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

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

Equations
(Structural, Fluid, Chemical, Electric, etc)
COMPUTING PROCEDURE

Mesh (Computing Domain)

\[ A_{11}x_1 + A_{13}x_3 + A_{14}x_4 = b_1 \]
\[ A_{31}x_1 + A_{33}x_3 + A_{34}x_4 = b_3 \]
\[ A_{41}x_1 + A_{43}x_3 + A_{44}x_4 = b_4 \]
COMPUTING PROCEDURE

Mesh (Computing Domain)

\[ A_{11}x_1 + A_{12}x_2 + A_{13}x_3 \ldots = b_1 \]
\[ A_{21}x_1 + A_{22}x_2 + A_{23}x_3 \ldots = b_2 \]
\[ A_{31}x_1 + A_{32}x_2 + A_{33}x_3 \ldots = b_3 \]

\[ \ldots \]

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

\[ Ax = b \]
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)

Top Level

Level - 2

Assemble front 0 and front 1

Assemble front 2 and front 3

Level - 1

Front 0

Front 1

Front 2

Front 3
Is leading dimension of frontal matrix is larger than 1024 (Fermi) or 2048 (Kepler)?

- **no**
  - **CPU Factorization**
    - POPTRF / SYTRF / GEPTRF
  - CUBLAS is used to apply GPU computing

- **yes**
  - **GPU Factorization**
    - GPU_POPTRF / GPU_SYTRF / GPU_GEPTRF

Top Level
GPU SPEEDUP (STRUCTURAL ANALYSIS)

Specifications

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2x Intel Xeon 2.97 GHz (8 cores)</td>
</tr>
<tr>
<td>RAM</td>
<td>48 Gbyte</td>
</tr>
<tr>
<td>GPU</td>
<td>Tesla C2075 1ea</td>
</tr>
</tbody>
</table>

Equation solver

<table>
<thead>
<tr>
<th>Threads</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 thread</td>
<td>617.246 sec</td>
</tr>
<tr>
<td>1 thread 1 GPU</td>
<td>162.942 sec</td>
</tr>
<tr>
<td>4 threads</td>
<td>318.879 sec</td>
</tr>
<tr>
<td>4 threads 1 GPU</td>
<td>97.095 sec</td>
</tr>
</tbody>
</table>

Normal Mode - Fan (LDLT - GPU_SYPRTRF)
**GPU SPEEDUP** *(STRUCTURAL ANALYSIS)*

Linear Static - casting tool (LLT - DPOPTRF)

**Specifications**

<table>
<thead>
<tr>
<th>CPU</th>
<th>2x Intel Xeon 2.97 GHz (8 cores)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>48 Gbyte</td>
</tr>
<tr>
<td>GPU</td>
<td>Tesla C2075 1ea</td>
</tr>
</tbody>
</table>

**Equation solver**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 thread</td>
<td>67.736 sec</td>
</tr>
<tr>
<td>1 thread 1 GPU</td>
<td>19.578 sec</td>
</tr>
<tr>
<td>4 threads</td>
<td>25.365 sec</td>
</tr>
<tr>
<td>4 threads 1 GPU</td>
<td>10.047 sec</td>
</tr>
</tbody>
</table>

**Graph**

- P1: x1.00
- P1 GPU: x3.46
- P4: x2.67
- P4 GPU: x6.74
ITERATIVE SOLVER

<table>
<thead>
<tr>
<th>Iterative Solvers</th>
<th>Preconditioners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conjugate Gradient</td>
<td>Incomplete LU0</td>
</tr>
<tr>
<td>Bi Conjugate Gradient</td>
<td>Incomplete LU1</td>
</tr>
<tr>
<td>Stabilized Conjugate Gradient</td>
<td>Algebraic Multi-grid(AMG)</td>
</tr>
<tr>
<td>Stabilized Conjugate Gradient(2)</td>
<td></td>
</tr>
<tr>
<td>Quasi Minimal Residual Method</td>
<td></td>
</tr>
<tr>
<td>Deflated Conjugate Gradient</td>
<td></td>
</tr>
<tr>
<td>Deflated Bi Conjugate Gradient</td>
<td></td>
</tr>
<tr>
<td>Deflated Stabilized Conjugate Gradient</td>
<td></td>
</tr>
<tr>
<td>Deflated Stabilized Conjugate Gradient(2)</td>
<td></td>
</tr>
<tr>
<td>Deflated Quasi Minimal Residual Method</td>
<td></td>
</tr>
</tbody>
</table>
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 ITERATIVE SOLVER

Conjugate gradient method

Solve $Ax = b$

$r = b - Ax$

$p_0 = r_0$

repeat

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

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

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

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

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

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

$k++$

end repeat

Used algorithms

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSRMV</td>
<td></td>
<td>gpu CSRMV</td>
</tr>
<tr>
<td>dot product</td>
<td>cublas dot product</td>
<td></td>
</tr>
<tr>
<td>axpy</td>
<td></td>
<td>gpu axpy</td>
</tr>
<tr>
<td>axpy2</td>
<td></td>
<td>gpu axpy2</td>
</tr>
</tbody>
</table>

csrmv : sparse matrix vector multiplication

axpy : $x = x + \alpha y$

axpy2 : $x = \alpha x + y$
**GPU AMG**

**Algebraic Multi-grid**

- **Relaxation**
- **Compute residual**
- **Restrict (Coarsen)**
- **Interpolate (Prolongation)**
- **Correcting**
- **Solve** $A_{\text{low}}^{x_{\text{low}}}=b_{\text{low}}$

- **Used algorithm**
  - **CPU**
    - CSRMV
  - **GPU**
    - gpu CSRMV

**csrmv**: sparse matrix vector multiplication
## ACCELERATE CSRMV

Compressed Sparse Row (CSR) format

<table>
<thead>
<tr>
<th>Values</th>
<th>Row pointer</th>
<th>Column index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 -1 0 0</td>
<td>2 -1 2 -1 -1</td>
<td>0 1 0 1 2</td>
</tr>
<tr>
<td>-1 2 -1 0</td>
<td>-1 2 -1</td>
<td>1 2 1 2</td>
</tr>
<tr>
<td>0 -1 2 -1</td>
<td>0 2 5 8 10</td>
<td>3 2 3</td>
</tr>
<tr>
<td>0 0 -1 2</td>
<td>0 2 -1 -1 2</td>
<td>3 2 3</td>
</tr>
</tbody>
</table>

**Values**
- 2
- -1
- 0
- 0
- 2
- -1
- 2
- -1
- 0
- 1
- 0
- 1
- 2
- 1
- 2
- -1
- -1
- 2

**Row pointer**
- 0
- 2
- 5
- 8
- 10

**Column index**
- 0
- 1
- 0
- 1
- 2
- 1
- 2
- 3
- 2
- 3
__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;
    }
}
__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];
    }
}
__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 * blockDim.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 SPEEDUP (COMPUTATIONAL FLUID DYNAMICS)

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

Turbulent Flow Analysis

- ELEMENTS: 4,286,496
- NODES: 857,547

Equation solver

<table>
<thead>
<tr>
<th>Threads</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 thread</td>
<td>4,233.8</td>
</tr>
<tr>
<td>4 threads</td>
<td>1,548.4</td>
</tr>
<tr>
<td>1 thread 1GPU</td>
<td>293.2</td>
</tr>
</tbody>
</table>

Equation Solver

- p1: x0.00
- p4: x4.00
- p1GPU: x16.00
- x2.73
- x14.44
GPU SPEEDUP (COMPUTATIONAL FLUID DYNAMICS)

 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

 Turbulent Flow / Temperature Analysis

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

 Equation solver

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

 x2.45  x16.59
**GPU SPEEDUP** (COMPUTATIONAL FLUID DYNAMICS)

**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

<table>
<thead>
<tr>
<th>Element</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELEMENTS</td>
<td>7,223,319</td>
</tr>
<tr>
<td>NODES</td>
<td>1,370,312</td>
</tr>
</tbody>
</table>

**Equation solver**

- 1 thread: 15,905.0 sec
- 4 threads: 6,417.4 sec
- 1 thread 1GPU: 874.5 sec

**Equation Solver**

- **x18.19**
- **x2.48**
GPU SPEEDUP (COMPUTATIONAL FLUID DYNAMICS)

Specifications

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2x Intel Xeon 2.97 GHz (8 cores)</td>
</tr>
<tr>
<td>RAM</td>
<td>48 Gbyte</td>
</tr>
<tr>
<td>GPU</td>
<td>GeForce GTX TITAN (DP on)</td>
</tr>
<tr>
<td>Solver</td>
<td>Stabilized Bi Conjugate Gradient(2), AMG</td>
</tr>
</tbody>
</table>

Turbulent Flow / Mesh Deformation

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

Equation Solver

<table>
<thead>
<tr>
<th>Threads</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 thread</td>
<td>21,072.5 sec</td>
</tr>
<tr>
<td>4 threads</td>
<td>11,484.5 sec</td>
</tr>
<tr>
<td>1 thread 1GPU</td>
<td>1,632.7 sec</td>
</tr>
</tbody>
</table>

Texture memory is not enough for large problem.
FUTURE PLAN (CFD)
PRESENT ISSUE

Specifications

CPU
2x Intel Xeon 2.97 GHz (8 cores)

RAM
48 Gbyte

GPU
GeForce GTX TITAN (DP on)

Turbulent Flow / Temperature Analysis

ELEMENTS
5,785,184

NODES
1,029,530

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

Time graph of the calculation with a GPU and 4 threads.
FUTURE PLAN (CFD)

<table>
<thead>
<tr>
<th>Year</th>
<th>2014</th>
<th>2015</th>
<th>2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU AMG Preconditioner</td>
<td>x5</td>
<td>x10</td>
<td>x30 (4gpus)</td>
</tr>
<tr>
<td>GPU Iterative Solver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU Matrix Assembler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU Preconditioner Constructor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi GPU</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
THANK YOU

JOIN THE CONVERSATION

#GTC15  

Twitter  Facebook  LinkedIn