Imperial College London

Anisotropic Mesh Adaptivity on nVIDIA's CUDA

George Rokos
Software Performance Optimisation Group, Department of Computing

Gerard Gorman
Applied Modelling and Computation Group

Imperial College London

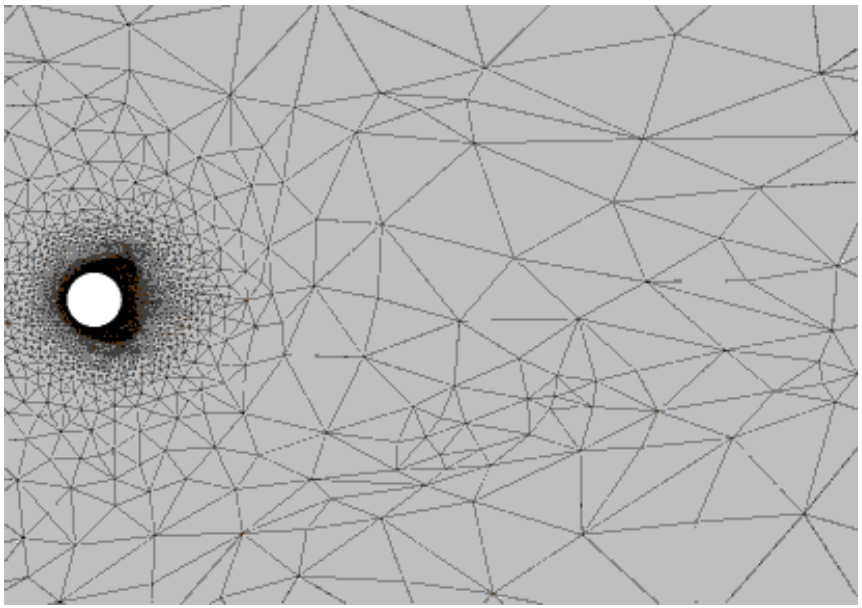
Introduction

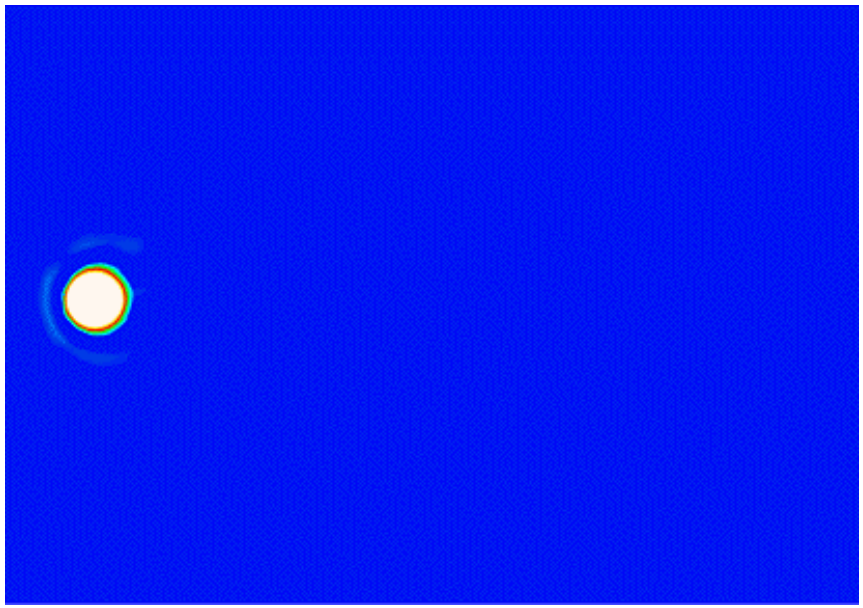
- Computational Fluid Dynamics (CFD)
  - Study of the motion of fluids
  - Modelled using the Navier-Stokes partial differential equations (PDEs)
  - Usually not possible to get analytical solutions
  - Numerical methods
- Finite Element Method (FEM)
  - The domain (i.e. the surface or volume) of interest and the PDE are discretised into sub-domains and the PDE is solved inside each one
  - Sub-domains are elements, usually of triangular (2D) or tetrahedral (3D) shape, which form a structure referred to as mesh.
  - Structured vs. unstructured meshes: tradeoff between computational complexity and numerical flexibility.

Imperial College London

Introduction

- Example application: Mesh adaptivity following flow features past an infinite heated cylinder, simulating the pattern of vortices known as Kármán vortex street.





http://amcg.ese.ic.ac.uk/index.php?title=Heated_Cylinder_Adapt_Example

Imperial College London

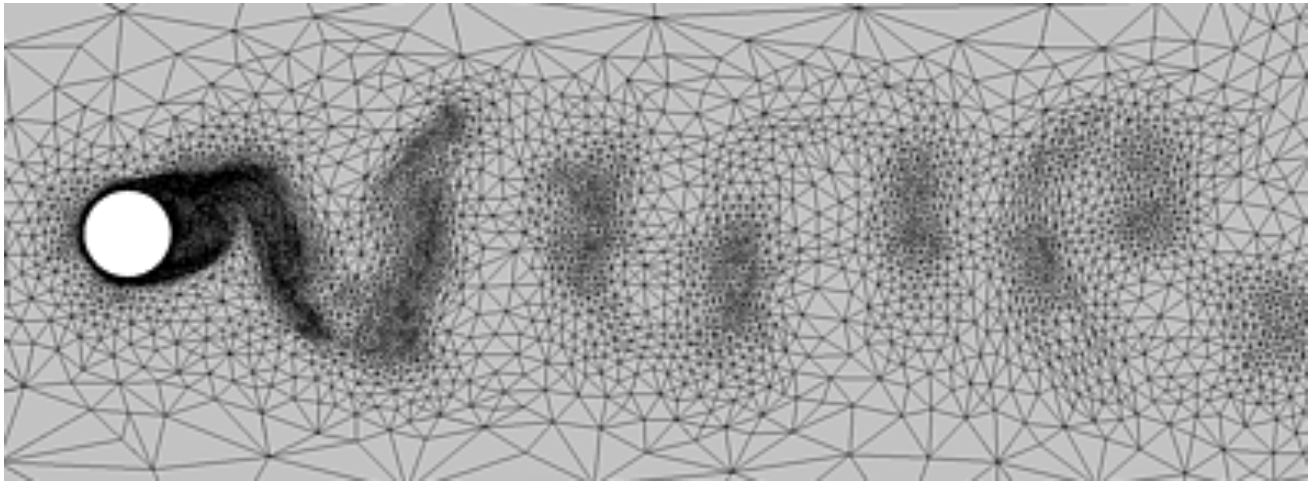
Motivation

- Mesh adaptivity is a computationally heavy task
  - Lots of floating-point arithmetic
- Parallel versions of adaptivity algorithms exist
  - Help speed up the process using MPI, multi-core architectures, GPGPUs etc.
- Good candidate for GPGPU computing
  - Massively parallel architectures manipulating thousands of threads
  - Great floating-point capabilities
- Related work: Fluidity
  - Open source CFD solver, developed at the Applied Modelling and Computation Group, Imperial College London
  - Latest release (4.0) runs on conventional hardware

Imperial College London

Unstructured Meshes

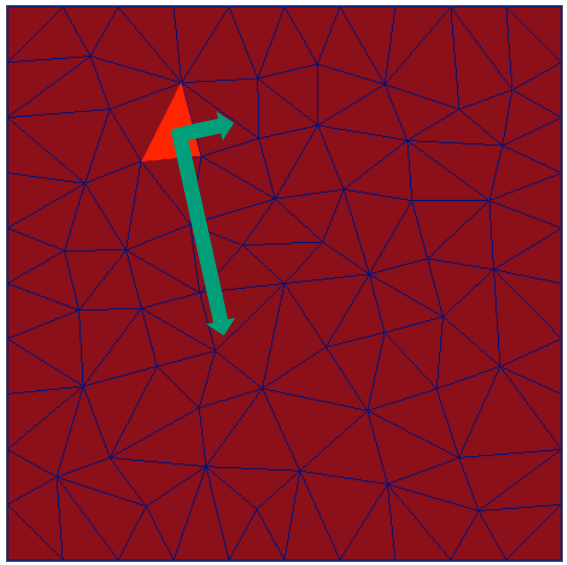
- No regular pattern of discretisation (in contrast to, e.g., a matrix)
- Need explicit information about vertex and element adjacency

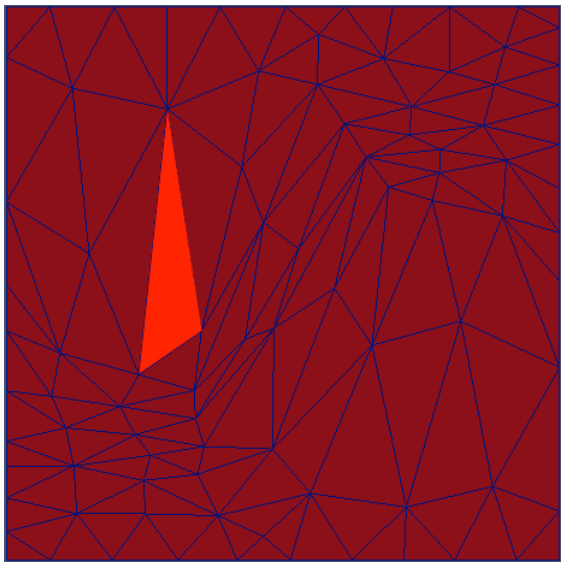


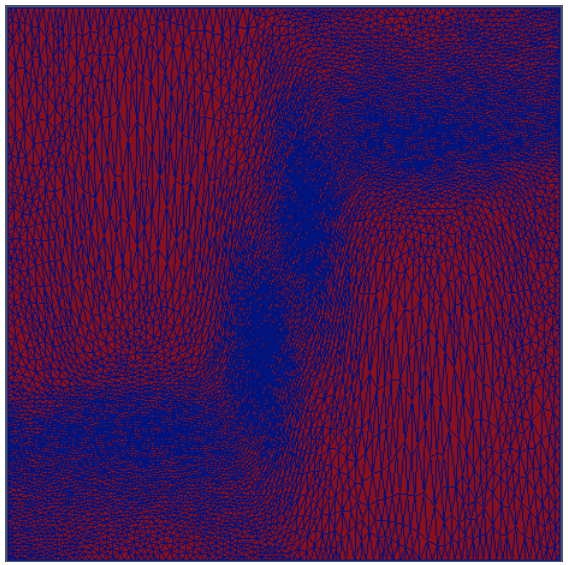
Imperial College London

Anisotropic PDEs

- Directional dependencies: convergence speed and accuracy are favoured if mesh elements have a suitable orientation (size and shape).
- Metric tensor field: for each point, represents the desired length and orientation of a mesh edge containing this point
  - Can be decomposed as: $M = Q \Lambda Q^T$
    - » Q is an orthonormal matrix representing the rotation of the axis system so that the base vectors show the directions to which the element has to be stretched.
    - » Λ is a diagonal matrix representing the amount of stretching in each direction.



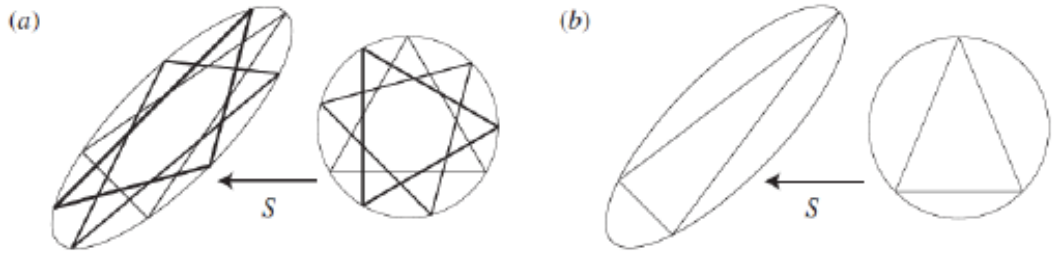




Imperial College London

Metric tensor field and element quality

- In 2D space, it is a 2x2 matrix: $M = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$
- Discretised vertex-wise.
- Vasilevskii & Lipnikov, 1999: the ideal triangle is an equilateral one with edges of unit length with respect to the metric tensor field.

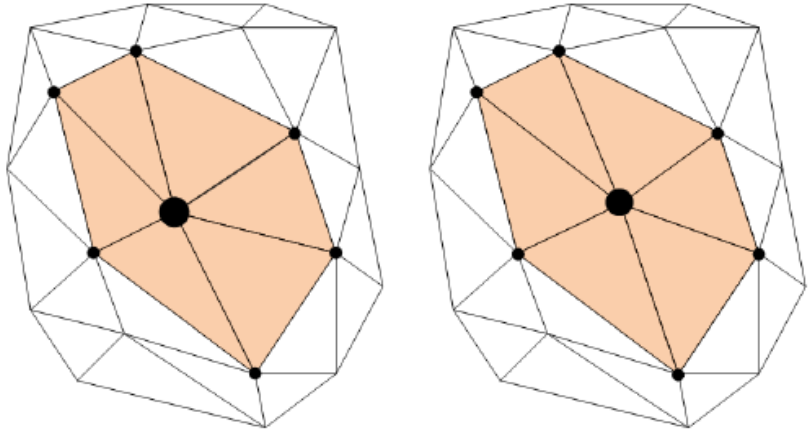


"Anisotropic mesh adaptivity for multi-scale ocean modelling", Piggott, Farrell, Wilson, Gorman, Pain, 2009

Imperial College London

Vertex Smoothing

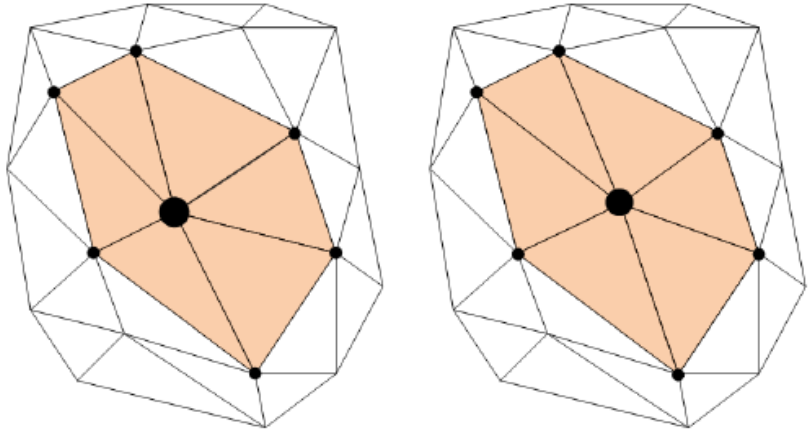
- A simple technique which does not affect mesh topology, i.e. number of elements and connectivity of vertices.
- Examines one mesh “Cavity” at a time, i.e. a central vertex and all its adjacent vertices.
- Central vertex is relocated to a new position such that the quality of the lowest-quality triangle is maximised.



Imperial College London

Propagation and Iterations

- The scope of optimisation is local, i.e. within the cavity.
- After neighbouring cavities have been processed, i.e. adjacent vertices have been relocated, quality of the first cavity may have been degraded.
- Need to perform multiple passes over the mesh in order to bring things to an equilibrium.
- As vertices are relocated to new positions, the metric tensor field for the new position is calculated by interpolating the metric tensor values at the nearest vertices.



Imperial College London

Algorithm by Pain et al.

- The new position x of the central vertex (with initial position p) with respect to the positions y^j of the adjacent vertices is obtained by solving the linear system:

$$(D + A)(x - p) = 0.5(q - Ap)$$

where

$$A = \sum_l M_l, \forall\ edge\ \ l\ \in cavity$$

$$q = \sum_l M_l y_l^j, \forall\ edge\ \ l\ \in cavity$$

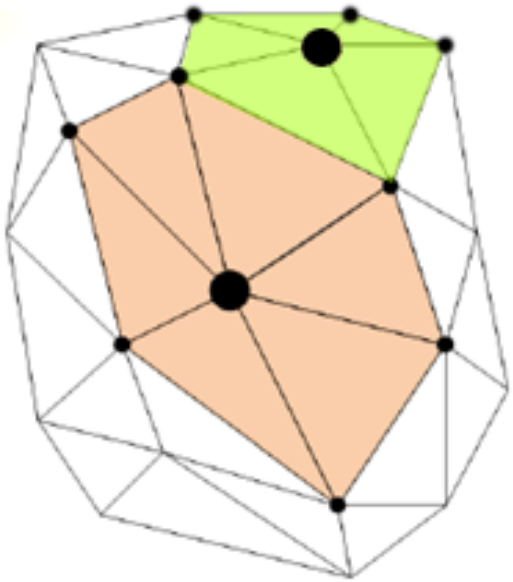
and

$$D_{jk} = \begin{cases} \max\left(A_{jj}, 1.01 \sum_{m=1, m\neq j} |A_{jm}|\right), if\ j = k \\ 0, if\ j \neq k \end{cases}$$

Imperial College London

Parallel Execution

- Adjacent vertices cannot be relocated simultaneously because the new position of a vertex depends on the old positions of all its neighbouring vertices.
- Idea: mesh vertices are coloured so that no two adjacent vertices have the same colour.
- All vertices that have the same colour, i.e. belong to the same independent set, can be smoothed simultaneously.



Based on the framework by Freitag et al.

Implementation

- Array V of vertices, i.e. pairs of coordinates.
- Array C of cavities, i.e. sets of vertex indices of array V, such that the cavity defined by a vertex is accessed using the same index as the one used to access the vertex in V.
- 2D array containing the values of the metric tensor field:
  - The structured representation allows us to treat the metric tensor field as a graphics texture, taking advantage of specialised hardware.
  - In order to transform the unstructured representation to a structured one, it is necessary to super-sample the unstructured representation.
- For a pre-defined number of iterations over the mesh do:
  - Process one independent set (colour) at a time:
    - » Invoke the optimisation kernel, creating as many GPU threads as there are vertices in the set, grouping them in CUDA blocks according to the execution configuration.
    - » Each thread optimises exactly one cavity.
  - Until all independent sets have been processed.

Imperial College London

Metric Tensor Field as Graphics Texture

- Recall that metric tensor field values at new vertex positions are the result of interpolation.
- The same thing happens with graphics textures.
  - A texture is a collection of colour samples at various points.
  - Values at in-between points are calculated by interpolation.
- Benefits:
  - Interpolation is done by dedicated, specialised texture hardware.
    - » Computations in hardware may be completed faster than in software.
    - » CUDA execution units are free to accomplish other tasks.
    - » Fewer registers are used by the kernel, leading to better warp occupancy.
  - Textures are cached in a dedicated cache, leaving more room for vertex data in shared memory/L1/L2 caches.
- Drawbacks:
  - Super-sampling needs more memory.
  - Loss in accuracy is also possible, as mesh density in certain areas may be higher than the resolution a structured representation can offer.

Using dedicated texture hardware to do the interpolation

$$tex(x,y) = (1-\alpha)(1-\beta)T[i,j] + \alpha(1-\beta)T[i+1,j] + (1-\alpha)\beta T[i,j+1] + \alpha\beta T[i+1,j+1]$$

```
#if defined(USE_TEXTURE_MEMORY)
    float4 floatMetric = tex2D(metricTex, coords.iCoord, coords.jCoord);
#else
    double iIndex = coords.jCoord * meshSize;
    int i = floor(constant1 * iIndex);
    iIndex -= i;

    double jIndex = coords.iCoord * meshSize;
    int j = floor(constant1 * jIndex);
    jIndex -= j;

    if(i == constant3)        // if we are on the top or bottom boundary
            metric = cudaMetricValues[constant2 +   j  ] * (1 - jIndex) +
                     cudaMetricValues[constant2 + (j+1)] *    jIndex;
    else if(j == constant3) // if we are on the left or right boundary
            metric = cudaMetricValues[(i+1) * meshSize - 1] * (1 - iIndex) +
                     cudaMetricValues[(i+2) * meshSize - 1] *    iIndex;
    else
            metric = cudaMetricValues[  i   * meshSize +   j  ] * (1 - iIndex) * (1 - jIndex) +
                     cudaMetricValues[  i   * meshSize + (j+1)] * (1 - iIndex) *    jIndex    +
                     cudaMetricValues[(i+1) * meshSize +   j  ] *    iIndex    * (1 - jIndex) +
                     cudaMetricValues[(i+1) * meshSize + (j+1)] *    iIndex    *    jIndex;
#endif
```

Setting up texture memory

```
#if defined(USE_TEXTURE_MEMORY)
    cudaChannelFormatDesc channelDesc =
            cudaCreateChannelDesc(32, 32, 32, 32, cudaChannelFormatKindFloat);
    cudaMallocArray(&cudaMetricValues, &channelDesc, meshSize, meshSize);
    cudaMemcpyToArray(cudaMetricValues, 0, 0, floatMetricValues,
            meshSize * meshSize * sizeof(float4), cudaMemcpyHostToDevice);

    metricTex.normalized = true;
    metricTex.filterMode = cudaFilterModeLinear;
    metricTex.addressMode[0] = cudaAddressModeClamp;
    metricTex.addressMode[1] = cudaAddressModeClamp;

    cudaBindTextureToArray(metricTex, cudaMetricValues, channelDesc);
#else
    cudaMalloc((void **) &cudaMetricValues, meshSize * meshSize * sizeof(Vector2dPair));
    cudaMemcpy(cudaMetricValues, metricValues,
            meshSize * meshSize * sizeof(Vector2dPair), cudaMemcpyHostToDevice);
#endif
```

Imperial College London

Experiments

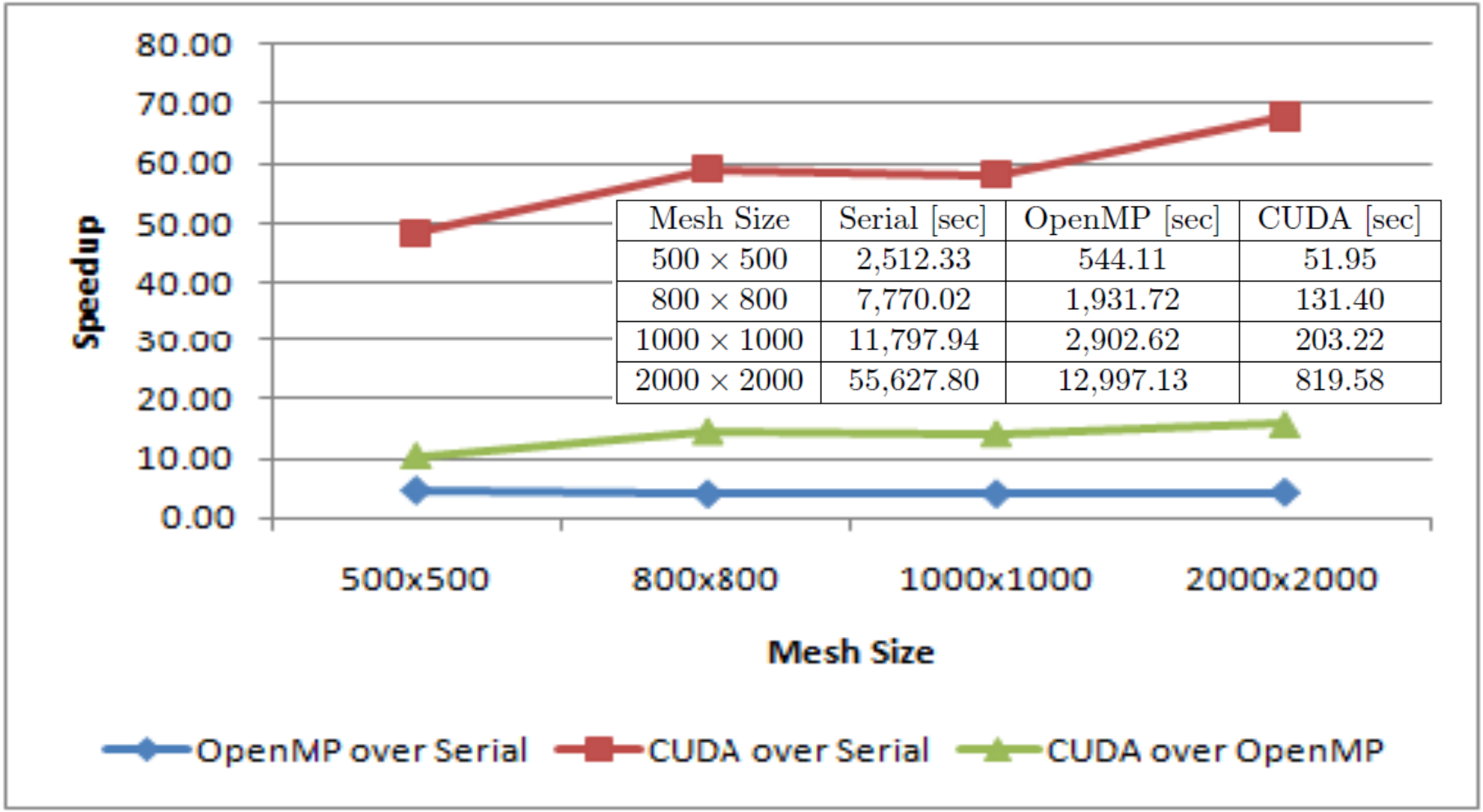
- Serial vs. 8-threaded OpenMP vs. CUDA
- Double-precision arithmetic
  - Apart from the metric tensor field – single precision is enough
- Workstation hosting:
  - Two Intel Clovertown quad-core Xeon X5355 (2.66GHz)
  - One Tesla C2050 graphics board (Fermi architecture).
  - Environment:
    - » Ubuntu Server, running kernel 2.6.32-24-server x86_64
    - » gcc 4.4.3
    - » CUDA SDK 3.1 and nVIDIA Compilation Tools 3.1, V0.2.1221
    - » nVIDIA ForceWare driver, version 256.40

Imperial College London

Experimental results

- Register usage and warp occupancy:
  - Structured metric tensor field – non-textured: 59 registers, 33.3% occupancy
  - Structured metric tensor field – textured: 51 registers, 37.5% occupancy
  - In both cases, occupancy is noticeably high
    - » Leaving as future work the break-down of the CUDA kernel into smaller parts.
- Best execution configuration was found to be:
  - 32 threads per CUDA block for the non-textured version.
  - 16 threads per CUDA block for the textured version.
  - Using more threads per block has a negative impact on performance, indicating that memory access latency is significant
    - » Leaving as future work the partitioning of the mesh in an effort to improve data locality.

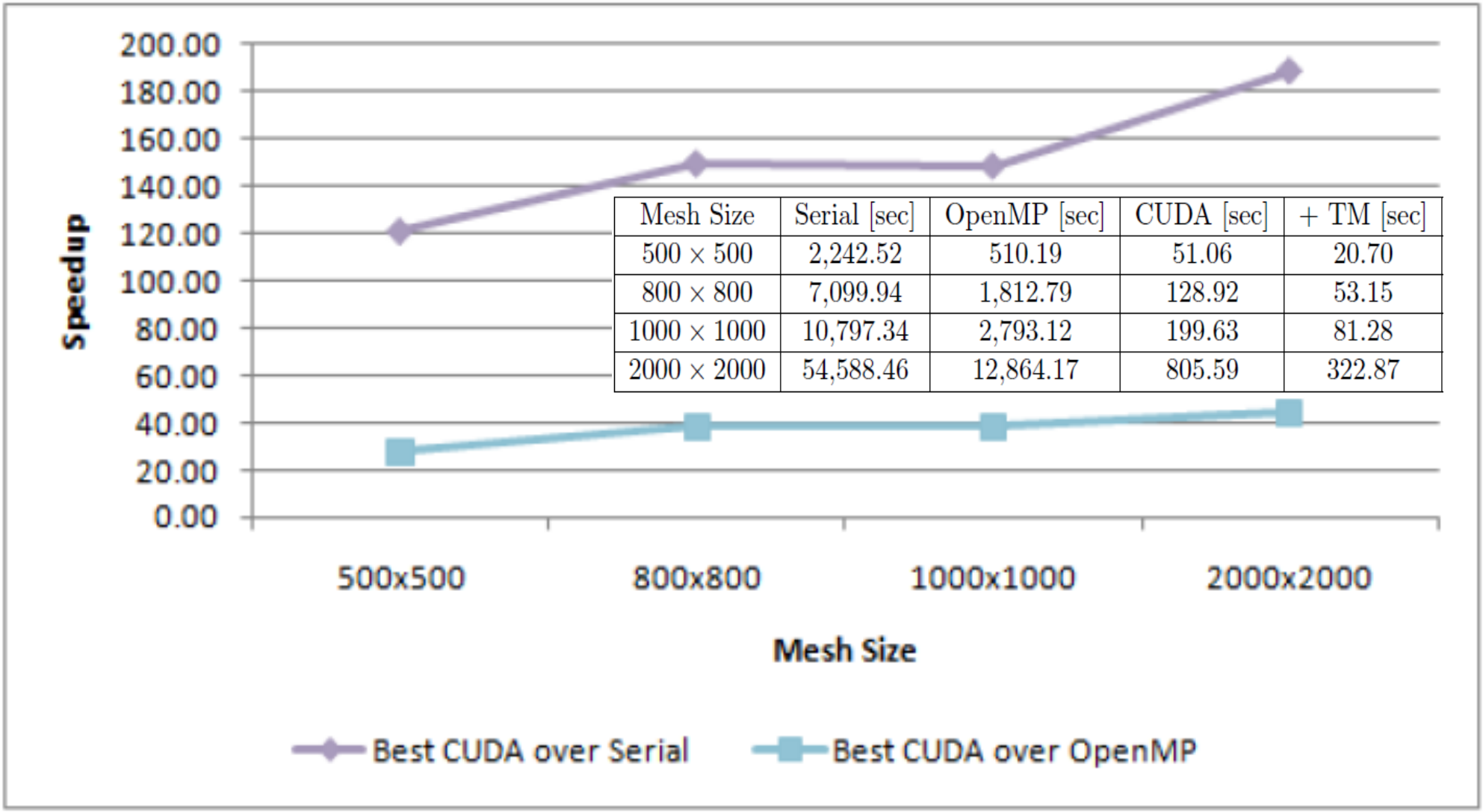
Speedup diagrams – non-textured approach



Mesh Size	Serial [sec]	OpenMP [sec]	CUDA [sec]
500 × 500	2,512.33	544.11	51.95
800 × 800	7,770.02	1,931.72	131.40
1000 × 1000	11,797.94	2,902.62	203.22
2000 × 2000	55,627.80	12,997.13	819.58

- Serial vs. 8-threaded OpenMP vs. non-textured CUDA
  - The 8-threaded OpenMP version is approx. x5 faster than the serial version
  - The CUDA version is up to x16 faster than the OpenMP version.
  - The CUDA version is up to x68 faster than the serial version.
    - » Highest speedups are observed when running really large problems.

Speedup diagrams – textured approach



Mesh Size	Serial [sec]	OpenMP [sec]	CUDA [sec]	+ TM [sec]
500 × 500	2,242.52	510.19	51.06	20.70
800 × 800	7,099.94	1,812.79	128.92	53.15
1000 × 1000	10,797.34	2,793.12	199.63	81.28
2000 × 2000	54,588.46	12,864.17	805.59	322.87

- Serial vs. 8-threaded OpenMP vs. textured CUDA
  - Using the texture approach speeds up the CUDA version by up to x2.5!
  - The optimized CUDA version is up to x45 faster than the OpenMP version.
  - The optimised CUDA version is up to x190 faster than the serial version.

Imperial College London

Conclusions and future work

- This first attempt to implement anisotropic mesh adaptivity on CUDA showed that great benefits can still be obtained, despite the unstructured nature of the problem.
- Speedup can be as high as x190 compared to the serial CPU code and up to x45 compared to an eight-threaded OpenMP code.
- Treating the metric tensor field as a graphics texture has more than doubled CUDA performance.
- Breakdown of this CUDA kernel into smaller parts and partitioning to improve data locality are left as future work.
- Porting the application to OpenCL would offer a testing framework to assess OpenCL compilers and also compare CUDA vs. Stream vs. CBE.
- Porting the codebase to OP2 to assess ease of coding and achieved performance (a task which is already in progress).
- The full work along with references to publications for further reading can be obtained from my web page:

http://www.doc.ic.ac.uk/~gr409/