University of Pittsburgh

High Performance Matrix-Free Method for Large-Scale FEA on GPUs

Petros Apostolou, Florian Dugast and Albert To*

Dept. of Mechanical Engineering and Materials Science

University of Pittsburgh

April-02-2020

Outline

1. Problem Statement
2. Finite Element System Formulation
3. Necessity of Matrix-Free Method

1-3 Introduction

4. State of the Art
5. CPU Matrix-Free Conjugate Gradient
6. GPU Matrix-Free Conjugate Gradient

4-6 Method

7. Contribution
8. Future Research

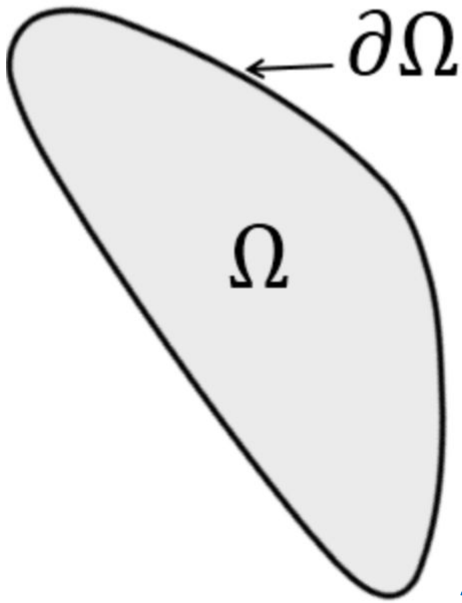
7-8 Results

Problem Statement

3D Steady-State Heat Conduction (Isotropic thermal conductivity k) :

$$\overset{0}{\rho \mathbf{C_p} \frac{\partial \mathbf{T}}{\partial \mathbf{t}}} = \overset{1}{\kappa} \left(\frac{\partial^2 \mathbf{T}}{\partial \mathbf{x}^2} + \frac{\partial^2 \mathbf{T}}{\partial \mathbf{y}^2} + \frac{\partial^2 \mathbf{T}}{\partial \mathbf{z}^2} \right) + \mathbf{Q}(\mathbf{x}, \mathbf{y}, \mathbf{z}) \quad \Rightarrow$$

$$-\left(\frac{\partial^2 \mathbf{T}}{\partial \mathbf{x}^2} + \frac{\partial^2 \mathbf{T}}{\partial \mathbf{y}^2} + \frac{\partial^2 \mathbf{T}}{\partial \mathbf{z}^2} \right) = \mathbf{Q}(\mathbf{x}, \mathbf{y}, \mathbf{z}) \quad \forall (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \boldsymbol{\Omega}$$



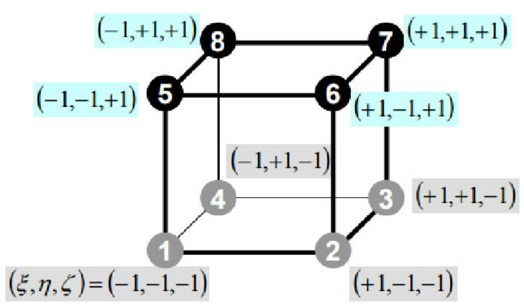
Boundary Conditions (Dirichlet):

$$\mathbf{T}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathbf{0} \quad \forall (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \partial \boldsymbol{\Omega}$$

Finite Element System Formulation

"**Weak**" form of the heat conduction equation (Galerkin method):

$$\int_V \left[\left(\frac{\partial N}{\partial x}\right)^{\mathsf{T}} \left(\frac{\partial N}{\partial x}\right) + \left(\frac{\partial N}{\partial y}\right)^{\mathsf{T}} \left(\frac{\partial N}{\partial y}\right) + \left(\frac{\partial N}{\partial z}\right)^{\mathsf{T}} \left(\frac{\partial N}{\partial z}\right) \right] \left[T \right] dV = \int_V Q dV$$



N = [N1 N2 … N8]: shape functions for the 8-node finite element (hexahedron)

N1 = 1/8 (1-ξ)(1-n)(1-ζ)

[Ke]{Te} = {Fe}: finite element system for ele. **e**

Ke[8 x 8] = B^{T}B: where **B = ∇N**

$$\boxed{\mathbf{KT = F}}$$

K: global conductivity matrix

T: temperature field

F: heat source vector

Necessity of Matrix-Free Method

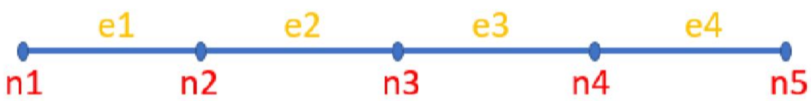
$$\overset{[N \text{ x } N]}{\begin{bmatrix} \quad K \quad \end{bmatrix}} \overset{[N \text{ x } 1]}{\begin{bmatrix} T \end{bmatrix}} = \overset{[N \text{ x } 1]}{\begin{bmatrix} F \end{bmatrix}}$$

Size[K] becomes a major bottleneck for large sizes (e.g. N > 10M):

- N becomes **λN** → **[λN x λN] = (λN)^2** → **Expensive** **(λ = 1,2,3,...)**
- **Sparse-matrix storage is NOT enough** → **Exceeds memory limits**

Suggestion:
Matrix-free method allows for larger size domains by avoiding the global matrix assembly → [N x N] is not necessary to be stored or computed

Necessity of Matrix-Free Method



Assuming:

$$\mathbf{K_e} = \mathbf{K_{ij}} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$\begin{array}{ccccc} \mathbf{K} & & \mathbf{T} & & \mathbf{F} \\ \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 1+1 & -1 & 0 & 0 \\ 0 & -1 & 1+1 & -1 & 0 \\ 0 & 0 & -1 & 1+1 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} & & \begin{bmatrix} T1 \\ T2 \\ T3 \\ T4 \\ T5 \end{bmatrix} & = & \begin{bmatrix} F1 \\ F2 \\ F3 \\ F4 \\ F5 \end{bmatrix} \end{array}$$

K1 e1 K2 e2 K3 e3 K4 e4

If local to global nodal values transfers are given then instead of K only Ke is needed!!

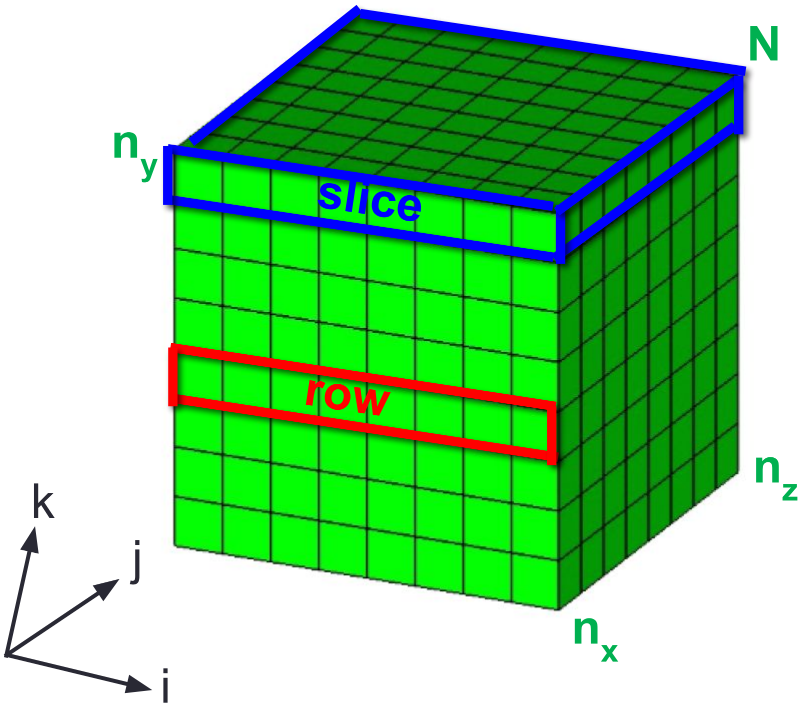
Writing the linear finite equation for node 2:

$$K_e^{e_1}[2][1]T[1] + K_e^{e_1}[2][2]T[2] + K_e^{e_2}[1][1]T[2] + K_e^{e_2}[1][2]T[3] = F[2]$$

Necessity of Matrix-Free Method

Voxel-based technique in hexahedral mesh:

- **$N = [n_x \times n_y \times n_z]$** : mesh nodes
- **$id = i + (n_x \times j) + [(n_x \times n_y) \times k]$**
- **node(id+1) = node(id) + 1**
  row(id+1) = row(id) + 1
  slice(id+1) = slice(id) + 1
- **e □ id □ edof[id]**



Regular connectivity facilitates matrix-free methods

State of the Art

Matrix-Free Methods:

Thomas Hughes "***An element-by-element solution algorithm for problems of structural and solid mechanics***". Division of Applied Mechanics, Durand Building, Stanford University, Stanford, August 1982, U.S.A. **[1st Instance]**

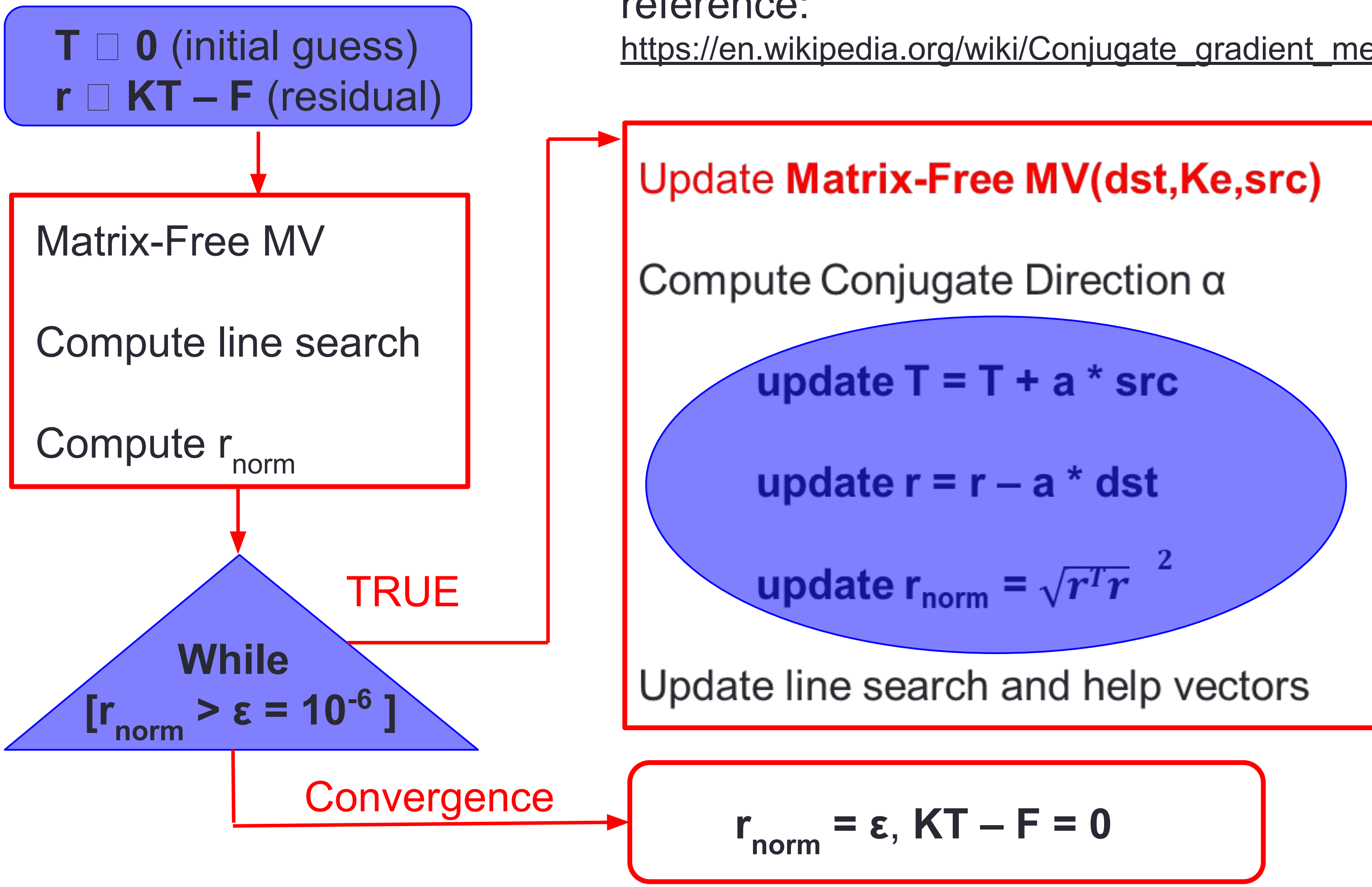
J.M.Frutos & D.H.Perez "***Efficient matrix-free GPU implementation of Fixed Grid Finite Element Analysis***" Department of Structures and Construction, Technical University of Cartagena, May 2015 Spain. **[20 x Speed-Up – 1 GPU]**

Pros: Significant savings in computational time and memory

Cons: Difficult for Unstructured Mesh and Complex Boundaries

CPU Matrix-Free Conjugate Gradient

reference:
https://en.wikipedia.org/wiki/Conjugate_gradient_method



CPU Matrix-Free Conjugate Gradient

Matrix-Free MV(dst, Ke, src):

```
set dst □ 0
```

```
for element in mesh
   for rows in Ke[8 x 8]
        extract row_index
        define tmp □ 0
        for nodes in element
            extract DOF_index
            tmp = tmp + Ke * src
        end
        dst = dst + tmp
   end
end
```

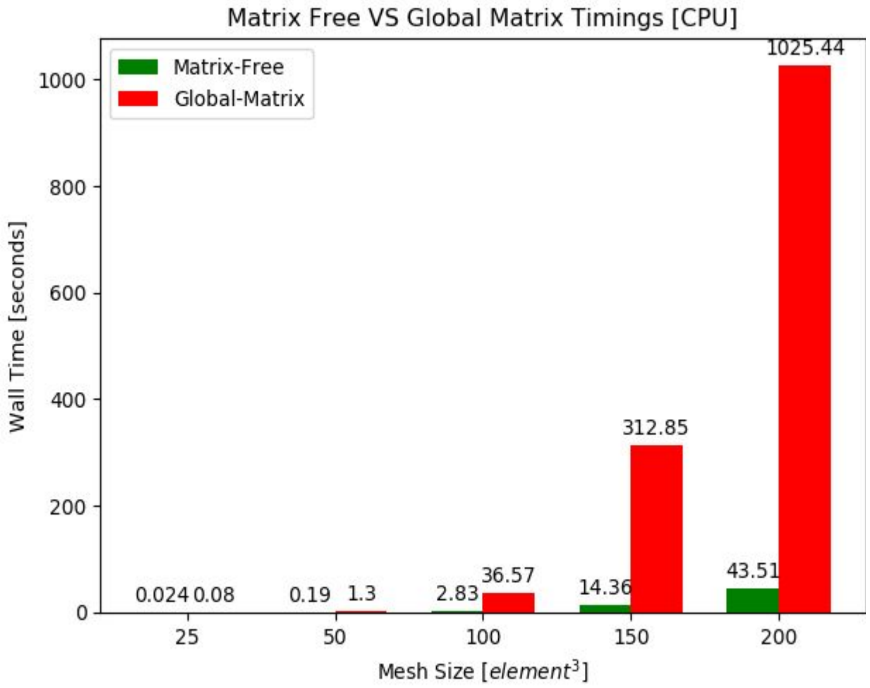
```
! impose Dirichlet BC
for element in mesh
   if  [element == face]
        src □ 0
        dst □ 0
   end
end

return dst
```

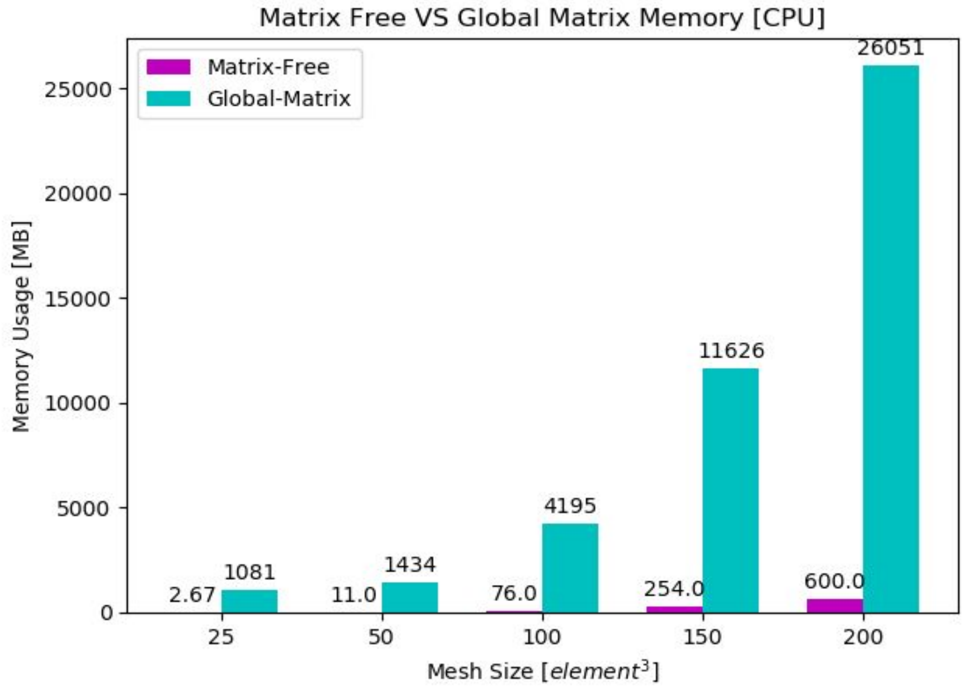
MFMV consumes 80% out of the total computations

CPU Matrix-Free Conjugate Gradient

Computational Time on CPU:



Memory usage on CPU:



MFCG is 23.84 x times faster than the global matrix assembly solver
MFCG consumes 43.42 x times less memory

CPU Matrix-Free Conjugate Gradient

Verification of the numerical results with the **PETSc** library
Ref: https://www.mcs.anl.gov/petsc/

Relative numerical error :

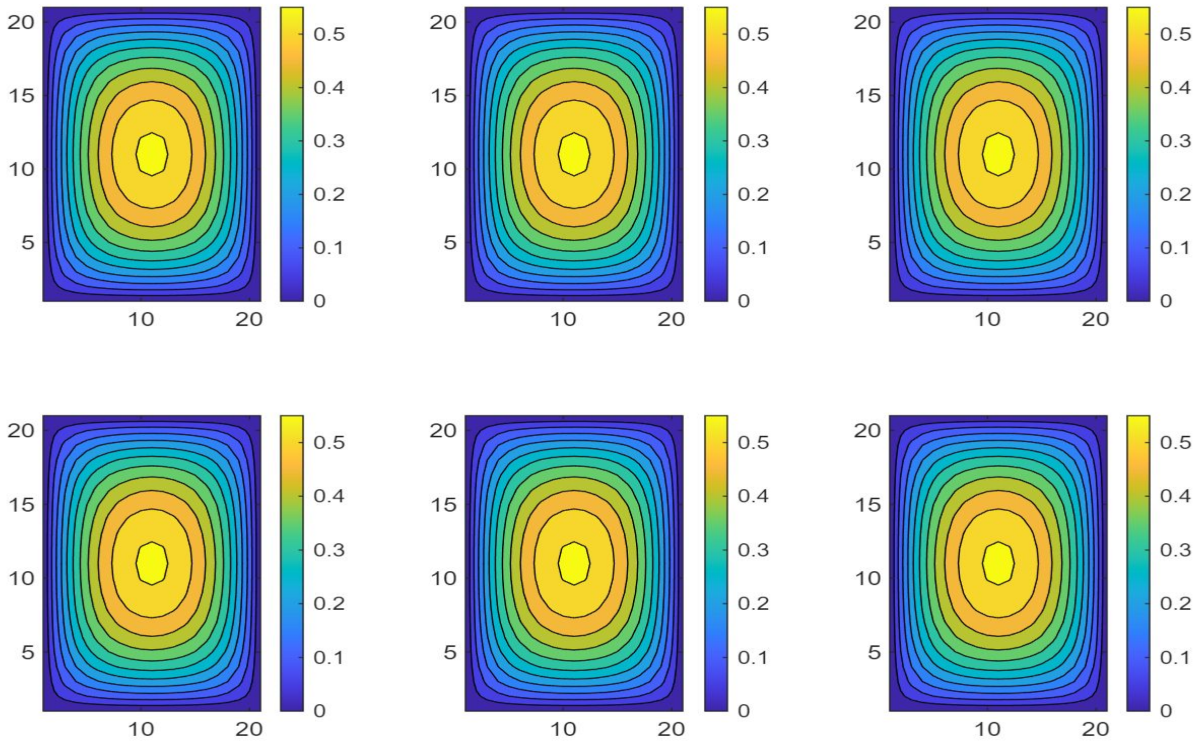
$$\mathrm{err} = \sum_{i=1}^{N} \left| \frac{\mathrm{PETSc}^{i} - \mathrm{MFCG}^{i}}{\mathrm{PETSc}^{i}} \right| = 0.0582\%$$

Dirichlet BC: T = 0 is satisfied on boundary faces

CPU Matrix-Free Conjugate Gradient

Verification of the numerical results with the **FreeFem++** library – https://freefem.org/

xy-plane **xz-plane** **yz-plane**



FreeFem++
SIZE = 20^3

MFCG
SIZE = 20^3

Good agreement with FreeFemm++ library
3D Symmetric (x,y,z)

GPU Matrix-Free Conjugate Gradient

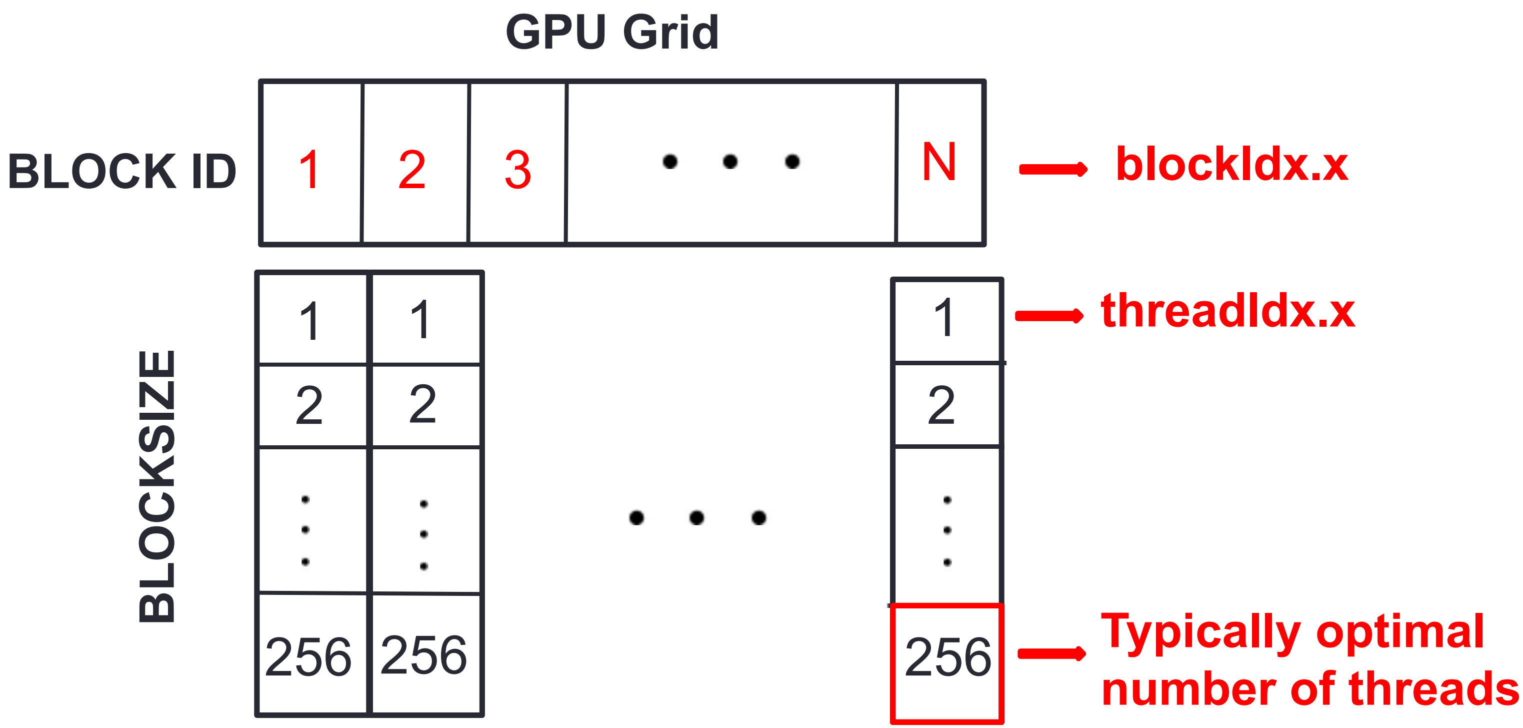
NVIDIA GPU Specification Comparison

	Titan V	Titan Xp	GTX Titan X (Maxwell)	GTX Titan
CUDA Cores	5120	3840	3072	2688
Tensor Cores	640	N/A	N/A	N/A
ROPs	96	96	96	48
Core Clock	1200MHz	1485MHz	1000MHz	837MHz
Boost Clock	1455MHz	1582MHz	1075MHz	876MHz
Memory Clock	1.7Gbps HBM2	11.4Gbps GDDR5X	7Gbps GDDR5	6Gbps GDDR5
Memory Bus Width	3072-bit	384-bit	384-bit	384-bit
Memory Bandwidth	653GB/sec	547GB/sec	336GB/sec	288GB/sec
VRAM	12GB	12GB	12GB	6GB
L2 Cache	4.5MB	3MB	3MB	1.5MB
Single Precision	13.8 TFLOPS	12.1 TFLOPS	6.6 TFLOPS	4.7 TFLOPS
Double Precision	6.9 TFLOPS (1/2 rate)	0.38 TFLOPS (1/32 rate)	0.2 TFLOPS (1/32 rate)	1.5 TFLOPS (1/3 rate)

5120 cuda cores & 12 GB memory with ~ 7TFLOPS double precision

GPU Matrix-Free Conjugate Gradient

CUDA Thread-Block Architecture:



N = { **ELE_BLOCKS = numElems/BLOCKSIZE + 1**
NOD_BLOCKS = numNodes/BLOCKSIZE + 1

GPU Matrix-Free Conjugate Gradient

Outline of basic steps for the **GPU (CUDA) MFCG** algorithm:

Ref: ***CUDA TOOLKIT***: https://docs.nvidia.com/cuda/index.html

1) Get GPU device **cudaGetDevice**(ID)
2) Allocate global memory **cudaMallocManaged**(…)
3) Copy host **Ke** and **all vectors** to device **cudaMemcpy(…HostToDevice**)
4) Matrix-Free MV<<<**ELE_BLOCKS**,**BLOCKSIZE**>>>(…)
5) cuBLAS for linear algebra updates (**ddot,daxpy,dnrm2**)
6) UpdateVector<<<**NOD_BLOCKS**,**BLOCKSIZE**>>>(…)
7) Copy **T** device to host **cudaMemcpy(…DeviceToHost**)
8) **cudaFree**(…)

GPU Matrix-Free Conjugate Gradient

Matrix-free matrix-vector product **MFMV(dst,src,Ke)**

e = blockIdx.x * BLOCKSIZE + threadIdx.x

if (e < SIZE)

SIZE: Number of Mesh Elements

```
for rows in Ke[8 x 8]
   extract row_index
   define tmp □ 0
   for nodes in element
      extract DOF_index
      tmp[node] = tmp[node] + Ke * src
   end
     sum = tmp[node]
     atomicAdd(&dst[DOF_index [row]], sum)
end
```

Cuda **atomicAdd** operation:

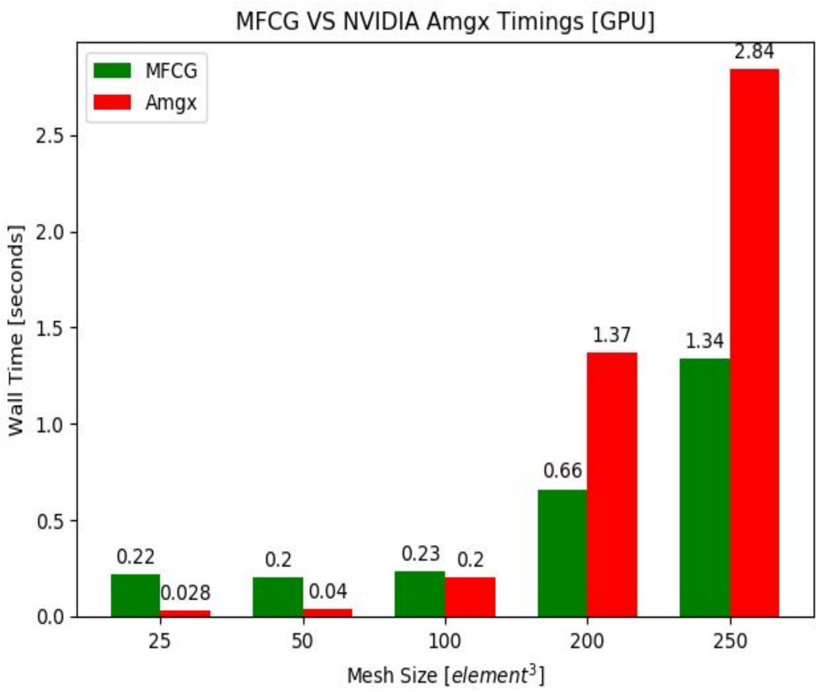
atomicAdd(&vec,scal)
perform n times:
vec = vec + scal

returns: vec at step n

!atomicAdd handles "races conditions" occurring in SIMD

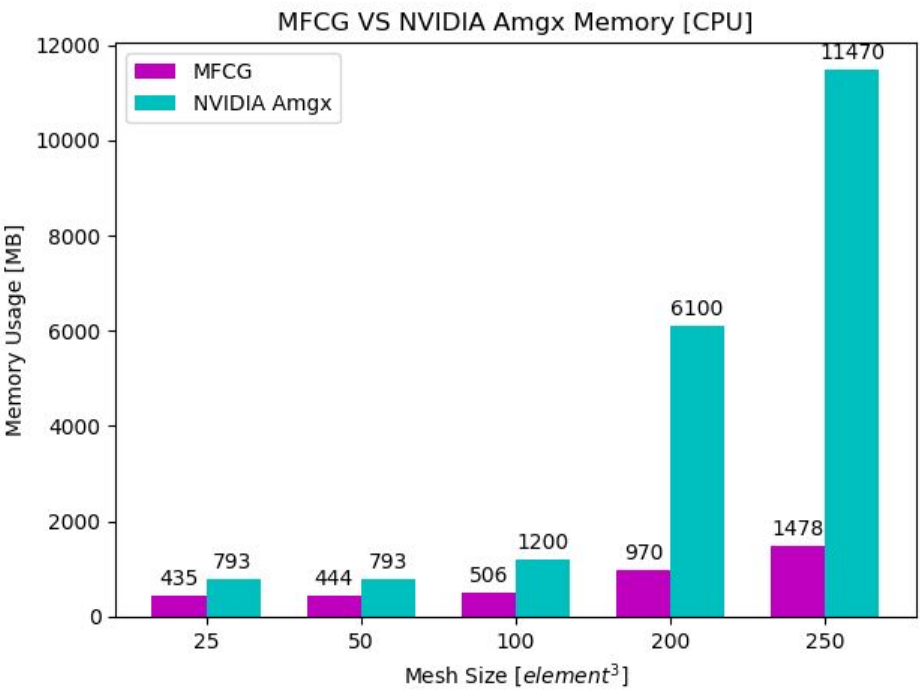
GPU Matrix-Free Conjugate Gradient

Timings of MFCG vs NVIDIA Amgx Up to Size = 250^3:



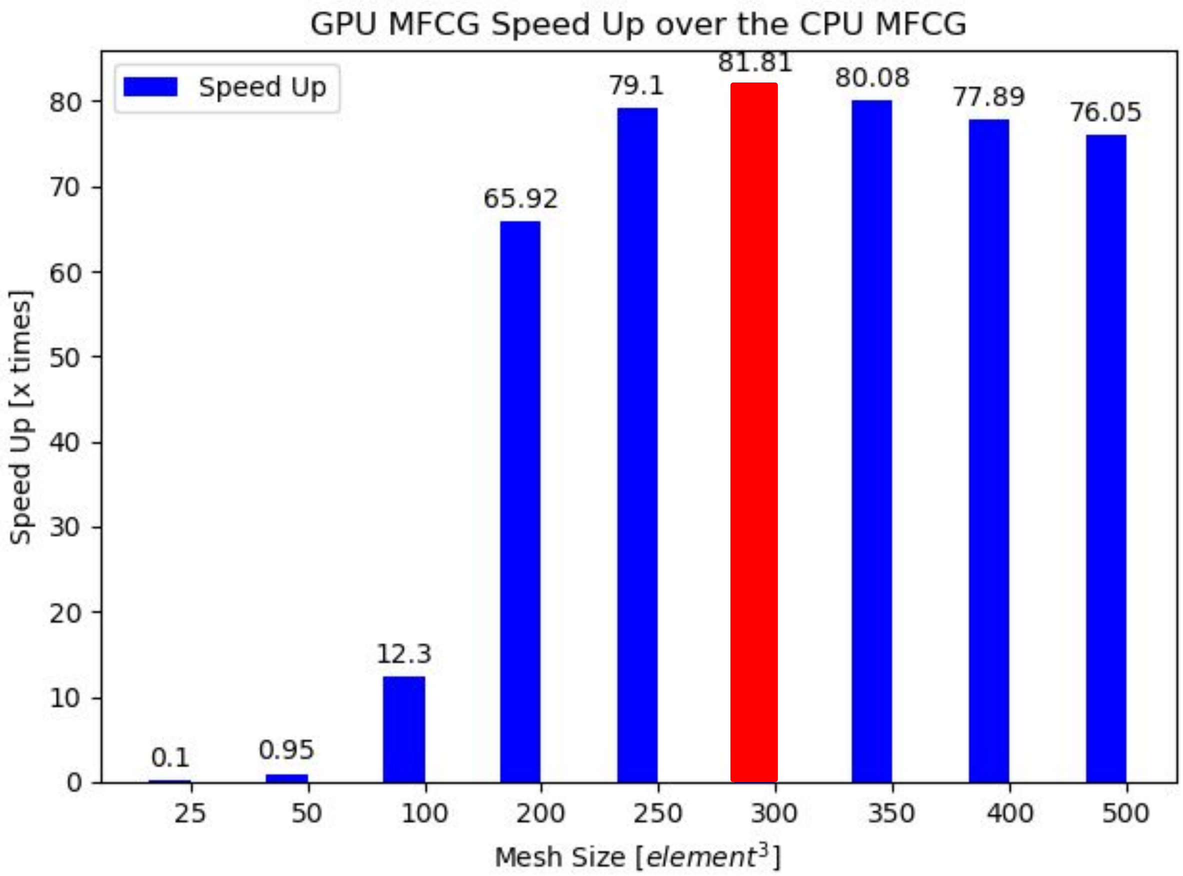
Memory of MFCG vs NVIDIA Amgx Up to Size = 250^3:

Ref: https://github.com/NVIDIA/AMGX



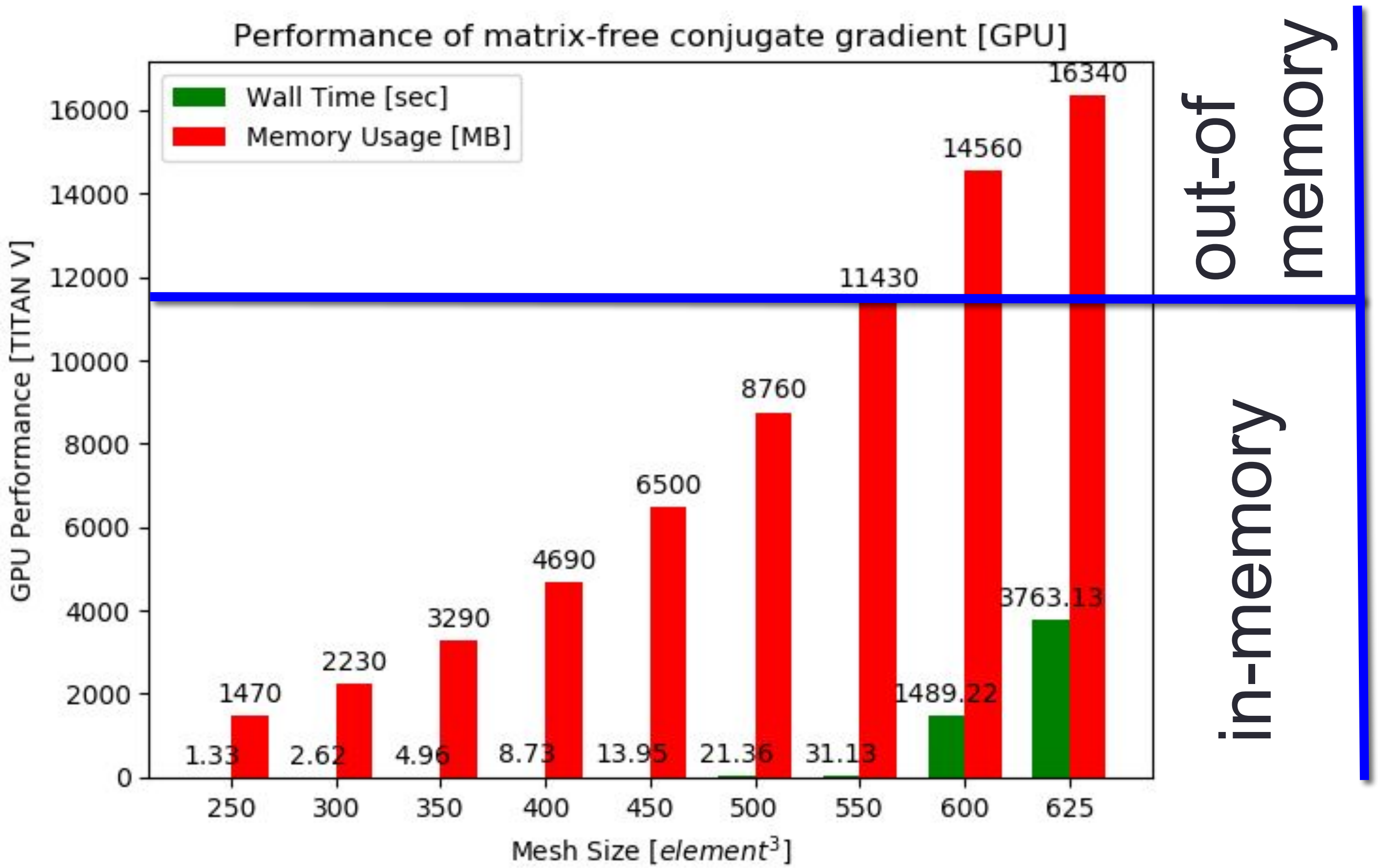
MFCG is 2.12 x times faster than the NVIDIA Amgx Solver
MFCG consumes 7.76 x times less memory than NVIDIA Amgx

GPU Matrix-Free Conjugate Gradient



Maximum GPU Speed-Up = 81.81 x times for double precision

GPU Matrix-Free Conjugate Gradient



Maximum mesh size = 550^3 elements is computed in Wall Time = 31.31s
GPU overruns for size > 550^3 (not recommended)

Contribution

- CPU (C) implementation of MFCG method for a Large-Scale FEA of the Steady-State Heat Conduction using the Voxel-Based Technique

- GPU (CUDA) implementation of MFCG method for the Acceleration and Scaling of the Large-Scale FEA up to 550^3 = 166,375,000 Elements

- A Performance Analysis on the Timings and Memory between:

  a) the CPU MFCG method and PETSc (global matrix-assembly)

  b) the GPU MFCG method and NVIDIA Amgx (global matrix-assembly)

  ! Both PETSc & Amgx use “Sparse-Matrix Storage” (Non-Zero Storage)

Future Research

- ✔ **Muti-GPU (CUDA) on 4 TITAN V GPUs for Large-Scale Transient Heat-Transfer on Layer-by-Layer Process Simulation for L-PBF**
- ✔ **MF-CG MF-PCG (Matrix-Free Jacobi Preconditioner CG)**
- ✔ **Hybrid MPI + Multi-GPU Scheme for High Core Scaling**

Questions…

Thank You

University of Pittsburgh

High Performance Matrix-Free Method for Large-Scale FEA on GPUs

Petros Apostolou, Florian Dugast and Albert To*
Dept. of Mechanical Engineering and Materials Science
University of Pittsburgh
April-02-2020