Implementation of Reynolds Equation Solver on GPGPU for Gas Film Lubrication Problem

Ji-Hoon Kang, Hun Joo Myung, Kwang Jin Oh, Chan Yeol Park, Supercomputing Center, KISTI

Problem overview

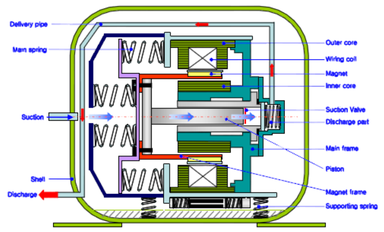
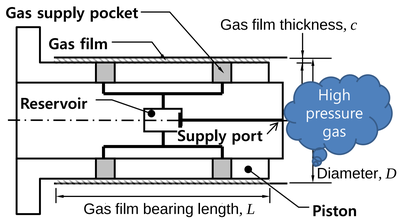


Figure of linear compressor
H. Lee, et al., ICSV15 (2008)



Schematic diagram of piston and gas film bearing

Gas film bearing in linear compressor

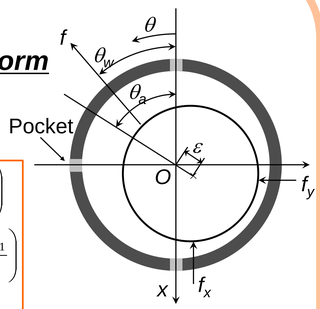
- One of key components in linear compressor
- Supporting the piston in reciprocating motion
- Having gas supply pocket/port for high pressure gas supply
- Generating supporting pressure using high pressure gas

Formulation

Reynolds equation and discretized form

$$\frac{\partial}{\partial Z}\left(PH^3\frac{\partial P}{\partial Z}\right)+\frac{\partial}{\partial \theta}\left(PH^3\frac{\partial P}{\partial \theta}\right)=\Lambda\frac{\partial PH}{\partial Z}+\Gamma\frac{\partial PH}{\partial \tau}$$

$$\frac{1}{\Delta Z}\left(P_{i+1/2,j}H^3_{i+1/2,j}\frac{P_{i+1,j}-P_{i,j}}{\Delta Z}-P_{i-1/2,j}H^3_{i-1/2,j}\frac{P_{i,j}-P_{i-1,j}}{\Delta Z}\right)$$
$$+\frac{1}{\Delta \theta}\left(P_{i,j+1/2}H^3_{i,j+1/2}\frac{P_{i,j+1}-P_{i,j}}{\Delta \theta}-P_{i,j-1/2}H^3_{i,j-1/2}\frac{P_{i,j}-P_{i,j-1}}{\Delta \theta}\right)$$
$$=\Lambda\frac{P_{i+1/2,j}H_{i+1/2,j}-P_{i-1/2,j}H_{i-1/2,j}}{\Delta Z}+\Gamma\frac{P^n_{i,j}-P^{n-1}_{i,j}}{\Delta \tau}$$

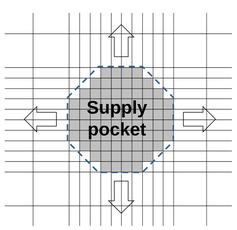


$$H=\frac{h}{c}=1+\varepsilon\cos(\theta-\theta_a)$$

Boundary condition

- Bearing ends $P(\theta,0)=P(\theta,L)=P_0$
- Periodicity $P(0,z)=P(2\pi,z)$
- Mass continuity at supply pocket

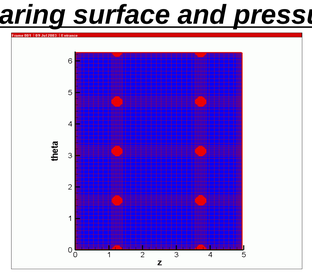
$$\dot{M}=\oint_{\partial S}\left(\Lambda PH\bar{z}_u-PH^3\nabla P\right)\cdot\vec{n}ds=A\left(P^2_{Supply}-P^2_{Pocket}\right)$$

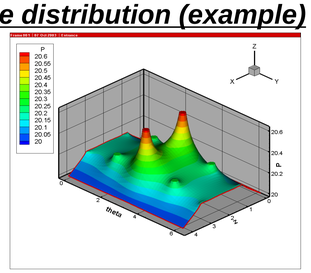


- Capillary restrictor

$$\dot{M}=\frac{12\mu R_g T}{c^3 p_a^2}\frac{\rho_s \pi d^4}{256\mu L}\left(P^2_{Supply}-P^2_{Pocket}\right)$$

Bearing surface and pressure distribution (example)





Implementation on GPGPU

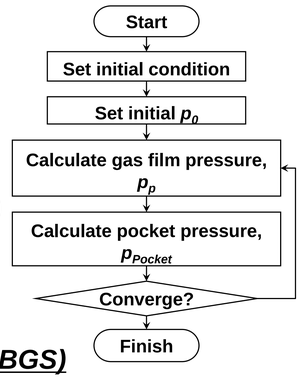
Gauss-Siedel iteration

- Reynolds eqn. for gas film pressure

$$a_P P_P=a_E P_E+a_W P_W+a_N P_N+a_S P_S+bP_P^{(n-1)}$$

- Mass continuity eqn. for pocket pressure

$$\sum_{faces}a_P P_{Pocket}=\sum_{faces}\left(a_E P_E+a_W P_W+a_N P_N+a_S P_S+bP^{(n-1)}_{Pocket}\right)+A_{Supply}P_{Supply}$$



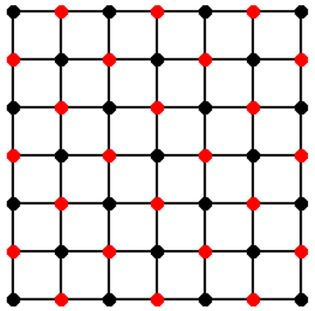
Red/Black Gauss-Siedel iteration (RBGS)

In iteration n:

- **1st pass**
  All red nodes are updated using old values of black nodes.
- **2nd pass**
  All black nodes are updated using updated values of red nodes

No data dependency among nodes of same colors

→ the nodes can be assigned to each thread and updated in parallel within each pass



- Assign each z node to each thread and each theta node to each block

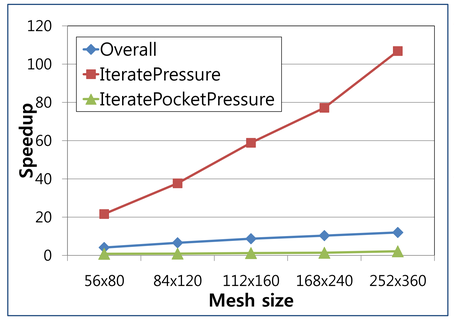
```
// Red/Black Gauss-Siedel iterative pressure solver structure
void Pressure_CUDA::iterate(){
  int redblack;
  unsigned int threads = NzNode;
  unsigned int blocks  = NthNode;
  do{
    redblack = 0;    //1st pass
    iteratePressure_kernel<<<blocks, threads, Mem_size>>>(d_MeshType,⋯,redblack);
    cutilSafeCall(cudaMemcpy(err,d_err, sizeof(double)*blocks, cudaMemcpyDeviceToHost));
    for(i=0;i<blocks;i++) { error+=err[i]; } // Error summation

    redblack = 1;    //2nd pass
    iteratePressure_kernel<<<blocks, threads, Mem_size>>>(d_MeshType,⋯,redblack);
    cutilSafeCall(cudaMemcpy(err,d_err, sizeof(double)*blocks, cudaMemcpyDeviceToHost));
    for(i=0;i<blocks;i++) { error+=err[i]; } // Error summation
  }while(error>criteria);
}
```

```
// Computational kernel structure
__global__ void iteratePressure_kernel(int *d_MeshType, ⋯, int redblack ) {
  int iIdx = threadIdx.x;
  int jIdx = blockIdx.x;
  if(((iIdx+jIdx)&1) == redblack && d_MeshType[jIdx*const_NzNode+iIdx]==-1) {
    // … Coefficient calculation
    aP=aE+aW+aN+aS+Fe-Fw+b;
    d_dP[jIdx*const_NzNode+iIdx]=(aE*PE+aW*PW+aN*PN+aS*PS+b*oldPP)/aP-PP;
    d_P[jIdx*const_NzNode+iIdx] +=0.4*d_dP[jIdx*const_NzNode+ iIdx];
    // … Error calculation
  }
  double_reduction( error,  d_err);
}
```

Results

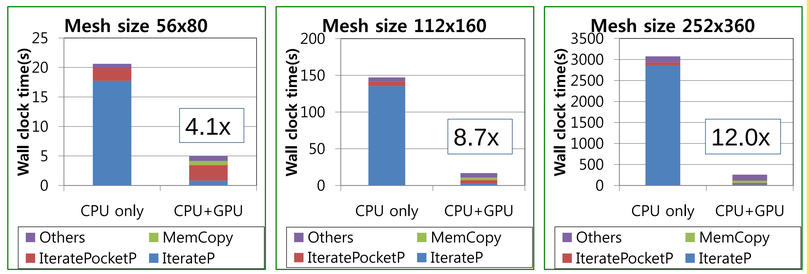
GPU speedup



CPU : AMD Llano A8-3850, 2.9GHz
GPU : GTX 580, 795MHz, 512 threads

- Overall speedup
  - Increase with mesh size
  - Achieved 12X at 252x360 (Limit: 21X by Amahl's law)
  - Org. code serial part: 3~4.5%
- IteratePressure
  - Excellent parallel speedup
  - Achieved 106X at 252x360 (Theoretical limit: 140X from clock speed))
- IteratePocketPressure
  - Poor parallel speedup
  - Small and fixed pocket mesh size

Wall clock time comparison



Mesh	CPU/GPU	Loop (s)	IterateP(s)	ItrPocketP(s)	MemCp(s)	Others(s)
56x80	CPU	20.6	17.7 (86%)	2.2 (11%)	-	0.7 (3%)
	CPU+GPU	5.0	0.8 (16%)	2.6 (52%)	0.8(15%)	0.8 (16%)
112x160	CPU	147.2	135.4 (92%)	6.0 (4%)	-	5.7 (4%)
	CPU+GPU	16.9	2.3 (14%)	4.9 (29%)	3.7(22%)	6.0 (36%)
252x360	CPU	3074.5	2861.6(93%)	72.2 (2%)	-	140.6 (5%)
	CPU+GPU	256.2	26.8 (10%)	33.4 (13%)	54.1(21%)	141.9 (55%)

- A small serial part becomes a critical bottleneck as the problem size gets bigger and GPU speedup increases.
- Not only core calculation part but also general calculation part needs to be parallelized for better speedup.

Conclusion & Future work

- Implemented a Reynolds equation solver on GPGPU for gas film lubrication problem.
- Achieved 106x speedup for core calculation part and overall 12x speedup (DP) using RBGS, relative to 1core of AMD Llano A8-3850.
- A tiny serial part can become a critical bottleneck as the problem size gets bigger and GPU efficiency increases.
- Future work will include the development of general gas film analysis solver and the development of parallelization scheme for remaining serial part, such as integration, error check, and et al.