GPU TECHNOLOGY CONFERENCE

▸ TRACK 3 CUDA / COMPUTER VISION

13:10~13:40	KAIST 유동근 연구원	AttentionNet for Accurate Localization and Detection of Objects
13:40~14:10	한국생산기술연구원 진경찬 그룹장	CUDA를 이용한 3D IC 패키지 검사
14:10~14:40	이화여자대학교 김성기 박사	모바일 GPGPU 어플리케이션을 위한 GPU DVFS 알고리즘
14:40~15:00		휴식 및 전시 관람
15:00~15:30	MidasIT 이노훈 주임연구원	Midas NFX의 GPU 개발 현황과 향후 로드맵
15:30~16:00	IBM 허욱 실장	IBM Power와 NVIDIA GPU가 그리는 차세대 컴퓨팅 솔루션
16:00~16:30	Mellanox Korea 정연구 기술이사	100Gb/s EDR InfiniBand & GPUDirect RDMA
16:30~17:00		폐회사 및 경품 추첨

GPU DEVELOPMENT & FUTURE PLAN OF MIDAS NFX

September 22 2015

Noh-hoon Lee lnh0702@midasit.com

SOFTWARE ENGINEER / CFD DEVELOPMENT TEAM

MIDASIT

CONTENTS

1. Introduction to MIDASIT
2. Computing Procedure
3. GPU Equation Solver
4. Future Plan

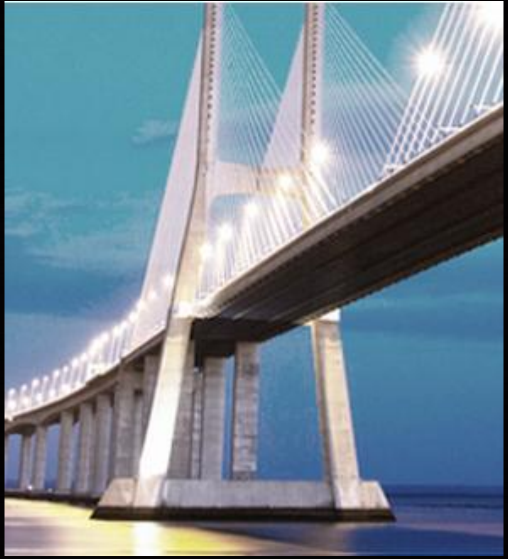
GPU TECHNOLOGY CONFERENCE

INTRODUCTION TO MIDASIT

MIDASIT

- We are worldwide developer & provider of CAE(Computer Aided Engineering) software

Architectural Engineering



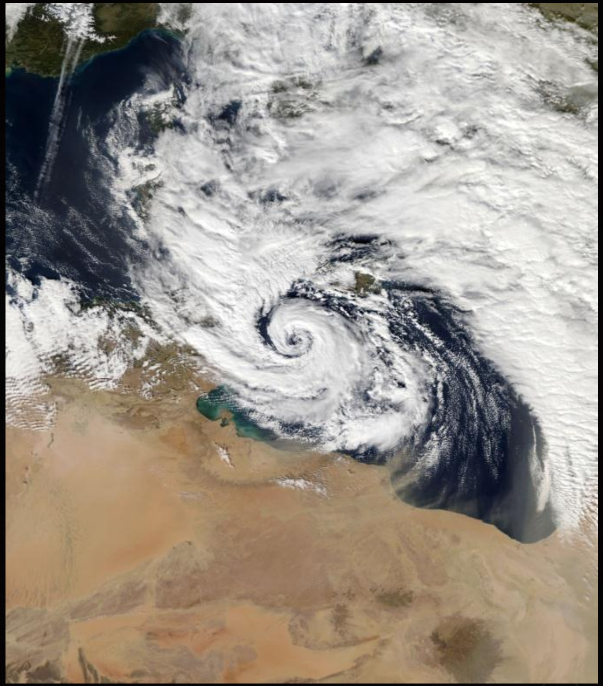
Civil Engineering

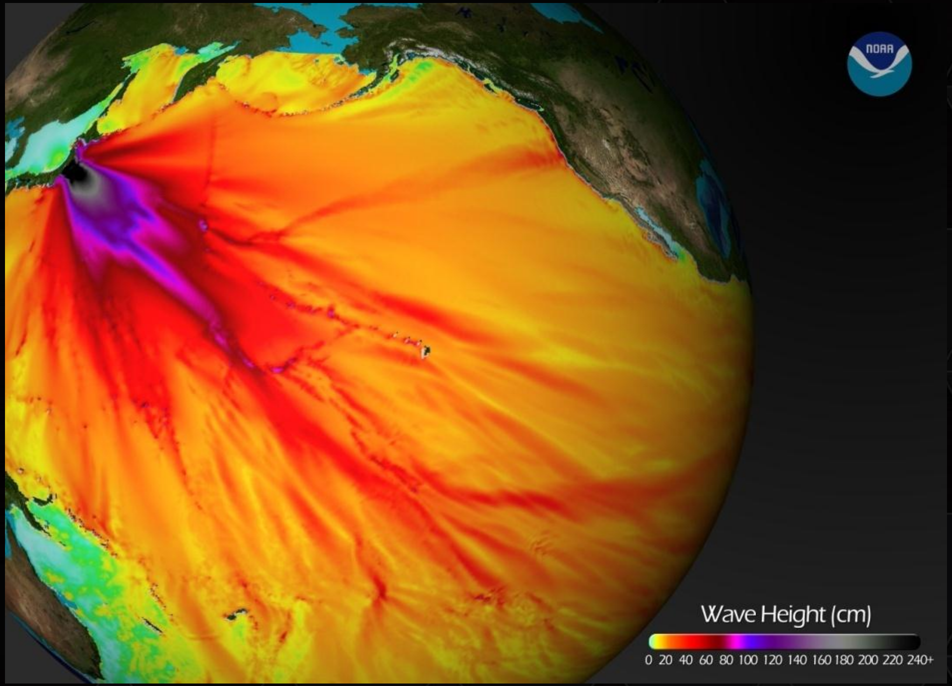
Geotechnical Engineering

Mechanical Engineering

STRUCTURAL ANALYSIS

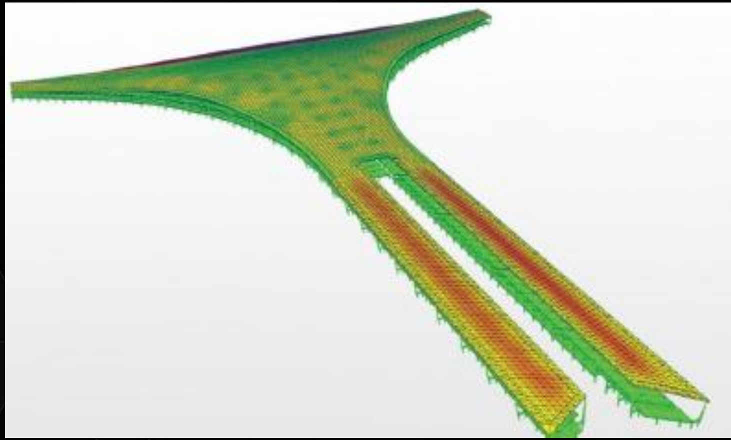
COMPUTATIONAL FLUID DYNAMICS(CFD)





BUSINESS AREA

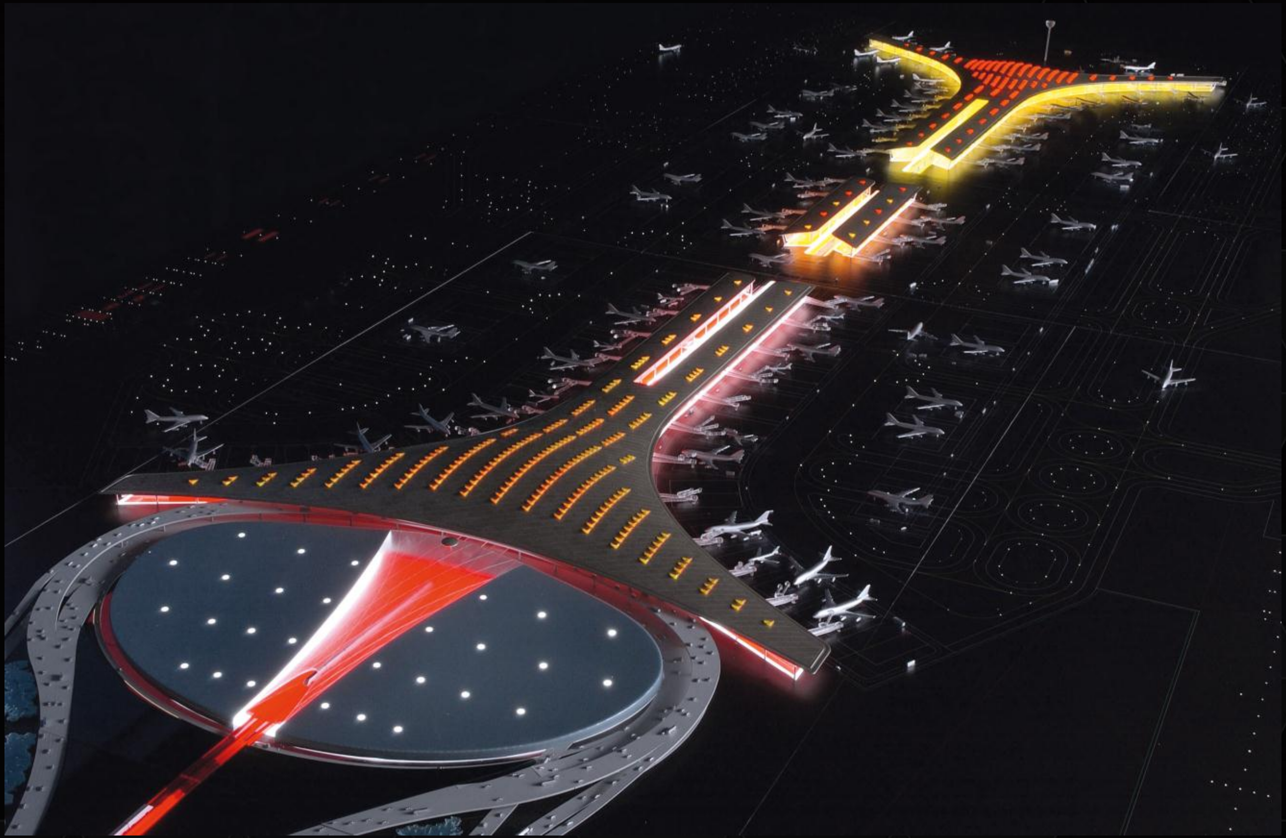
Architectural Engineering



Beijing Capital International Airport

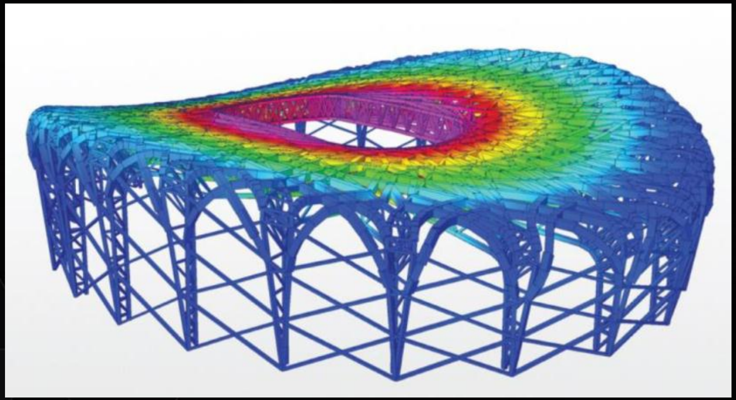
Beijing, China

midas Gen



BUSINESS AREA

Architectural Engineering



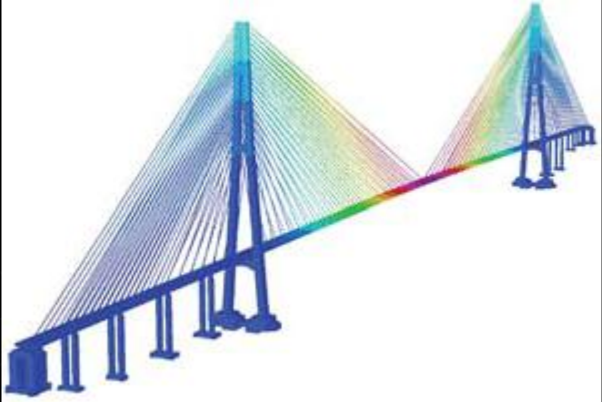
Beijing National Stadium

Beijing, China

midas Gen

BUSINESS AREA

Civil Engineering



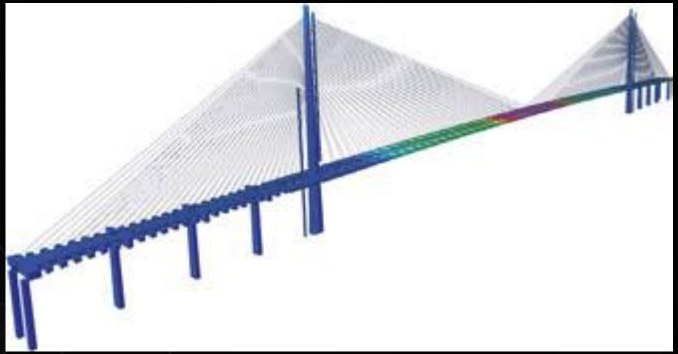
Russky Island Bridge

Vladivostok, Russia

midas Civil

BUSINESS AREA

Civil Engineering



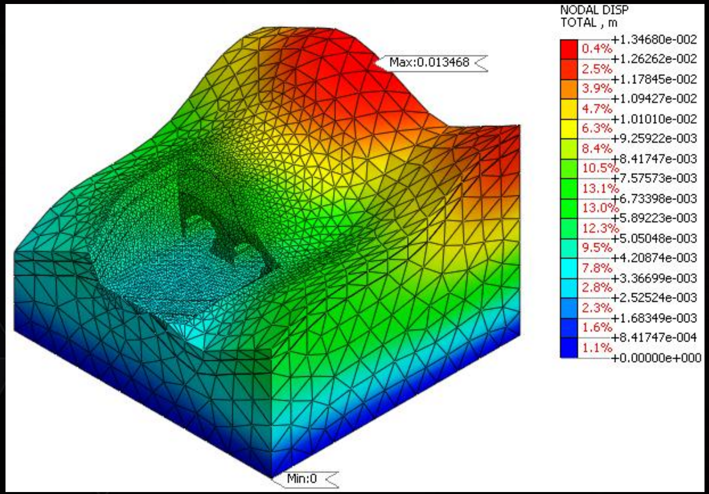
Stonecutters Bridge

Hong Kong, Chinna

midas Civil

BUSINESS AREA

Geotechnical Engineering

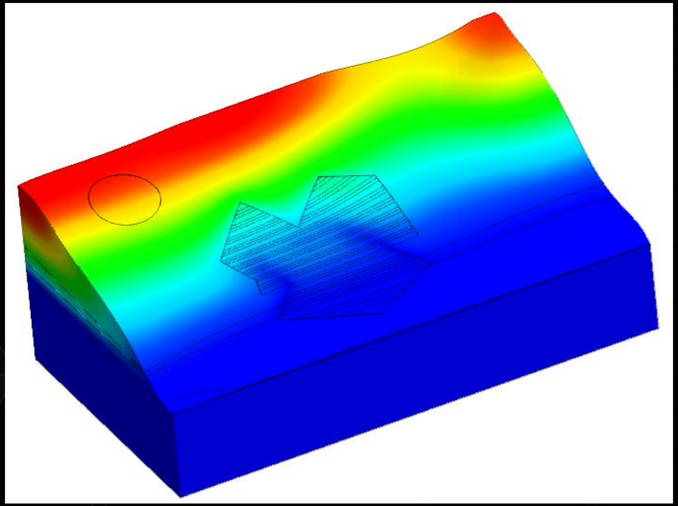


Subway tunnel Analysis

midas GTS NX

BUSINESS AREA

Geotechnical Engineering

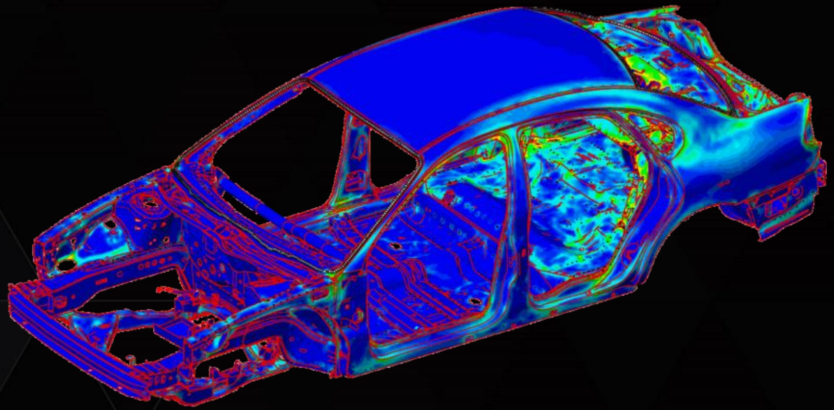


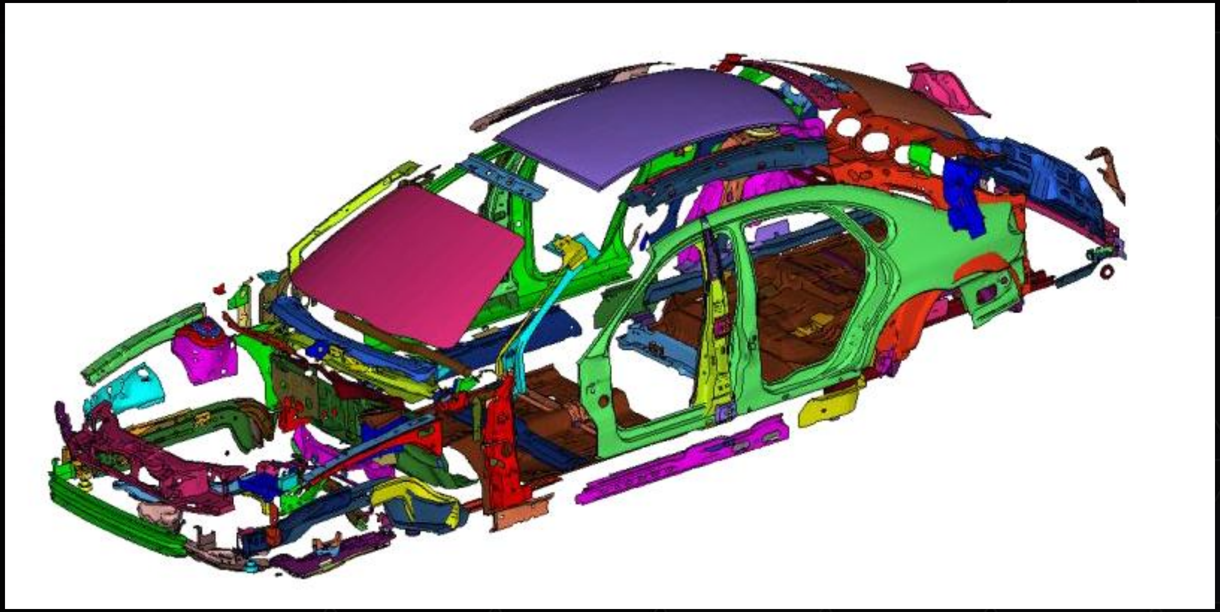
Foundation / Stage Analysis

midas GTS NX

BUSINESS AREA

Mechanical Engineering



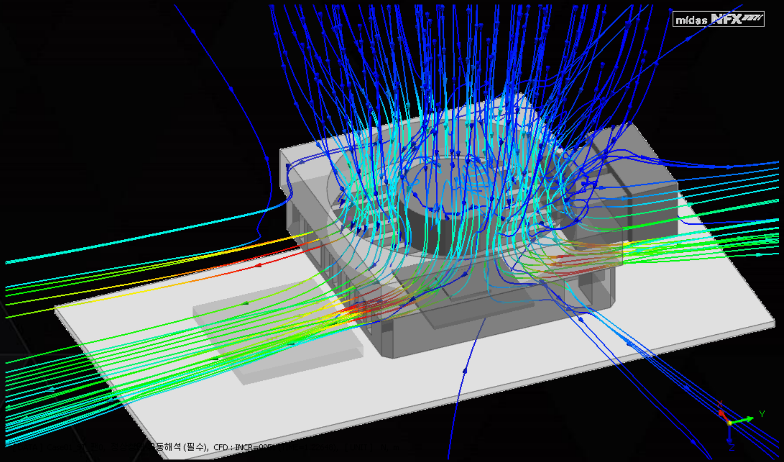


Modal / Static Analysis

midas NFX(structure)

BUSINESS AREA

Mechanical Engineering



Thermal Flow Analysis

midas NFX(CFD)

COMPUTING PROCEDURE

COMPUTING PROCEDURE

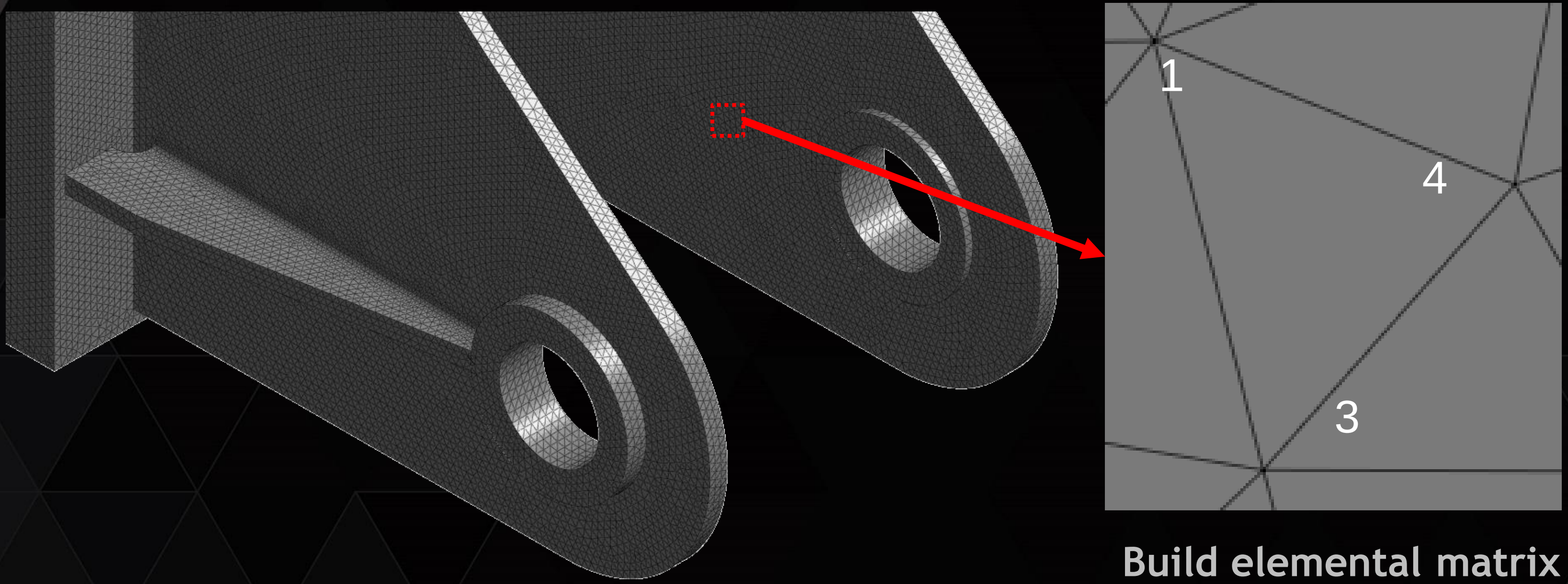
ASSEMBLE MATRIX
Build system of linear equations

← Equations
(Structural, Fluid, Chemical, Electric, etc)

SOLVE LINEAR EQUATION
Direct Solver(Structural Analysis) / Iterative Solver(CFD)

COMPUTING PROCEDURE

Mesh(Computing Domain)



Build elemental matrix

$$
\begin{bmatrix} A_{11} & & A_{13} & A_{14} \\ & & & \\ A_{31} & & A_{33} & A_{34} \\ A_{41} & & A_{43} & A_{44} \end{bmatrix}
\begin{bmatrix} X_1 \\ \\ X_3 \\ X_4 \end{bmatrix}
=
\begin{bmatrix} b_1 \\ \\ b_3 \\ b_4 \end{bmatrix}
$$

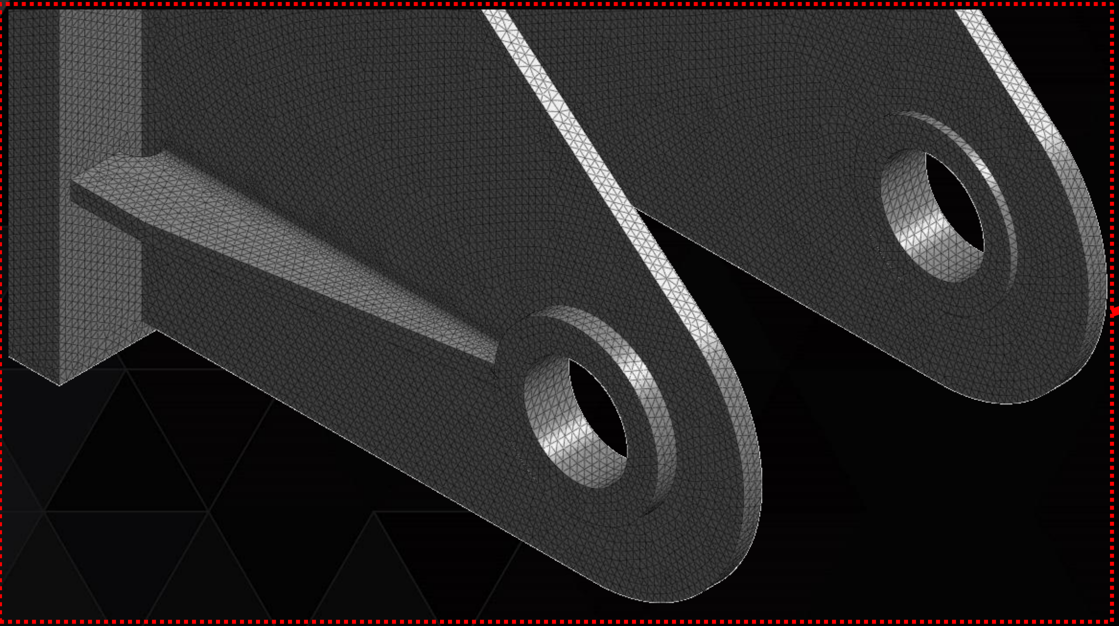
$$A_{11}x_1 + A_{13}x_3 + A_{14}x_4 = b_1$$

$$A_{31}x_1 + A_{33}x_3 + A_{34}x_4 = b_3$$

$$A_{41}x_1 + A_{43}x_3 + A_{44}x_4 = b_4$$

COMPUTING PROCEDURE

Mesh(Computing Domain)



$$A_{11}x_1 + A_{12}x_2 + A_{13}x_3 ...... = b_1$$
$$A_{21}x_1 + A_{22}x_2 + A_{23}x_3 ...... = b_2$$
$$A_{31}x_1 + A_{32}x_2 + A_{33}x_3 ...... = b_3$$
$$......$$

Build system matrix

$$Ax = b$$

Now we have linear equations!

COMPUTING PROCEDURE

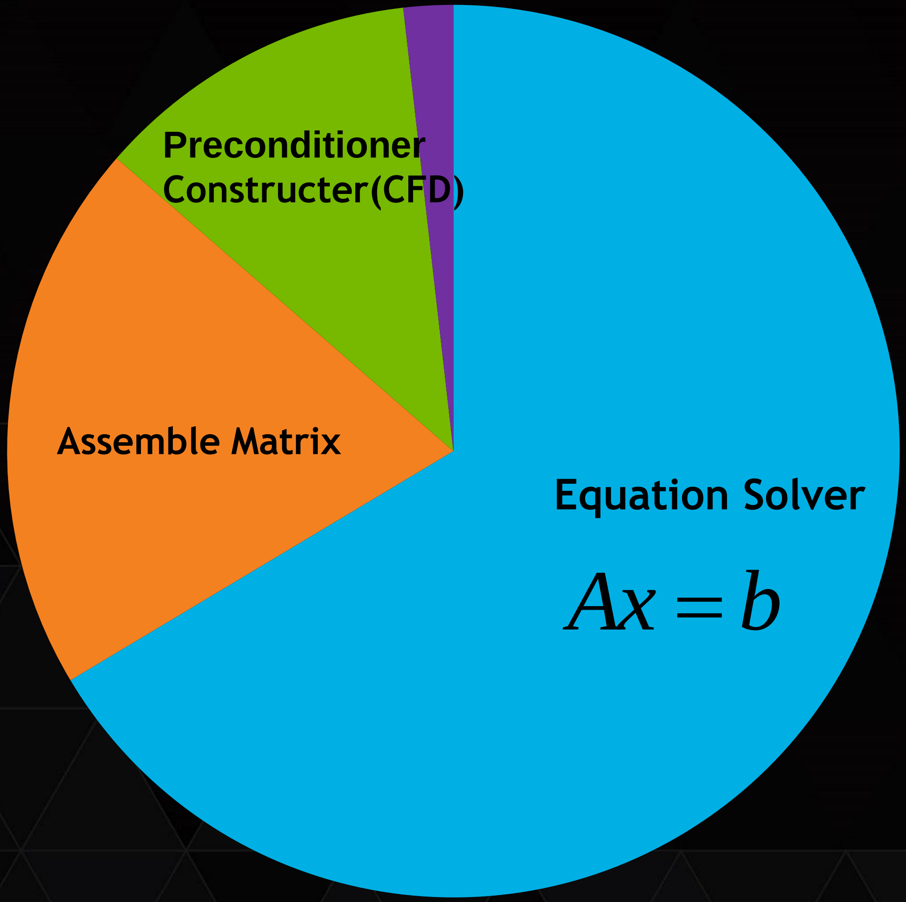
ASSEMBLE MATRIX
Build system of linear equations

SOLVE LINEAR EQUATION
Direct Solver(Structural Analysis) / Iterative Solver(CFD)

Solve linear equations $Ax = b$

GPU EQUATION SOLVER

COMPUTATION TIME



Equation Solver : 60~90% of total computing time

(accelerate this first)

EQUATION SOLVER

Structural Analysis :

Direct Solver(Multi Frontal Solver(MFS))

Computational Fluid Dynamics :

Iterative Solver

Preconditioner(ILU(n), AMG, etc)

DIRECT SOLVER

$$x + y = 2$$

$$2x - y = 1$$

$$(x + y) + (2x - y) = 2 + 1$$

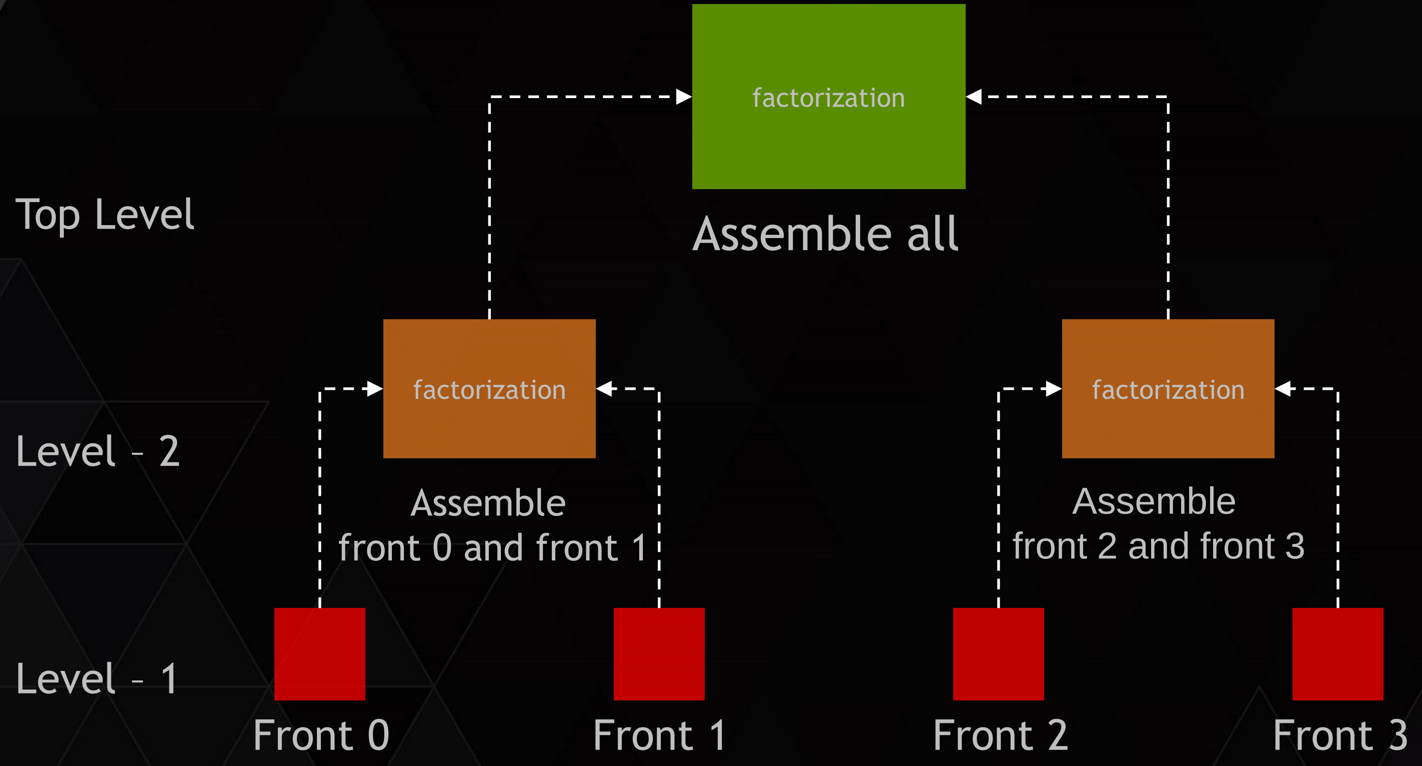
$$3x = 3$$

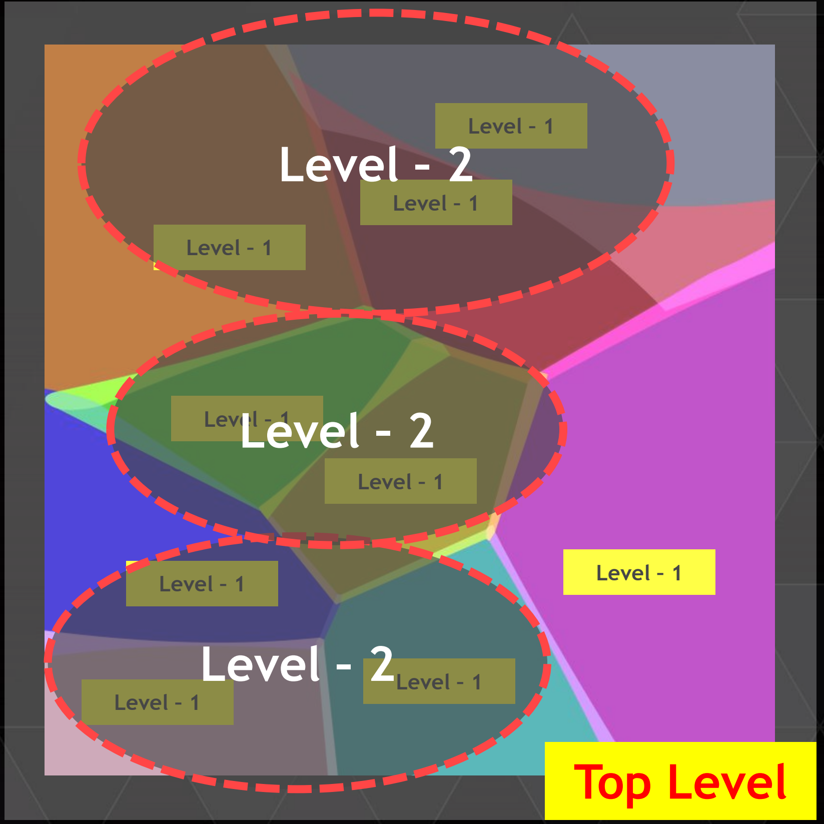
$$x = 1$$

$$1 + y = 2$$

$$y = 1$$

MULTI FRONTAL SOLVER(MFS)





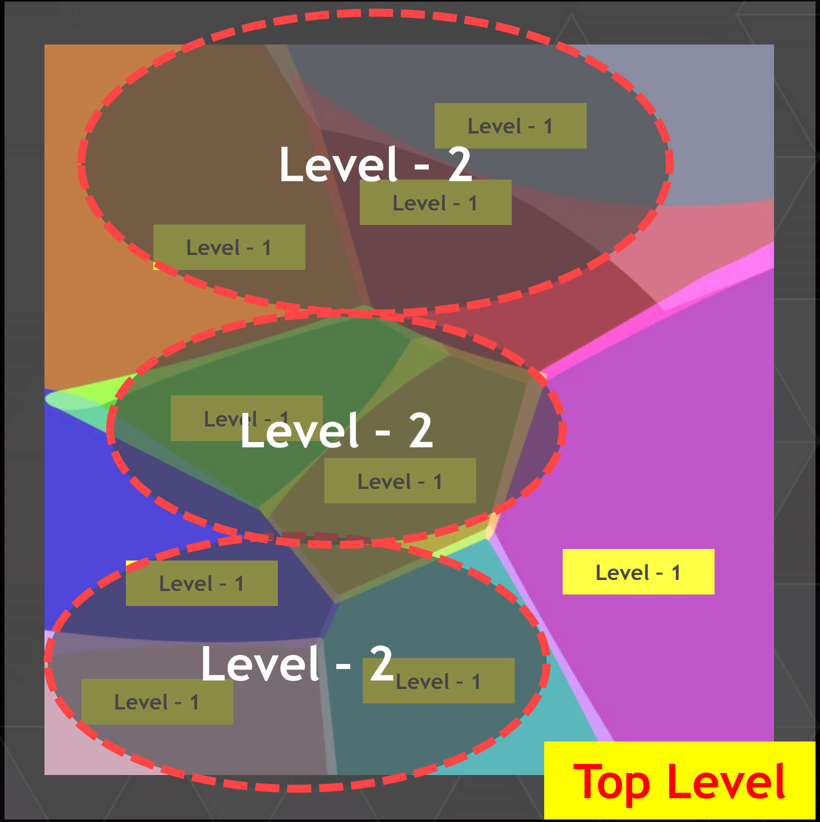
GPU FACTORIZATION

Is leading dimension of frontal matrix
is larger than 1024(Fermi) or 2048(Kepler)?

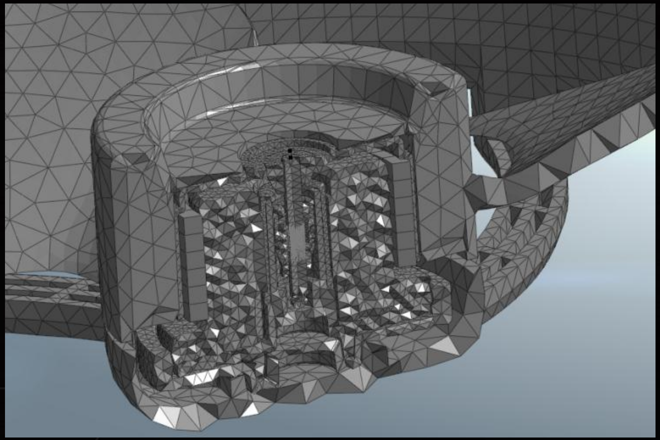
no → **CPU Factorization**
POPTRF / SYPTRF / GEPTRF

yes → **GPU Factorization**
GPU_POPTRF / GPU_SYPTRF / GPU_GEPTRF

CUBLAS is used to apply GPU computing



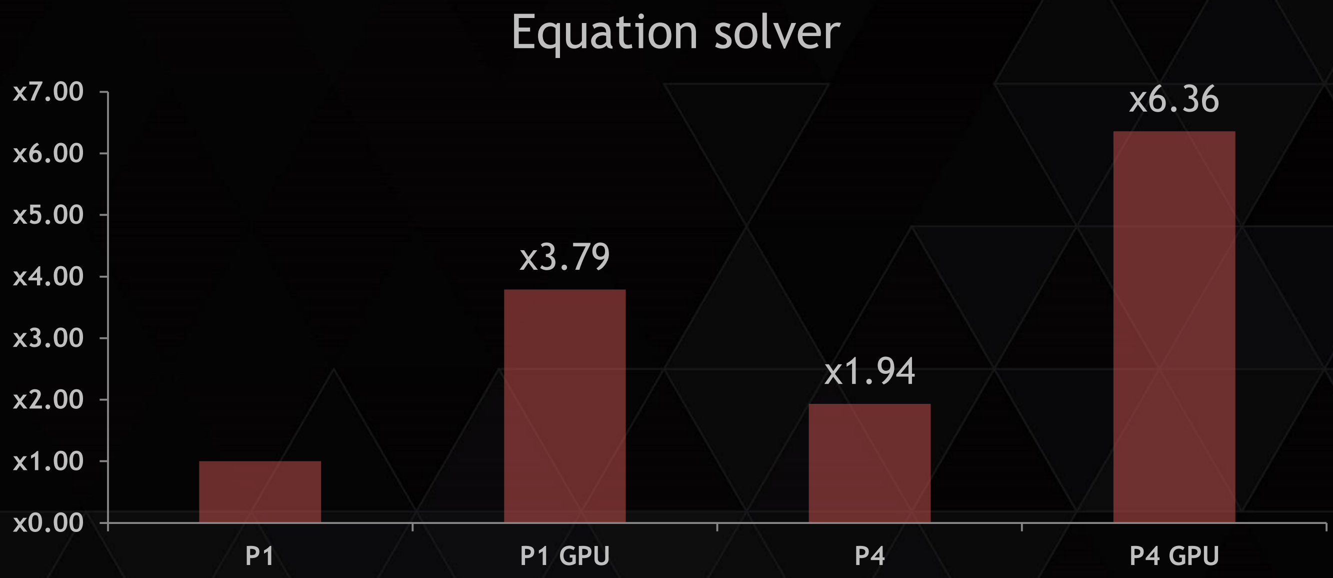
GPU SPEEDUP(STRUCTURAL ANALYSIS)



Normal Mode - Fan
(LDLT - GPU_SYPTRF)

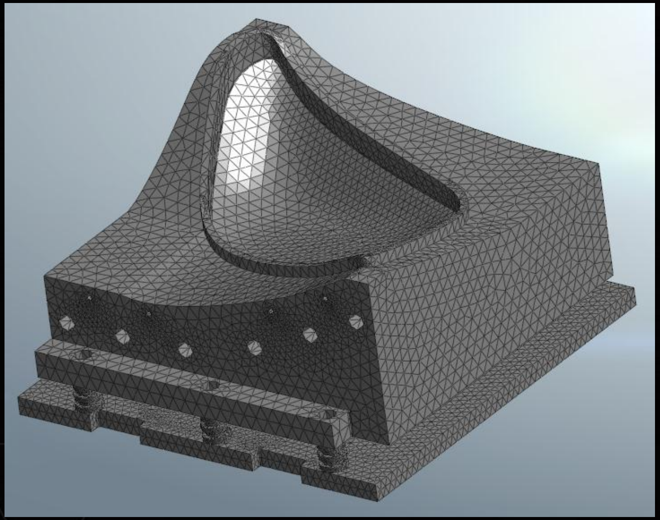
	Equation solver
1 thread	617.246 sec
1 thread 1 GPU	162.942 sec
4 threads	318.879 sec
4 threads 1 GPU	97.095 sec

Specifications	
CPU	2x Intel Xeon 2.97 GHz (8 cores)
RAM	48 Gbyte
GPU	Tesla C2075 1ea



GPU TECHNOLOGY CONFERENCE

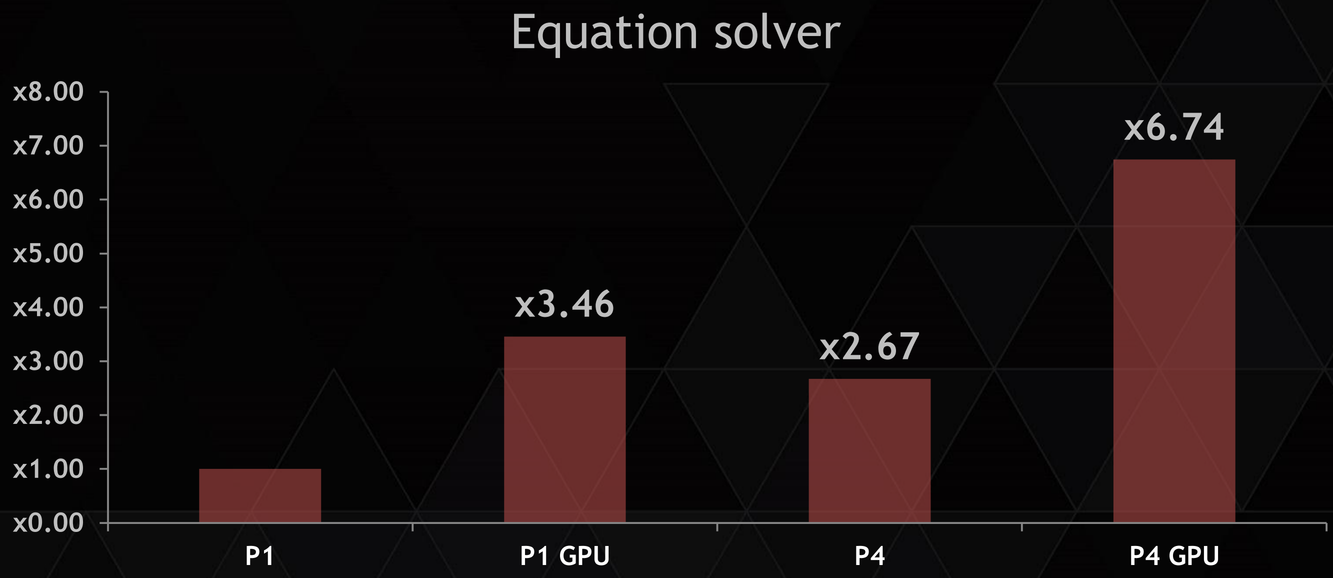
GPU SPEEDUP (STRUCTURAL ANALYSIS)



Linear Static - casting tool
(LLT - DPOPTRF)

	Specifications
CPU	2x Intel Xeon 2.97 GHz (8 cores)
RAM	48 Gbyte
GPU	Tesla C2075 1ea

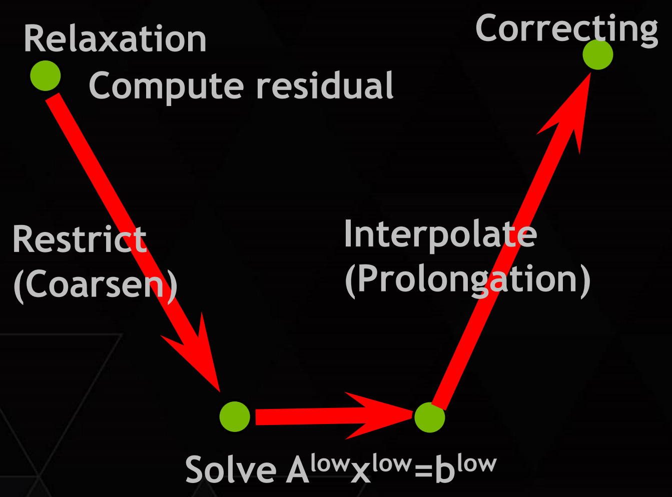
	Equation solver
1 thread	67.736 sec
1 thread 1 GPU	19.578 sec
4 threads	25.365 sec
4 threads 1 GPU	10.047 sec



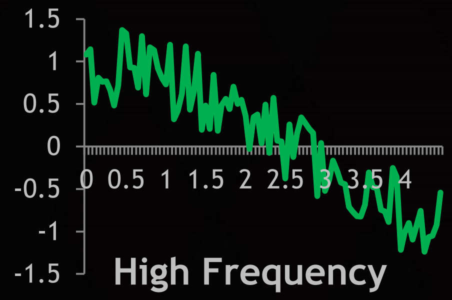
ITERATIVE SOLVER

Iterative Solvers	Preconditioners
Conjugate Gradient	
Bi Conjugate Gradient	
Stabilized Conjugate Gradient	
Stabilized Conjugate Gradient(2)	
Quasi Minimal Residual Method	Incomplete LU0
Deflated Conjugate Gradient	Incomplete LU1
Deflated Bi Conjugate Gradient	Algebraic Multi-grid(AMG)
Deflated Stabilized Conjugate Gradient	
Deflated Stabilized Conjugate Gradient(2)	
Deflated Quasi Minimal Residual Method	

GPU SOLVER (WITH ALGEBRAIC MULTIGRID PRECONDITIONER)



GPU Iterative Solvers	Preconditioners
GPU Conjugate Gradient GPU Stabilized Conjugate Gradient GPU Stabilized Conjugate Gradient(2) GPU Transpose Free Quasi Minimal Residual Method	GPU Algebraic Multi-grid(GPUAMG)



Many relaxation schemes have the smoothing property, where oscillatory modes of the error are eliminated effectively, but smooth modes are damped very slowly.
Smooth error can be represented on a coarse grid. Therefore, low frequency error is more effectively damped on coarse grid and high frequency error is effectively damped on a fine grid.

GPU TECHNOLOGY CONFERENCE

GPU ITERATIVE SOLVER

Conjugate gradient method

Solve $Ax = b$

$r = b - Ax$

$p_0 = r_0$

repeat

$$\alpha_k = \frac{dotproduct(r_k, r_k)}{dotproduct(p_k, Ap_k)}$$

$x_{k+1} = x_k + \alpha_k p_k$

$r_{k+1} = r_k - \alpha_k Ap_k$

if r_{k+1} is small enough, break

$$\beta_k = \frac{dotproduct(r_{k+1}, r_{k+1})}{dotproduct(r_k, r_k)}$$

$p_{k+1} = r_{k+1} + \beta_k p_k$

k++

end repeat

Used algorithms

CPU	GPU
CSRMV	gpu CSRMV
dot product	cublas dot product
axpy	gpu axpy
axpy2	gpu axpy2

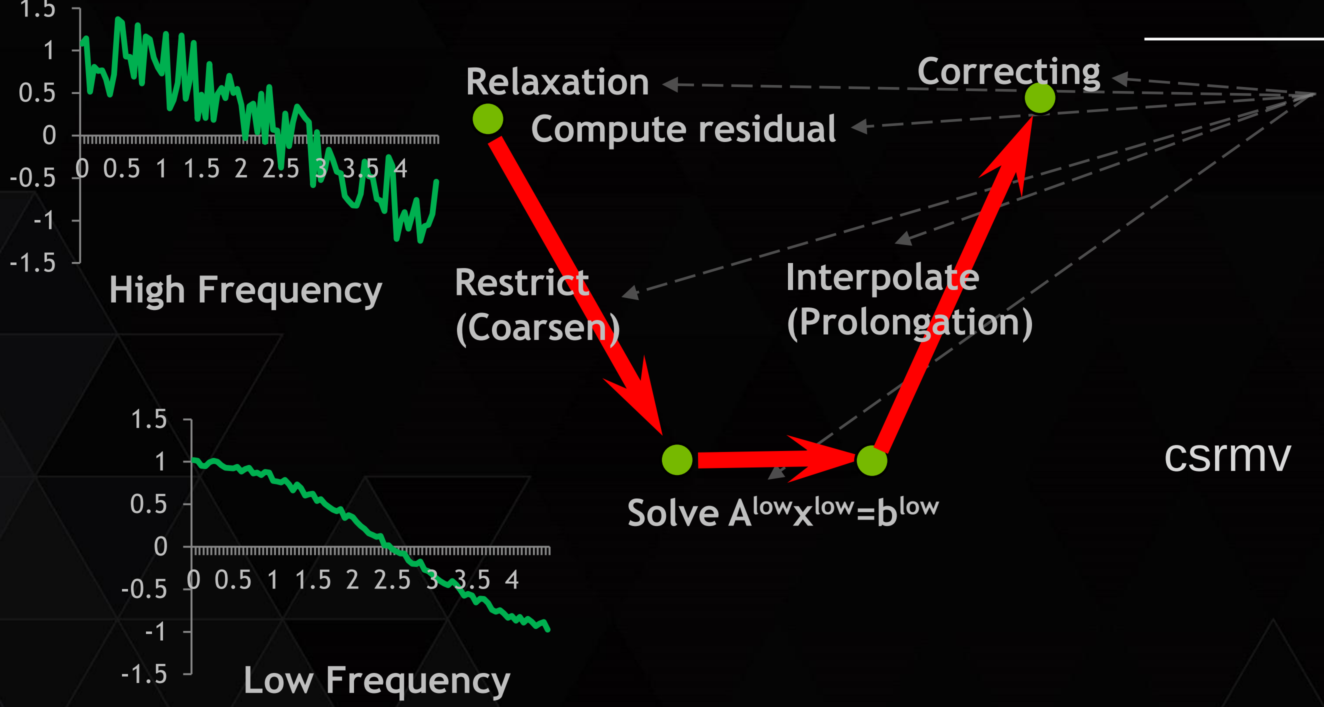
csrmv : sparse matrix vector multiplication

axpy : $x = x + \alpha y$

axpy2 : $x = \alpha x + y$

GPU AMG

Algebraic Multi-grid



Used algorithm

CPU	GPU
CSRMV	gpu CSRMV

csrmv : sparse matrix vector multiplication

ACCELERATE CSRMV

Compressed Sparse Row(CSR) format

2	-1	0	0
-1	2	-1	0
0	-1	2	-1
0	0	-1	2

Values	2	-1	-1	2	-1	-1	2	-1	-1	2
Row pointer	0	2	5	8	10					
Column index	0	1	0	1	2	1	2	3	2	3

ACCELERATE CSRMV

```
__global__ void csrmv_naive(const int MATRIX_SIZE, const double* val, const int* colind, const int* rowptr, double* X, double* res)
{
	signed int i, i_begin, i_end, row;
	double result;

	row = (blockIdx.y*gridDim.x+blockIdx.x)*blockDim.x + threadIdx.x;

	if (row<MATRIX_SIZE){
		i_begin = rowptr[row];
		i_end  =  rowptr[row+1];

		result =0;
		for (i = i_begin; i < i_end; i++){
			result += val[i] * X[colind[i]];
		}
		res[row] = result;
	}
}
```

One row per one thread

Too many branches and warp divergence.

ACCELERATE CSRMV

```
__global__ void csrmv_vectorize( const int N ,const int* rowptr ,const int* colind ,const double * val ,const double * x, double * y)
{
        int row, row_start, row_end, jj;
        __shared__ double y_aux[256];
        int thread_id = (blockIdx.y*gridDim.x+blockIdx.x)*blockDim.x + threadIdx.x ; // global thread index
        int warp_id = thread_id / 32; // global warp index
        int lane = thread_id & (32 - 1); // thread index within the warp
        row = warp_id ;
        if ( row < N ){
                row_start = rowptr [row ];
                row_end = rowptr [ row +1];
                // compute running sum per thread
                y_aux [ threadIdx.x ] = 0.0;
                for ( jj = row_start + lane ; jj < row_end ; jj += 32){
                        y_aux [ threadIdx.x ] += val [jj] * x[ colind [jj ]];
                }
                // parallel reduction in shared memory
                if ( lane < 16) y_aux [ threadIdx.x ] += y_aux [ threadIdx.x + 16];
                if ( lane < 8) y_aux [ threadIdx.x ] += y_aux [ threadIdx.x + 8];
                if ( lane < 4) y_aux [ threadIdx.x ] += y_aux [ threadIdx.x + 4];
                if ( lane < 2) y_aux [ threadIdx.x ] += y_aux [ threadIdx.x + 2];

                // first thread writes the result
                if ( lane == 0)     y[ row ] = y_aux [threadIdx.x]+y_aux [threadIdx.x+1];
        }
}
```

One row per one warp

Reference : Efficient Sparse Matrix-Vector Multiplication on CUDA, Nathan Bell and Michael Garland, 2008

ACCELERATE CSRMV

```
__global__ void csrmv_kernel_fixed_rule(const double *values, const int *rowPtrs, const int *colIdxs, const double *x, double *y,
                                         const int dimRow, const int repeat, const int coop)
{
  //reference : Design of efficient sparse matrix-vector multiplication for Fermi GPUs.
  extern __shared__ volatile double sdata[];
  int i = (repeat * blockIdx.x * blockDim.x + threadIdx.x) / coop;
  int coopIdx = threadIdx.x%coop;
  int start,end,j;
  unsigned int s;
  for(int r = 0; r < repeat; r++){
    sdata[threadIdx.x] = 0.0;
    if(i < dimRow){ // mat x vec
      start = rowPtrs[i];
      end   = rowPtrs[i+1];
      for(j = start + coopIdx; j < end; j += coop){
        sdata[threadIdx.x] += values[j] * x[colIdxs[j]];
      }
      // do reduction
      for(s = coop / 2; s > 0; s >>= 1){
        if(coopIdx < s) sdata[threadIdx.x] += sdata[threadIdx.x + s];
      }
      if(coopIdx == 0) y[i] = sdata[threadIdx.x] + sdata[threadIdx.x + 1];;
      i += blockDim.x / coop;
    }
  }
}
```

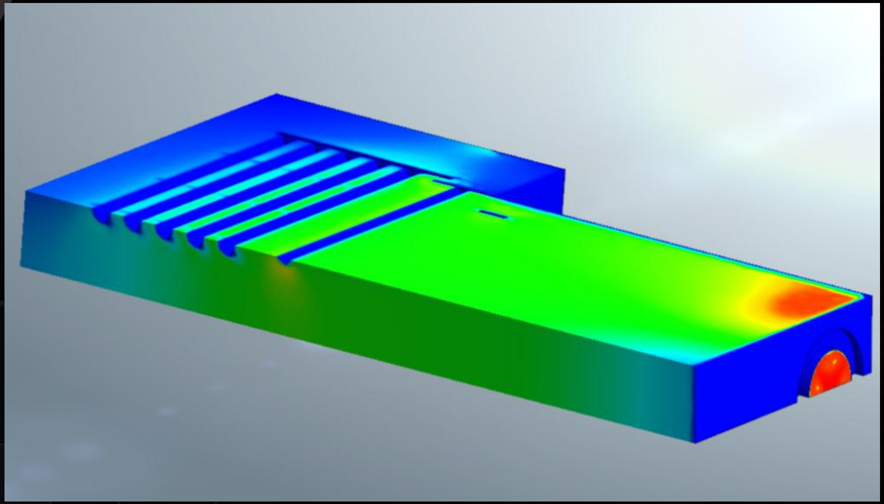
n rows per one warp

Enhancing memory bandwidth. (increase cache efficiency)

Reference :
Efficient sparse matrix-vector multiplication on cache-based GPUs,
Istvan Reguly and Mike Giles, 2012

GPU TECHNOLOGY CONFERENCE

GPU SPEEDUP (COMPUTATIONAL FLUID DYNAMICS)

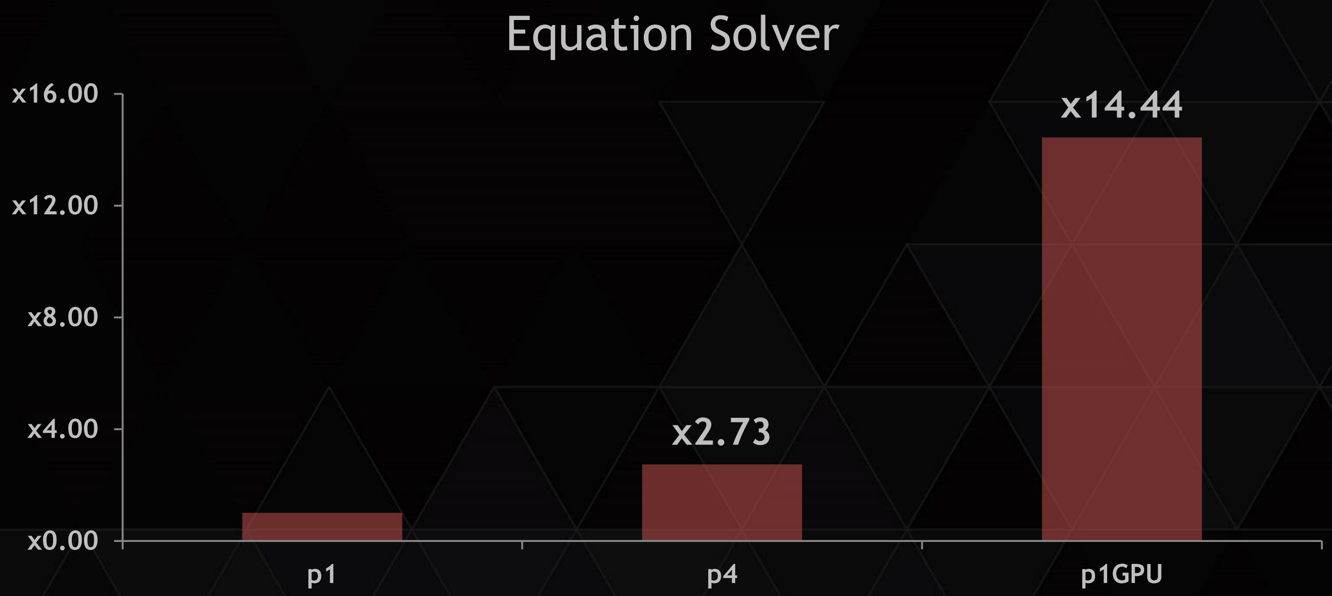


Turbulent Flow Analysis

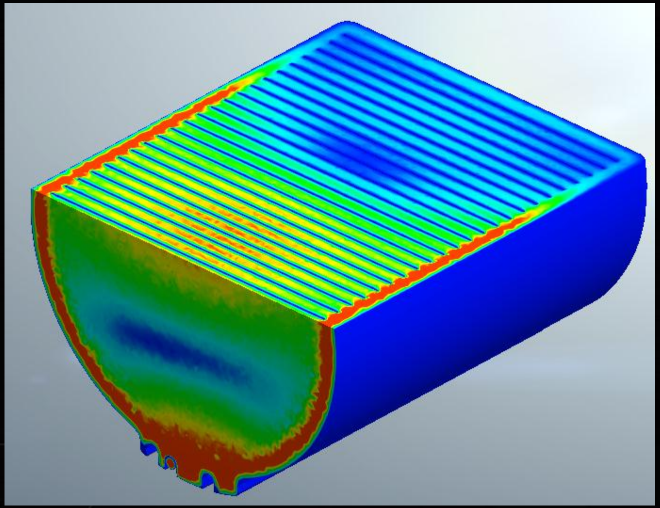
ELEMENTS	4,286,496
NODES	857,547

	Specifications
CPU	2x Intel Xeon 2.97 GHz (8 cores)
RAM	48 Gbyte
GPU	GeForce GTX TITAN(DP on)
Solver	Stabilized Bi Conjugate Gradient(2), AMG

	Equation solver
1 thread	4,233.8 sec
4 threads	1,548.4 sec
1 thread 1GPU	293.2 sec



GPU SPEEDUP (COMPUTATIONAL FLUID DYNAMICS)

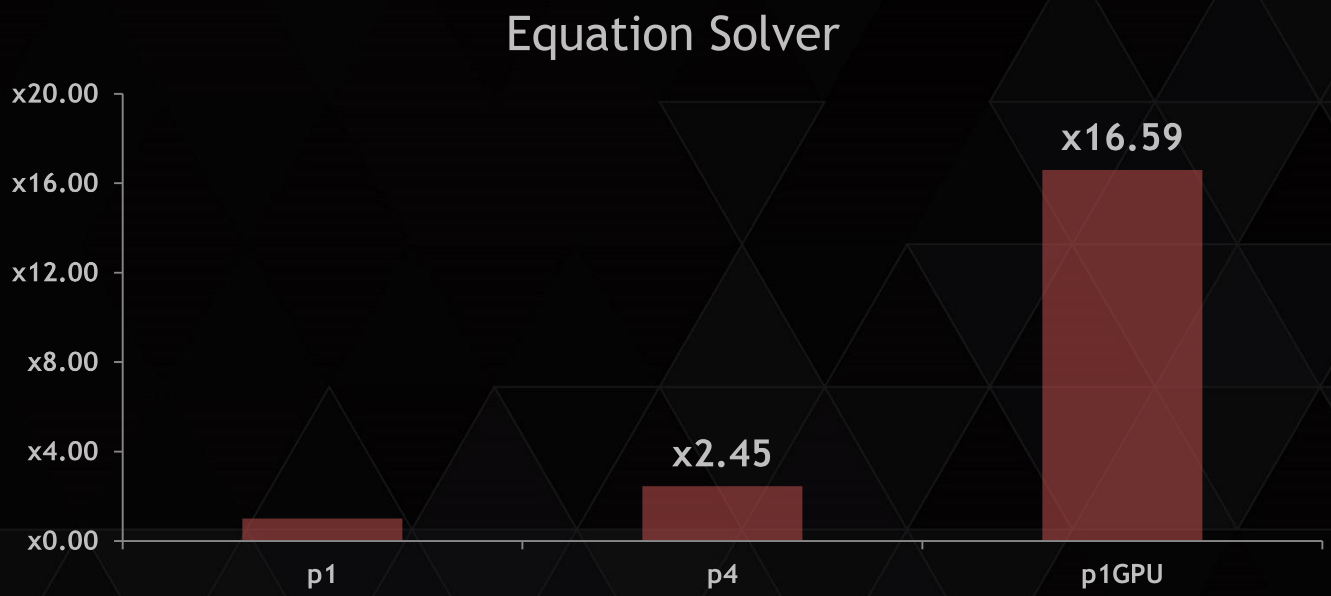


Turbulent Flow / Temperature Analysis

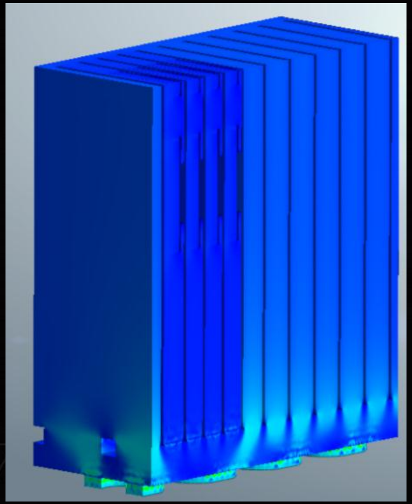
ELEMENTS	5,785,184
NODES	1,029,530

	Specifications
CPU	2x Intel Xeon 2.97 GHz (8 cores)
RAM	48 Gbyte
GPU	GeForce GTX TITAN(DP on)
Solver	Stabilized Bi Conjugate Gradient(2), AMG

	Equation solver
1 thread	21,661.3 sec
4 threads	8,855.2 sec
1 thread 1GPU	1,305.5 sec



GPU SPEEDUP (COMPUTATIONAL FLUID DYNAMICS)

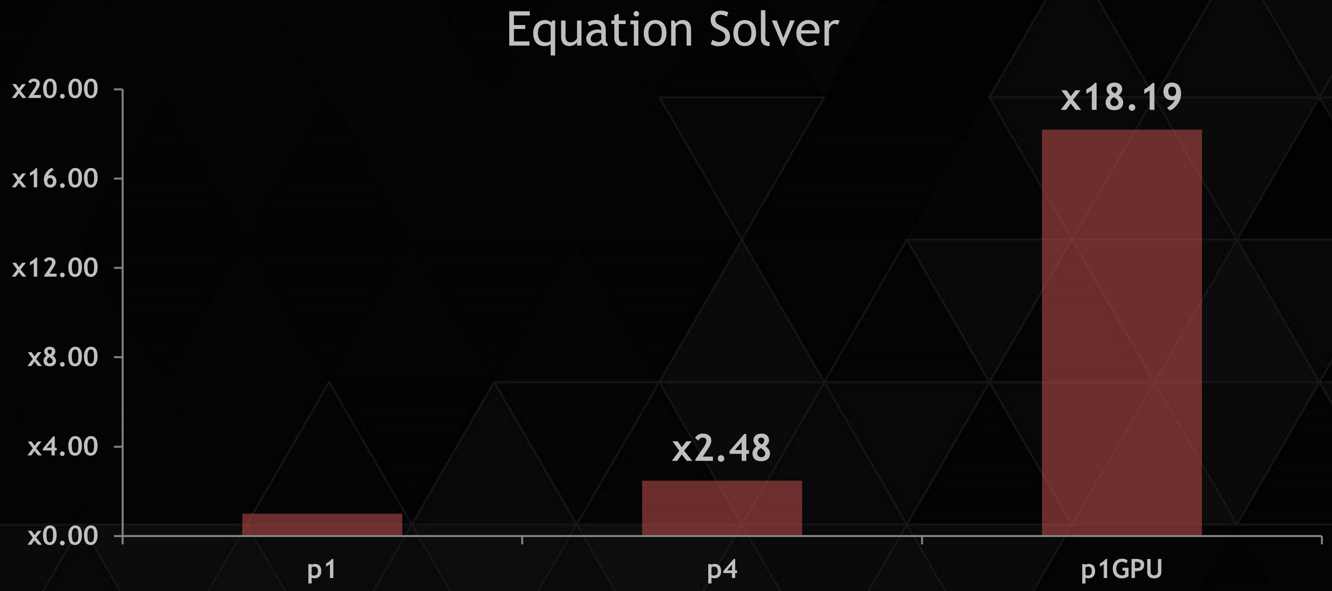


Turbulent Flow

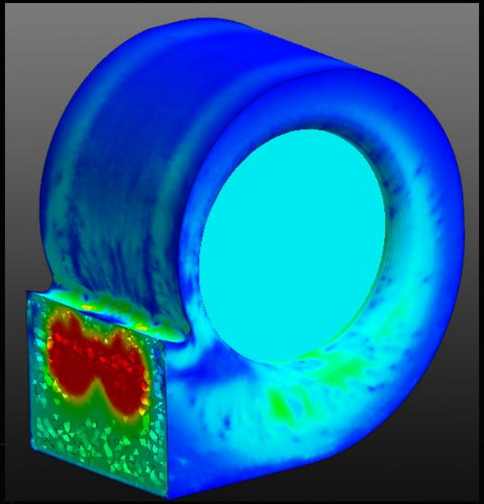
ELEMENTS	7,223,319
NODES	1,370,312

	Specifications
CPU	2x Intel Xeon 2.97 GHz (8 cores)
RAM	48 Gbyte
GPU	GeForce GTX TITAN(DP on)
Solver	Stabilized Bi Conjugate Gradient(2), AMG

	Equation solver
1 thread	15,905.0 sec
4 threads	6,417.4 sec
1 thread 1GPU	874.5 sec



GPU SPEEDUP (COMPUTATIONAL FLUID DYNAMICS)

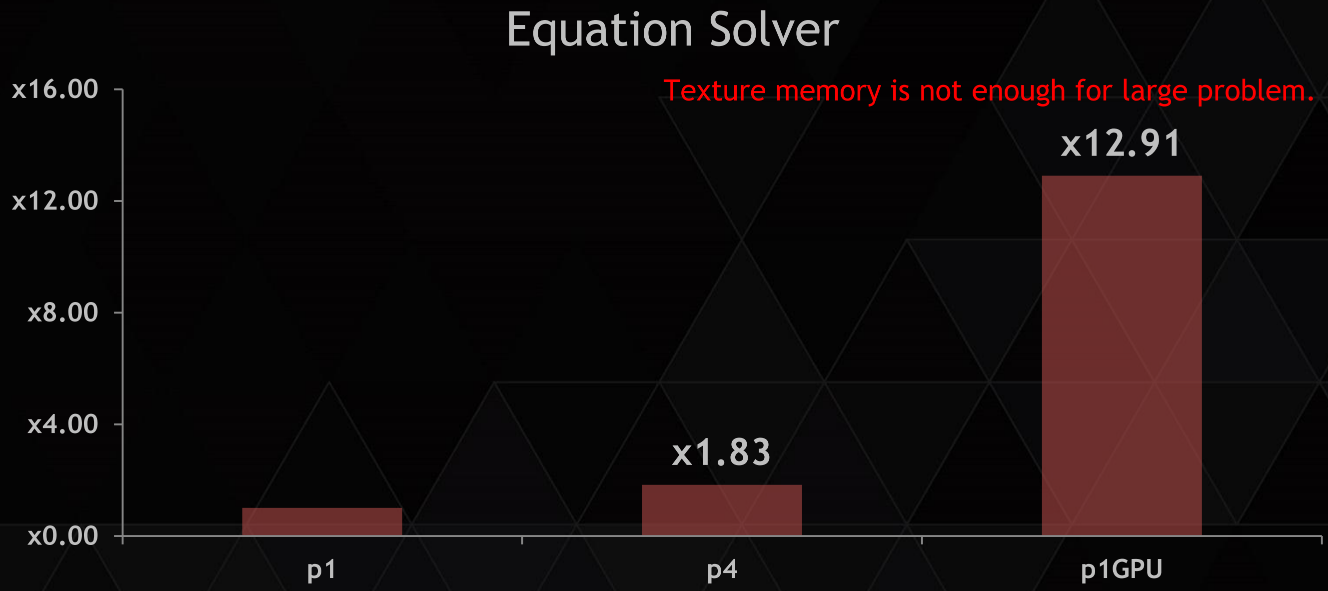


Turbulent Flow / Mesh Deformation

ELEMENTS	18,795,563
NODES	3,614,796

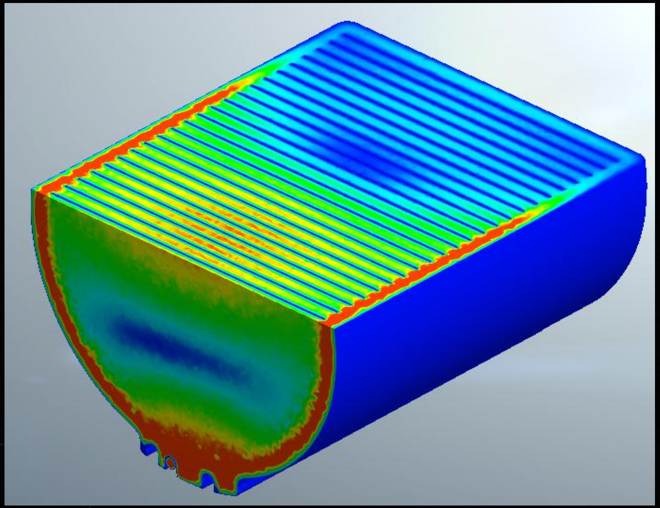
	Specifications
CPU	2x Intel Xeon 2.97 GHz (8 cores)
RAM	48 Gbyte
GPU	GeForce GTX TITAN(DP on)
Solver	Stabilized Bi Conjugate Gradient(2), AMG

	Equation solver
1 thread	21,072.5 sec
4 threads	11,484.5 sec
1 thread 1GPU	1,632.7 sec



FUTURE PLAN(CFD)

PRESENT ISSUE

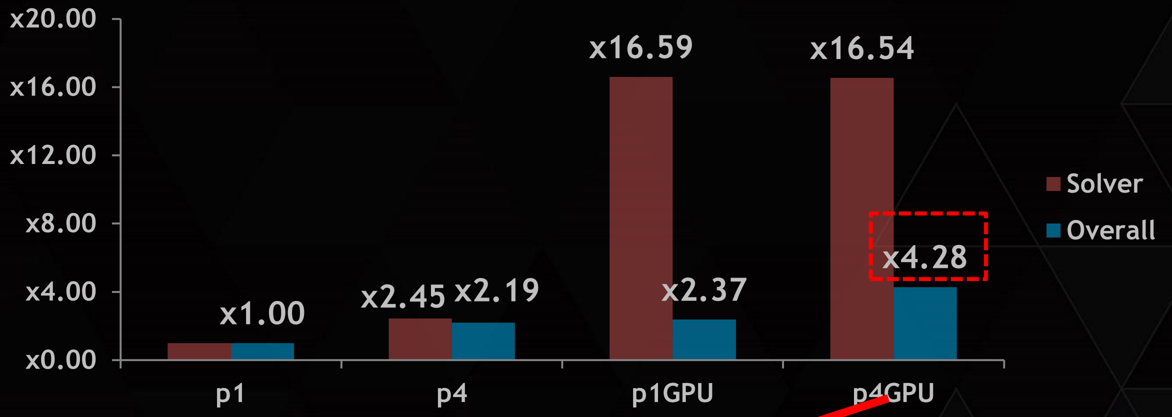


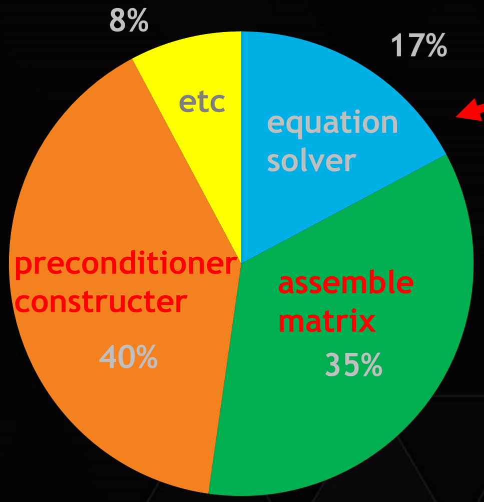
Turbulent Flow / Temperature Analysis

ELEMENTS 5,785,184

NODES 1,029,530

Specifications	
CPU	2x Intel Xeon 2.97 GHz (8 cores)
RAM	48 Gbyte
GPU	GeForce GTX TITAN(DP on)

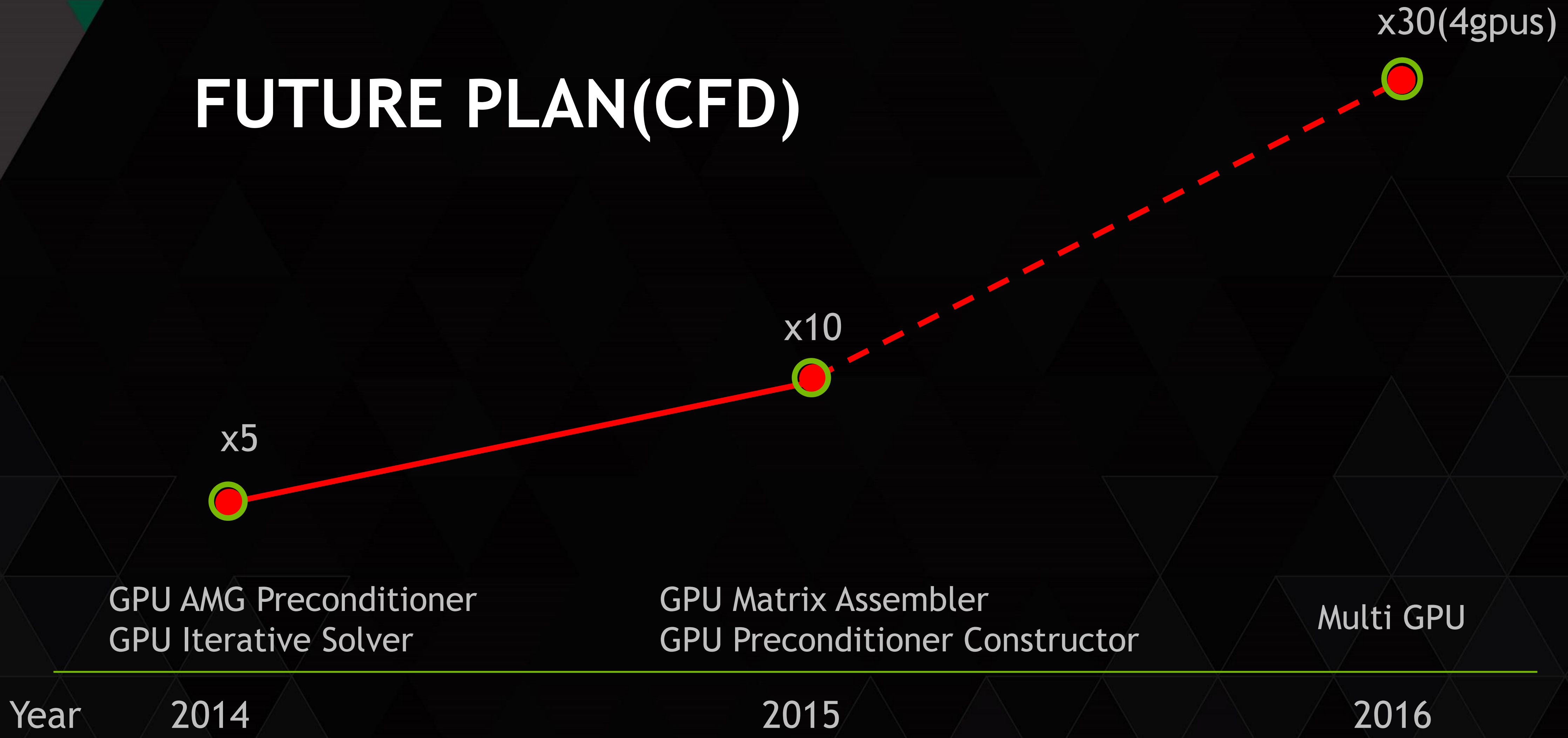




Assemble matrix + preconditioner constructor = 75% ← We are focusing on this.

Time graph of the calculation with a GPU and 4 threads.

FUTURE PLAN(CFD)



GPU TECHNOLOGY CONFERENCE

THANK YOU

JOIN THE CONVERSATION

#GTC15