TNL: Framework for numerical computing on modern parallel architectures

Tomáš Oberhuber Jakub Klinkovský Radek Fučík
Vítězslav Žabka Vladimír Klement Vít Hanousek
Aleš Wodecki

Department of Mathematics,
Faculty of Nuclear Sciences and Physical Engineering,
Czech Technical University in Prague

CSASC 2018, Bratislava

Why GPU?

	Nvidia V100	Intel Xeon E5-4660
Cores	5120 @ 1.3GHz	16 @ 3.0GHz
Peak perf.	15.7/7.8 TFlops	0.4 / 0.2 TFlops
Max. RAM	16 GB	1.5 TB
Memory bw.	900 GB/s	68 GB/s
TDP	300 W	120 W

$\approx 8,000$ \$

Difficulties in programming GPUs?

Unfortunately,

- the programmer must have good knowledge of the hardware
- porting a code to GPUs often means rewriting the code from scratch
- lack of support in older numerical libraries

Numerical libraries which makes GPUs easily accessible are being developed.

Template Numerical Library

TNL = Template Numerical Library

- is written in C++ and profits from meta-programming
- provides unified interface to multi-core CPUs and GPUs (via CUDA)
- wants to be user friendly
- www.tnl-project.org
- MIT license

Arrays and vectors

Arrays are basic structures for memory management

- `TNL::Array< ElementType, DeviceType, IndexType >`
- `DeviceType` says where the array resides
  - `TNL::Devices::Host` for CPU
  - `TNL::Devices::Cuda` for GPU
- memory allocation, I/O operations, elements manipulation ...

Vectors extend arrays with algebraic operations

- `TNL::Vector< RealType, DeviceType, IndexType >`
- addition, multiplication, scalar product, l_p norms ...

Matrix formats

TNL supports the following matrix formats (on both CPU and GPU):

- dense matrix format
- tridiagonal and multidiagonal matrix format
- Ellpack format
- CSR format
- SlicedEllpack format
- ChunkedEllpack format

Oberhuber T., Suzuki A., Vacata J., *New Row-grouped CSR format for storing the sparse matrices on GPU with implementation in CUDA*, Acta Technica, 2011, vol. 56, no. 4, pp. 447-466.

Heller M., Oberhuber T., *Improved Row-grouped CSR Format for Storing of Sparse Matrices on GPU*, Proceedings of Algoritmy 2012, 2012, Handlovičová A., Minarechová Z. and Ševčovič D. (ed.), pages 282-290.

Numerical meshes

Numerical mesh consists of *mesh entities* referred by their dimension:

	Mesh entity dimension			
Mesh dimension	0	1	2	3
1	vertex	cell	–	–
2	vertex	face	cell	–
3	vertex	edge	face	cell

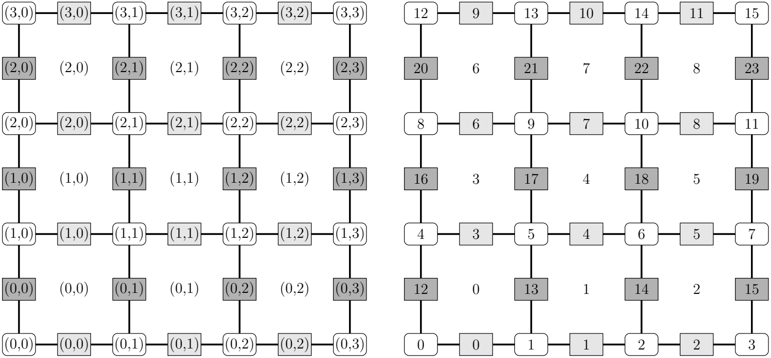
Numerical meshes

TNL supports

- structured orthogonal **grids** – 1D, 2D, 3D
  - mesh entities are generated on the fly
- unstructured **meshes** – nD
  - mesh entities are stored in memory

Structured grids

`TNL::Meshes::Grid< Dimensions,Real,Device,Index >`



Grid provides mapping between coordinates and global indexes.

Unstructured meshes

Unstructured numerical mesh is defined by:

- set of vertexes, cells, faces (and edges)
- coordinates of the vertexes
- each mesh entity may store subentities and superentities
  - see the next slide

The mesh does not store:

- mesh entity volume
- mesh entity normal
- etc.

Unstructured meshes

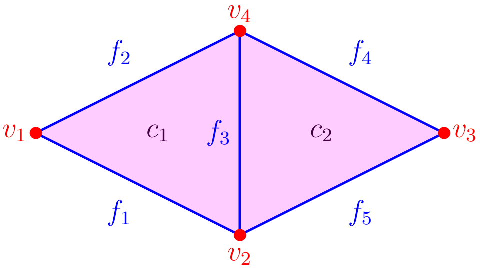
Subentities = mesh entities adjoined to another mesh entity with **higher** dimension

- faces adjoined to cell
- vertexes adjoined to cell
- ...

Superentities = mesh entities adjoined to another mesh entity with **lower** dimension

- cells adjoined to vertex
- cells adjoined to face
- faces adjoined to vertex
- ...

Unstructured meshes



$$I_{0,1} = \left(\begin{array}{c|ccccc} & f_1 & f_2 & f_3 & f_4 & f_5 \\ \hline v_1 & 1 & 1 & & & \\ v_2 & 1 & & 1 & & 1 \\ v_3 & & & & 1 & 1 \\ v_4 & & 1 & 1 & 1 & \end{array} \right) \quad I_{0,2} = \left(\begin{array}{c|cc} & c_1 & c_2 \\ \hline v_1 & 1 & \\ v_2 & 1 & 1 \\ v_3 & & 1 \\ v_4 & 1 & 1 \end{array} \right)$$

Unstructured meshes

`TNL::Meshes::Mesh< MeshConfig, Device >`

- can have arbitrary dimension
- `MeshConfig` controls what mesh entities, subentities and superentities are stored
- it is done in the compile-time thanks to C++ templates

Based on `MeshConfig`, the mesh is fine-tuned for specific numerical method in compile-time.

Solvers

ODEs solvers

- Euler, Runge-Kutta-Merson

Linear systems solvers

- Krylov subspace methods (CG, BiCGSTab, GMRES, TFQMR)

Oberhuber T., Suzuki A., Žabka V., *The CUDA implementation of the method of lines for the curvature dependent flows*, Kybernetika, 2011, vol. 47, num. 2, pp. 251–272.

Oberhuber T., Suzuki A., Vacata J., Žabka V., *Image segmentation using CUDA implementations of the Runge-Kutta-Merson and GMRES methods*, Journal of Math-for-Industry, 2011, vol. 3, pp. 73–79.

The heat eqaution

We solve the heat equation problem

$$\begin{aligned}\frac{\partial u(\mathbf{x},t)}{\partial t} - \Delta u(\mathbf{x},t) &= f(\mathbf{x},t) \quad \text{on } \Omega \times (0,T], \\ u(\mathbf{x},0) &= u_{ini}(\mathbf{x}) \quad \text{on } \Omega, \\ u(\mathbf{x},t) &= g(\mathbf{x},t) \quad \text{on } \partial\Omega \times (0,T].\end{aligned}$$

in 1D, 2D and 3D on time interval $[0,1]$ using both explicit and implicit numerical scheme.
Numerical simulations were performed on:

- 6-core CPU Intel i7-5820K at 3.3 GHz with 15 MB cache
- GPU Tesla K40 with 2880 CUDA cores at 0.745 GHz

Explicit scheme for the heat equation in 2D

	CPU							GPU	
DOFs	1 core	2 cores		4 cores		8 cores			
16^2	0.003	**0.002**	0.75	0.003	0.25	0.005	0.07	0.04	**0.05**
32^2	**0.011**	0.013	0.42	0.016	0.17	0.022	0.06	0.16	**0.07**
64^2	0.11	**0.08**	0.68	0.085	0.32	0.11	0.12	0.64	**0.12**
128^2	1.67	0.92	0.92	0.64	0.65	**0.58**	0.35	2.76	**0.21**
256^2	24.22	13.0	0.93	7.51	0.80	**4.84**	0.62	13.8	**0.35**
512^2	380	196.4	0.96	102.2	0.93	**56.6**	0.83	98.2	**0.57**
1024^2	8786	4590	0.95	2864	0.76	**2273**	0.48	1060.1	**2.14**
2048^2	149227	78283	0.95	46633	0.80	**40976**	0.45	14882.4	**2.75**

Explicit scheme for the heat equation in 3D

DOFs	CPU							GPU	
	1 core	2 cores		4 cores		8 cores			
	Time	Time	Eff.	Time	Eff.	Time	Eff.	Time	Speed-up
16^3	0.02	**0.01**	1.0	0.011	0.45	0.012	0.20	0.06	**0.16**
32^3	0.49	0.27	0.90	**0.17**	0.72	0.17	0.36	0.32	**0.53**
64^3	14.6	7.82	0.93	4.46	0.81	**2.7**	0.67	2.77	**0.97**
128^3	584.8	312.2	0.93	187.4	0.78	**142.1**	0.51	58.8	**2.64**
256^3	18425	9632	0.95	5648	0.81	**5523**	0.41	1793.1	**3.08**

Adding arithmetic intensity

$$Arithmetic\ intensity = \frac{number\ of\ operations}{transferred\ bytes}$$

- the arithmetic intensity is very low for the heat equation
- we will increase it by computationally intensive right-hand side

- in 2D

$$f(\mathbf{x},t) = \cos(t)\left(\frac{-2a}{\sigma^2}e^{\frac{-x^2-y^2}{\sigma^2}} + \frac{4ax^2}{\sigma^4}e^{\frac{-x^2-y^2}{\sigma^2}} + \frac{-2a}{\sigma^2}e^{\frac{-x^2-y^2}{\sigma^2}} + \frac{4ay^2}{\sigma^4}e^{\frac{-x^2-y^2}{\sigma^2}}\right)$$

- in 3D

$$\begin{aligned} f(\mathbf{x},t) &= \cos(t)\left(\frac{-2a}{\sigma^2}e^{\frac{-x^2-y^2-z^2}{\sigma^2}} + \frac{4ax^2}{\sigma^4}e^{\frac{-x^2-y^2-z^2}{\sigma^2}} + \frac{-2a}{\sigma^2}e^{\frac{-x^2-y^2-z^2}{\sigma^2}} + \right. \\ &\left. \frac{4ay^2}{\sigma^4}e^{\frac{-x^2-y^2-z^2}{\sigma^2}} + \frac{-2a}{\sigma^2}e^{\frac{-x^2-y^2-z^2}{\sigma^2}} + \frac{4az^2}{\sigma^4}e^{\frac{-x^2-y^2-z^2}{\sigma^2}}\right) \end{aligned}$$

Explicit scheme for the intensive heat equation in 2D

DOFs	CPU							GPU	
	1 core	2 cores		4 cores		8 cores			
	Time	Time	Eff.	Time	Eff.	Time	Eff.	Time	Speed-up
16^2	0.008	0.004	100%	0.004	50%	**0.004**	25%	0.01	**0.4**
32^2	0.08	0.04	100%	0.03	66%	**0.02**	50%	0.03	**0.7**
64^2	1.1	0.58	94 %	0.32	85%	**0.2**	68%	0.12	**1.7**
128^2	18.9	9.7	97 %	5.0	94%	**2.7**	87%	0.61	**4.4**
256^2	307	154.8	99 %	79	97%	**40.7**	94%	3.8	**10.7**
512^2	4955	2484	99 %	1258	98%	**635.3**	97%	37.1	**17**
1024^2	80377	40250	99 %	20308	98%	**15307**	65%	480.6	**32**
2048^2	1288961	645441	99 %	325901	98%	**255931**	62%	7248	**35**

Explicit scheme for the intensive heat equation in 3D

DOFs	CPU							GPU	
	1 core	2 cores		4 cores		8 cores			
	Time	Time	Eff.	Time	Eff.	Time	Eff.	Time	Speed-up
16^3	0.11	0.07	78%	0.05	55%	**0.03**	45%	0.01	**3**
32^3	4.1	2.17	94%	1.24	82%	**0.74**	69%	0.08	**9.2**
64^3	149.5	76.2	98%	40.7	91%	**21.9**	85%	1.41	**15.5**
128^3	5101	2581	98%	1391	91%	**1028**	62%	40.1	**25.6**
256^3	169138	85621	98%	44173	95%	**28172**	75%	1265	**22.3**

Implicit scheme for the heat equation in 2D

DOFs	CPU							GPU	
	1 core	2 cores		4 cores		8 cores			
	Time [s]	Time [s]	Eff.	Time [s]	Eff.	Time [s]	Eff.	Time [s]	Speed-up
16^2	**0.006**	0.015	20%	0.018	8%	0.03	2 %	0.04	**0.15**
32^2	**0.05**	0.078	32%	0.085	14%	0.12	5 %	0.18	**0.27**
64^2	0.38	0.28	67%	**0.26**	36%	0.34	13 %	0.49	**0.53**
128^2	4.17	2.1	99%	1.5	69%	**1.4**	37 %	1.43	**0.97**
256^2	55.3	23.9	115%	13.7	101%	**11.6**	59 %	6.01	**1.93**
512^2	842.5	466.2	90%	262.4	80%	**150.9**	69 %	43.1	**3.5**
1024^2	13936	7828	89%	4294	81%	**2486**	70 %	486.9	**5.1**
2048^2	248910	143353	86%	84847	73%	**64206**	48 %	7006.2	**9.1**

Implicit scheme for the heat equation in 3D

DOFs	CPU							GPU	
	1 core	2 cores		4 cores		8 cores			
	Time [s]	Time [s]	Eff.	Time [s]	Eff.	Time [s]	Eff.	Time [s]	Speed-up
16^3	0.06	0.05	60%	0.04	37%	**0.05**	15%	0.06	**0.83**
32^3	1.59	0.77	103%	0.56	70%	**0.38**	52%	0.29	**1.3**
64^3	35.3	15.13	116%	9.9	89%	**6.1**	72%	1.9	**3.2**
128^3	919.9	529	86%	310.3	74%	**217.8**	52%	30.9	**7.0**
256^3	23113	12961	89%	7768	74%	**5573**	51%	753.4	**7.4**

Explicit scheme for the heat equation with MPI

Anselm HPC Cluster

- IT4I Ostrava
- 209 nodes
- 16 cores (2x E5-2665 @ 2.4GHz)
- RAM 64GB (96GB)
- InfiniBand QDR
- (23x NVIDIA Kepler K20)

Explicit scheme for the heat equation with MPI

Mesh size: 8192 x 8192 (512MB)
1 MPI process per cpu socket / 16 OpenMP threads

Nodes	MPI	distr	time [s]	speedup	Efficiency [%]
0,5	1	1-1	68,5	1,00	**100**
1	2	2-1	35,1	1,95	**98**
2	4	2-2	18	3,8	**95**
4	8	4-2	9,9	6,9	**86**
8	16	4-4	5,2	13	**82**
16	32	8-4	2,7	25	**79**
32	64	8-8	0,798	85	**134**

Multiphase flow in porous media

We consider the following system of n partial differential equations in a general coefficient form

$$\sum_{j=1}^{n} N_{i,j} \frac{\partial Z_j}{\partial t} + \sum_{j=1}^{n} \mathbf{u}_{i,j} \cdot \nabla Z_j$$
$$+ \nabla \cdot \left[m_i \left(- \sum_{j=1}^{n} D_{i,j} \nabla Z_j + \mathbf{w}_i \right) + \sum_{j=1}^{n} Z_j \mathbf{a}_{i,j} \right] + \sum_{j=1}^{n} r_{i,j} Z_j = f_i$$

for $i = 1, ..., n$, where the unknown vector function $\vec{Z} = (Z_1, ..., Z_n)^T$ depends on position vector $\vec{x} \in \Omega \subset \mathbb{R}^d$ and time $t \in [0, T]$, $d = 1, 2, 3$.

Multiphase flow in porous media

Initial condition:

$$Z_j(\vec{x},0) = Z_j^{ini}(\vec{x}), \quad \forall \vec{x} \in \Omega,\ j = 1,\dots,n,$$

Boundary conditions:

$$\begin{aligned} Z_j(\vec{x},t) &= Z_j^{\mathcal{D}}(\vec{x},t), \quad \forall \vec{x} \in \Gamma_j^{\mathcal{D}} \subset \partial\Omega,\ \ j = 1,...,n, \\ \vec{v}_i(\vec{x},t) \cdot \vec{n}_{\partial\Omega}(\vec{x}) &= v_i^{\mathcal{N}}(\vec{x},t), \quad \forall \vec{x} \in \Gamma_i^{\mathcal{N}} \subset \partial\Omega,\ \ i = 1,...,n, \end{aligned}$$

where $\vec{v}_i$ denotes the conservative velocity term

$$\vec{v}_i = -\sum_{j=1}^{n} \mathbf{D}_{i,j} \nabla Z_j + \mathbf{w}_i.$$

Numerical method

- Based on the mixed-hybrid finite element method (MHFEM)
  - one global large sparse linear system for traces of $(Z_1, , \dots, Z_n)$ (on faces) per time step
- Semi-implicit time discretization
- General spatial dimension (1D, 2D, 3D)
- Structured and unstructured meshes

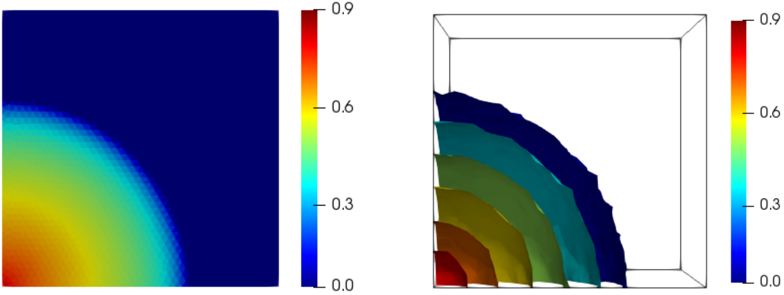
R. Fučík, J.Klinkovský, T. Oberhuber, J. Mikyška, *Multidimensional Mixed–Hybrid Finite Element Method for Compositional Two–Phase Flow in Heterogeneous Porous Media and its Parallel Implementation on GPU*, submitted to Computer Physics Communications.

McWhorter–Sunada problem

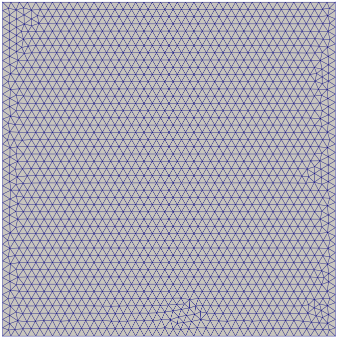
Benchmark problem – generalization of the McWhorter–Sunada problem

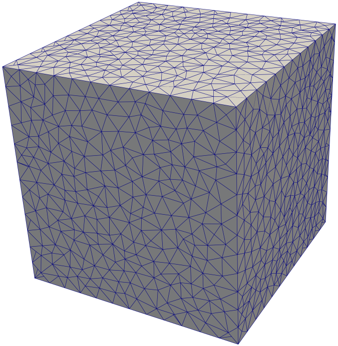
- Two phase flow in porous media
- General dimension (1D, 2D, 3D)
- Radial symmetry
- Point injection in the origin
- Incompressible phases and neglected gravity
- Semi-analytical solution by McWhorter and Sunada (1990) and Fučík *et al.* (2016)

McWhorter–Sunada problem



McWhorter–Sunada problem





McWhorter–Sunada problem

Numerical simulations were performed on:

- 6-core CPU Intel i7-5820K at 3.3 GHz with 15 MB cache
- GPU Tesla K40 with 2880 CUDA cores at 0.745 GHz

McWhorter–Sunada problem 2D

	GPU	CPU										
		1 core		2 cores			4 cores			6 cores		
DOFs	*CT*	*CT*	*GSp*	*CT*	*Eff*	*GSp*	*CT*	*Eff*	*GSp*	*CT*	*Eff*	*GSp*
Orthogonal grids												
960	1,5	0,7	**0,45**	0,4	0,79	**0,28**	0,3	0,52	**0,22**	0,3	0,41	**0,18**
3 720	11,0	13,2	**1,20**	7,6	0,87	**0,69**	4,8	0,68	**0,44**	4,0	0,55	**0,37**
14 640	46,3	197,0	**4,25**	107,5	0,92	**2,32**	65,7	0,75	**1,42**	52,6	0,62	**1,14**
58 080	380,0	4 325,7	**11,38**	2 360,6	0,92	**6,21**	1 448,1	0,75	**3,81**	1 195,8	0,60	**3,15**
231 360	4 449,9	91 166,3	**20,49**	49 004,3	0,93	**11,01**	29 182,1	0,78	**6,56**	24 684,0	0,62	**5,55**
Unstructured meshes												
766	1,5	0,4	**0,27**	0,3	0,60	**0,22**	0,2	0,45	**0,15**	0,2	0,32	**0,14**
2 912	8,9	6,2	**0,70**	3,7	0,84	**0,42**	2,3	0,66	**0,26**	2,0	0,52	**0,23**
11 302	51,1	122,0	**2,39**	67,7	0,90	**1,32**	40,3	0,76	**0,79**	32,5	0,63	**0,64**
44 684	396,1	2 695,6	**6,80**	1 480,7	0,91	**3,74**	855,2	0,79	**2,16**	671,7	0,67	**1,70**
178 648	4 008,3	57 404,2	**14,32**	32 100,5	0,89	**8,01**	18 814,1	0,76	**4,69**	16 414,0	0,58	**4,09**

McWhorter–Sunada problem 3D

	GPU	CPU										
		1 core		2 cores			4 cores			6 cores		
DOFs	*CT*	*CT*	*GSp*	*CT*	*Eff*	*GSp*	*CT*	*Eff*	*GSp*	*CT*	*Eff*	*GSp*
Orthogonal grids												
21 600	2,1	15,2	**7,30**	8,0	0,96	**3,82**	4,4	0,86	**2,13**	3,4	0,75	**1,62**
167 400	30,8	564,3	**18,33**	319,5	0,88	**10,38**	186,7	0,76	**6,07**	150,3	0,63	**4,88**
1 317 600	828,0	20 569,5	**24,84**	12 406,1	0,83	**14,98**	7 092,6	0,73	**8,57**	5 533,7	0,62	**6,68**
10 454 400	31 805,6	(not computed on 1, 2 and 4 cores)								234 066,0		7,36
Unstructured meshes												
5 874	1,4	2,0	**1,48**	1,2	0,85	**0,88**	0,7	0,68	**0,54**	0,6	0,54	**0,46**
15 546	2,6	8,7	**3,30**	4,9	0,89	**1,85**	2,9	0,75	**1,10**	2,3	0,64	**0,86**
121 678	23,9	330,9	**13,87**	184,8	0,90	**7,75**	107,9	0,77	**4,53**	93,4	0,59	**3,92**
973 750	566,2	12 069,5	**21,32**	6 506,3	0,93	**11,49**	3 771,0	0,80	**6,66**	3 306,2	0,61	**5,84**
7 807 218	37 695,3	(not computed on CPU)										

Conclusion

We have presented:

- data structures and solvers in TNL
- unstructured meshes
- MHFEM method for multiphase flow in porous media on GPU
- speed-up on the GPU is up to 7

Experimental features

- **support of distributed clusters using MPI and clusters with GPUs**
  - V. Hanousek
  - J. Klinkovský
- **lattice Boltzmann method**
  - SRT, MRT, CLBM
  - R. Fučík, P. Eichler, J. Klinkovský
- nD arrays
  - J. Klinkovský
- FVM on structured grids
  - J. Schafer
- Hamilton-Jacobi equations on GPUs
  - M. Fencl

Future plans

- **adaptive numerical grids**
  - A. Wodecki
- FEM, FVM
- geometric and algebraic multigrid
- documentation

More about TNL ...

TNL is available at

`www.tnl-project.org`

under MIT license.

Implicit scheme for the heat equation in 3D

Time spent in particular parts of the implicit solver

	GPU		GPU		
	Time	Ratio	Time	Ratio	GSp
Lin. System assembly	844	1.3%	204	2.9%	**4.13**
Lin. System solution	63361	98.7%	6802	97.1%	**9.31**
Total	64197		7006		**9.16**