Using GPUs to Speedup Computational Lithography

Constantin Chuyeshov
Bayram Yenikaya

Rough Outline

- Description of a problem
- Simplified algorithms
- CPU and GPU Results
- Full-scale problem
- CPU and GPU Results

Cross-section of a computer chip

…

TaN/Ta
barrier metal
electro-deposition seed

Si_3N_4 Etch-stop layers ~30nm

Cu lines
Metal-2 Layer

Cu lines
Metal-2 Layer

Cu contacts
(conical shaped)

~600 nm

Cu lines
Metal-1 Layer

Cu lines
Metal-1 Layer

~600 nm

W (Tungsten)
plugs

Gate Dielectric
2 nm

Metal-Silicide

Poly-Si
Gate

SiO2
~600 nm thick

Isolation
trench
SiO_2

L_g

Doped Si

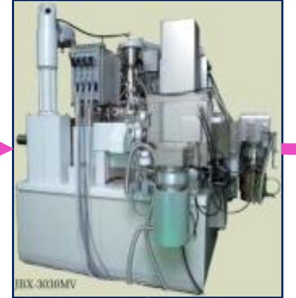
Crystaline Si Substrate

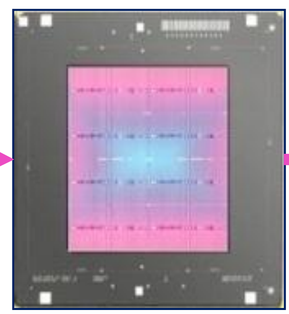
~7 layers of metal
interconnects

Transistors

How the chips are made

Mask making



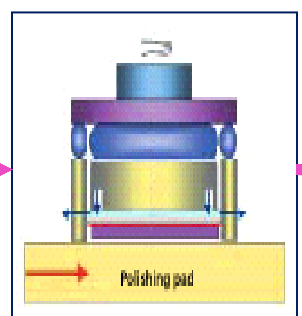


Lithography

Resist processing

Etching

CMP

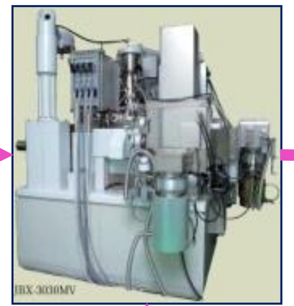


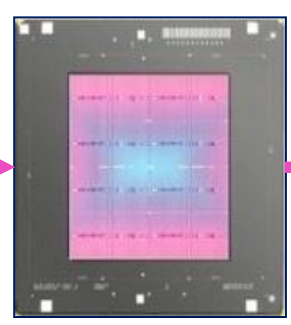
Measurements

Repeat for each layer

How the chips are made

Mask making



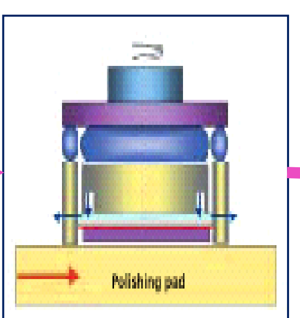


Lithography

Resist processing

Etching

CMP



Measurements

Repeat for each layer

The process

Light source

mask

projection optics

photoresist

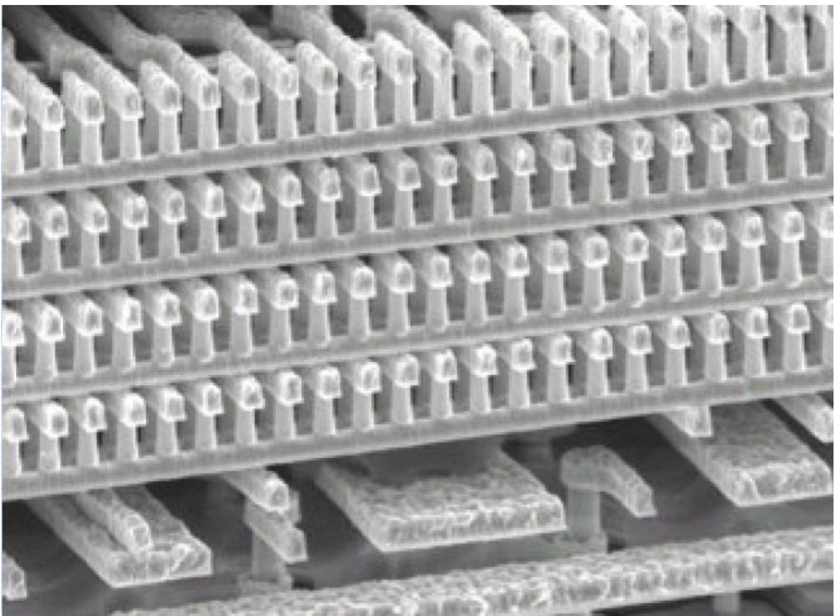
deposited layer
to be patterned

developed resist

layer etched

resist removed

Result:



The process

Light source

mask

projection optics

photoresist

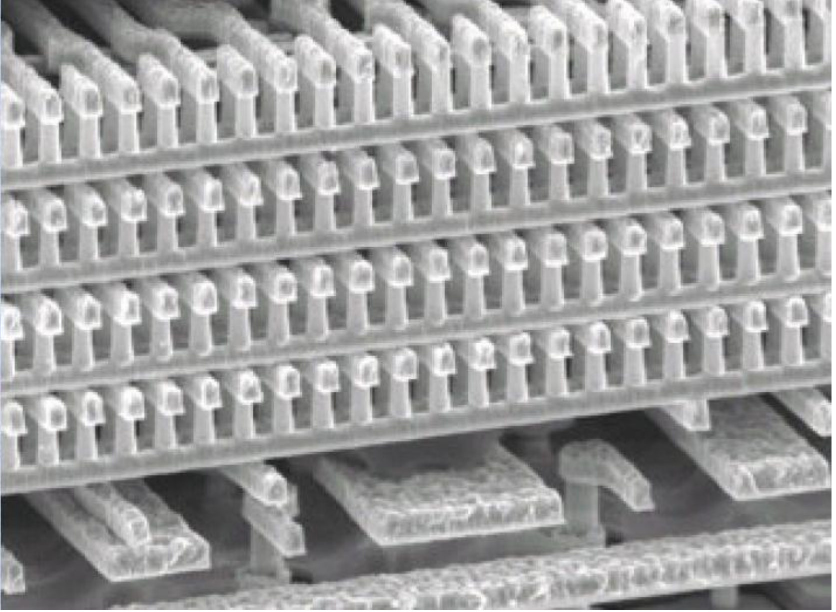
deposited layer
to be patterned

developed resist

layer etched

resist removed

Result:



What is Lithography?

Light source

Illumination optics

Mask

Projection optics

Photoresist on wafer

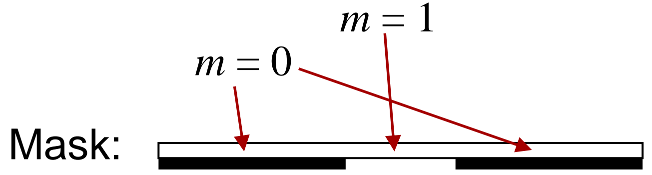
Why Computational Lithography?

To print a satisfactory layer of a wafer you need to:

- Design and manufacture the mask
- Deposit various films on the wafer
- Expose the mask (optical lithography)
- Develop the resist
- Etch the unneeded areas
- Fill in the etched gaps with insulator
- Polish the surface

This might have to be done several times per layer to optimize the mask design and to improve yield. It's <u>much</u> cheaper and faster to simulate it on a computer, than to manufacture.

Simple optical lithography model

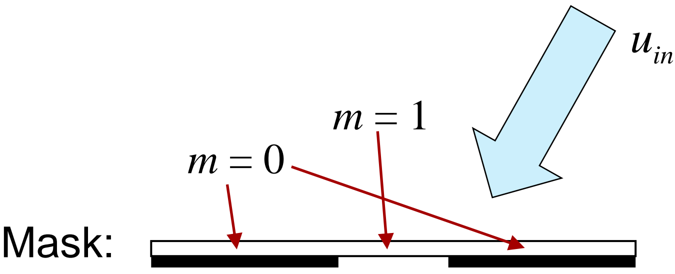


$m(r_\perp)$ -- mask function

$\hat{m}(k_\perp)$ -- its Fourier Transform

$$m(r_\perp) = \iint \exp(ik_\perp \cdot r_\perp) \cdot \hat{m}(k_\perp) dk_\perp$$

Simple optical lithography model



$u_{inc}(r_\perp)$ -- plane-wave incident field from a single-point light source

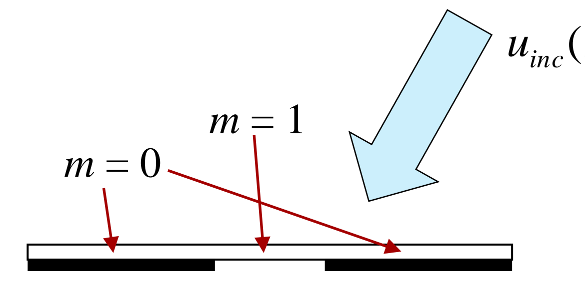
$$u_{inc}(r_\perp) = \exp(ik_\perp^{(i)} \cdot r_\perp)$$

$m(r_\perp)$ -- mask function

$\hat{m}(k_\perp)$ -- its Fourier Transform

$$m(r_\perp) = \iint \exp(ik_\perp \cdot r_\perp) \cdot \hat{m}(k_\perp) dk_\perp$$

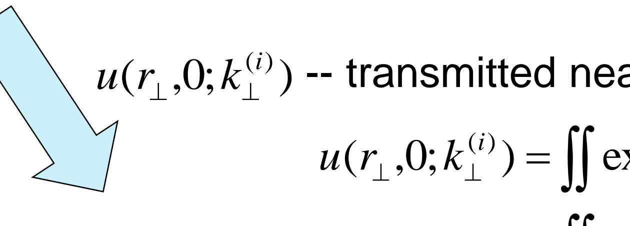
Simple optical lithography model



$u_{inc}(r_\perp)$ -- plane-wave incident field from a single-point light source

$$u_{inc}(r_\perp) = \exp(ik_\perp^{(i)} \cdot r_\perp)$$

Mask:



$m(r_\perp)$ -- mask function

$\hat{m}(k_\perp)$ -- its Fourier Transform

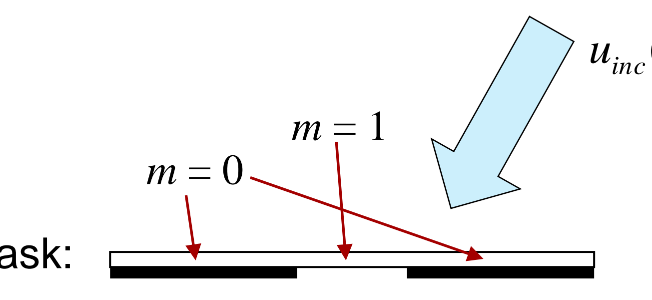
$$m(r_\perp) = \iint \exp(ik_\perp \cdot r_\perp) \cdot \hat{m}(k_\perp) dk_\perp$$

$u(r_\perp, 0; k_\perp^{(i)})$ -- transmitted near field at $z = 0$

$$u(r_\perp, 0; k_\perp^{(i)}) = \iint \exp\left(i\left[k_\perp + k_\perp^{(i)}\right] \cdot r_\perp\right) \cdot \hat{m}(k_\perp)\, dk_\perp$$

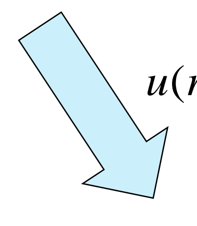
$$= \iint \exp\left(ik_\perp \cdot r_\perp\right) \cdot \hat{m}\left(k_\perp - k_\perp^{(i)}\right) dk_\perp$$

Simple optical lithography model



$u_{inc}(r_\perp)$ -- plane-wave incident field from a single-point light source

$$u_{inc}(r_\perp) = \exp(ik_\perp^{(i)} \cdot r_\perp)$$



$m(r_\perp)$ -- mask function

$\hat{m}(k_\perp)$ -- its Fourier Transform

$$m(r_\perp) = \iint \exp(ik_\perp \cdot r_\perp) \cdot \hat{m}(k_\perp) dk_\perp$$

$u(r_\perp, 0; k_\perp^{(i)})$ -- transmitted near field at $z = 0$

$$u(r_\perp, 0; k_\perp^{(i)}) = \iint \exp\left(i\left[k_\perp + k_\perp^{(i)}\right] \cdot r_\perp\right) \cdot \hat{m}(k_\perp)\, dk_\perp$$
$$= \iint \exp(ik_\perp \cdot r_\perp) \cdot \hat{m}\left(k_\perp - k_\perp^{(i)}\right) dk_\perp$$

Propagated near-field:

$$u(r_\perp, z; k_\perp^{(i)}) = \iint \exp\left(ik_\perp \cdot r_\perp - ik_z z\right) \cdot \hat{m}\left(k_\perp - k_\perp^{(i)}\right) dk_\perp$$

$$k_z = \sqrt{\left(\frac{2\pi n}{\lambda}\right)^2 - k_\perp \cdot k_\perp}$$

λ – wavelength

n – refractive index

cadence™

Simple optical lithography model

Including the lens aberrations and reflections ($P(k_\perp)$), the field becomes:

$$u(r_\perp, z; k_\perp^{(i)}) = \iint \exp\left(ik_\perp \cdot r_\perp - ik_z z\right) \cdot P(k_\perp) \cdot \hat{m}\left(k_\perp - k_\perp^{(i)}\right) dk_\perp$$

Simple optical lithography model

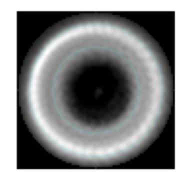
Including the lens aberrations and reflections ($P(k_\perp)$), the field becomes:

$$u(r_\perp, z; k_\perp^{(i)}) = \iint \exp\left(ik_\perp \cdot r_\perp - ik_z z\right) \cdot P(k_\perp) \cdot \hat{m}\left(k_\perp - k_\perp^{(i)}\right) dk_\perp$$

Light intensity in the photoresist:

$$U(r_\perp, z) = \iint \left| u(r_\perp, z; k_\perp^{(i)}) \right|^2 \cdot I(k_\perp^{(i)})\, dk_\perp^{(i)}$$

$I(k_\perp^{(i)})$ -- illumination function



Algorithm description

Given all the model parameters:

- Compute Fourier coefficients of the mask
- Compute light intensity at each point in the photoresist

Algorithm description

Given all the model parameters:

- Compute Fourier coefficients of the mask
- Compute light intensity at each point in the photoresist

However, there is a known physical limit on resolution of the optical system.

Thus, a faster algorithm:

- Compute Fourier coefficients of the mask
- Compute light intensity on coarse grid in the photoresist
- Interpolate the result to a dense grid

GPU vs CPU Results, Algorithm 1 (small clip)

	CPU, 2.8 GHz, 1 core	Tesla C2070 (Fermi), 1.15 GHz, CUDA 4.2	
Copy the polygons	-	0.4 ms	
Computation of Fourier coefficients	120 ms	7.2 ms	16.5x
Coarse intensity computation	2025 ms	169 ms	12x
Resample to a dense grid	24 ms	0.2 ms	~100x
GPU return intensity	-	2.2 ms	
Total	2169 ms	179 ms	12x

Total GPU times include data transfers from CPU to GPU memory.

Last column is C2070 speedup with respect to CPU.

A different approach

If model parameters are not changing, then another simplification is possible.
As before, light intensity in the photoresist:

$$U(r_\perp, z) = \iint \left| \iint \exp\left(ik_\perp \cdot r_\perp - ik_z z\right) \cdot P(k_\perp) \cdot \hat{m}\left(k_\perp - k_\perp^{(i)}\right) dk_\perp \right|^2 \cdot I(k_\perp^{(i)})\, dk_\perp^{(i)}$$

cādence™

A different approach

If model parameters are not changing, then another simplification is possible.
As before, light intensity in the photoresist:

$$U(r_\perp, z) = \iint \left| \iint \exp\left(ik_\perp \cdot r_\perp - ik_z z\right) \cdot P(k_\perp) \cdot \hat{m}\left(k_\perp - k_\perp^{(i)}\right) dk_\perp \right|^2 \cdot I(k_\perp^{(i)})\, dk_\perp^{(i)}$$

Which can be rewritten in a quadratic form as

$$U(r_\perp, z) = \langle m | H | m \rangle$$

Here H is a positive semi-definite self-adjoint operator.

A different approach

If model parameters are not changing, then another simplification is possible.
As before, light intensity in the photoresist:

$$U(r_\perp, z) = \iint \left| \iint \exp\left(ik_\perp \cdot r_\perp - ik_z z\right) \cdot P(k_\perp) \cdot \hat{m}\left(k_\perp - k_\perp^{(i)}\right) dk_\perp \right|^2 \cdot I(k_\perp^{(i)})\, dk_\perp^{(i)}$$

Which can be rewritten in a quadratic form as

$$U(r_\perp, z) = \langle m | H | m \rangle$$

Here H is a positive semi-definite self-adjoint operator. Thus, the eigenvalue expansion gives

$$U(r_\perp, z) = \langle m | H | m \rangle = \sum_n \mu_n \left| \langle V_n | m \rangle \right|^2 = \sum_n \mu_n \left| \iint V_n(q_\perp) \cdot m(q_\perp - r_\perp) dq_\perp \right|^2$$

Another advantage is that the eigenvalues are decreasing rapidly, so a series can be truncated at some point.

Algorithm description

After precomputing the eigenvalues and eigenvectors:

- Compute Fourier coefficients of the mask
- Compute sum of squared convolutions to get light intensity on coarse grid in the photoresist
- Interpolate the result to a dense grid

This approach is 5-15 times faster than the previous one, but loses the flexibility to arbitrarily change the model during computation.

GPU vs CPU Results, Algorithm 2 (small clip)

	CPU, 2.8 GHz, 1 core	Tesla C2070 (Fermi), 1.15 GHz, CUDA 4.2	
Copy the polygons	-	0.4 ms	
Computation of Fourier coefficients	120 ms	7.2 ms	16.5x
Coarse intensity computation	55 ms	1.2 ms	46x
Resample to a dense grid	26 ms	0.2 ms	~100x
GPU return intensity	-	2.2 ms	
Total	201 ms	11.2 ms	18x

Total GPU times include data transfers from CPU to GPU memory.

Last column is C2070 speedup with respect to CPU.

Size of the problem

Suppose you want to simulate how a full computer chip of size L×L would be printed on wafer.

- Fourier Coefficient computation ~ L^4
- Intensity computation ~ L^3 (approx.)
- Resampling ~ L^2

Chip size (L) can be up to 10 mm, feature size can be down to 10 nm. Each layer can be several 100's Gb in size.

Size of the problem

Suppose you want to simulate how a full computer chip of size $L \times L$ would be printed on wafer.

- Fourier Coefficient computation ~ L^4
- Intensity computation ~ L^3 (approx.)
- Resampling ~ L^2

Chip size (L) can be up to 10 mm, feature size can be down to 10 nm. Each layer can be several 100's Gb in size.

Solution: break the chip into many small tiles, carefully choose the tile size, deal with the boundary effects. Different tile sizes might be needed for CPU and GPU.

Full-Chip Results

- Full-chip verification
- Compute the intensity using second approach (model is not changing)
- Contour the intensity
- Check printability, critical dimensions, edge placement errors.
- 2 layers from 8.5mm × 8mm chip at 30nm node.
- 2 layers from 6.8mm × 7mm chip at 40nm node.
- Multiple focus and dose conditions.

1-core CPU at 2.8 GHz.
1-core CPU at 2.8 GHz and Tesla C2070 GPU at 1.15 GHz with CUDA 4.2.

GPU vs CPU Full-Chip Results

	30nm contact	40nm contact	30nm metal	40nm metal
Number of mask points	103M	245M	87M	299M
Number of target points	21M	25M	12M	18M
Focus/Dose conditions	2/1	2/2	2/3	3/2
CPU time, hh:mm	**31:29**	**49:50**	**69:15**	**319:06**
CPU + C2070 time, hh:mm	**2:54**	**4:35**	**5:02**	**23:09**
Speedup	**10.8x**	**10.9x**	**13.7x**	**13.8x**

GPU data transfers are asynchronous.
CPU and GPU are working together.

Conclusions

• Before implementing anything on a GPU check if the CPU algorithm can be improved (very often it can be!).
• CPU and GPU might have different optimal problem sizes.

In real-life applications, GPUs make speedups of 10-15x (with respect to 1 CPU) possible!

cadence™