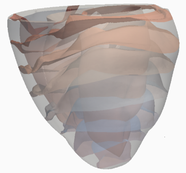
Parallel Low Memory Footprint Eikonal Solver in Cardiovascular Applications

Eikonal equation

Mathematical Description

The electrophysiology of a human heart describes the expansion of an electrical stimulation therein and it is usually modeled by the bidomain equations (2 time-dependent PDEs + nonlinear ODE system).



- Instead of the full transmembrane potential some cardiovascular applications require only the arrival time of the excitation wave in different heart regions.
- In this case we can apply several model simplifications that lead to the Eikonal equation which is a special case of non-linear Hamilton–Jacobi partial differential equations:

$$\begin{cases} H(x, \nabla\phi) = \sqrt{(\nabla\phi)^T M(\nabla\phi)} = 1, & \forall x \in \Omega \subset \mathbb{R}^3 \\ \phi(x) = B(x), & \forall x \in \beta \subset \Omega \end{cases}$$

where Ω is an open set in $\mathbb{R}^n$ with well-behaved boundary, $M(x)$ is a 3x3 symmetric positive-definite matrix encoding the speed information on Ω and β the boundary conditions.

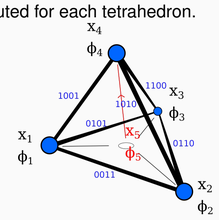
- The solution $\phi(x)$ is the shortest time needed to travel from the boundary $\partial\Omega$ to x inside Ω.

Eikonal Solver

We use the fast iterative method (FIM) by Fu, Kirby and Whitaker [2] as baseline Eikonal solver for our improvements in our research [1]. It is based on the local solver computed for each tetrahedron.

- An upwind scheme is used to compute an unknown solution ϕ_4, assuming ϕ_1, ϕ_2, ϕ_3 are known.
- The goal is to find the location of x_5 which minimizes the travel time from x_5 to x_4, $e_{5,4} = x_4 - x_5$, (Fermat's principle) i.e.,

$$\phi_4(\lambda_1, \lambda_2) = \lambda_1\phi_1 + \lambda_2\phi_2 + (1 - \lambda_1 - \lambda_2)\phi_3 + \sqrt{e_{5,4}^T M e_{5,4}} \ .$$



Minimizing ϕ_4 requires solving (1) with $\lambda = [\lambda_1\ \lambda_2\ 1]^T$.

$$(1) \quad \begin{cases} \phi_{1,3}\sqrt{\lambda^T M'\lambda} &= \lambda^T \underline{\alpha} \\ \phi_{2,3}\lambda^T \underline{\alpha} &= \phi_{1,3}\lambda^T \underline{\beta} \end{cases} . \qquad \text{where } M' = \begin{cases} [e_{1,3}^T M e_{1,3},\ e_{2,3}^T M e_{1,3},\ e_{3,4}^T M e_{1,3}]^T \\ [e_{1,3}^T M e_{2,3},\ e_{2,3}^T M e_{2,3},\ e_{3,4}^T M e_{2,3}]^T \\ [e_{1,3}^T M e_{3,4},\ e_{2,3}^T M e_{3,4},\ e_{3,4}^T M e_{3,4}]^T \end{cases} .$$

Solving (1) requires the coordinates of x_1, x_2, x_3, x_4 and the symmetric matrix M.

- $(4 \times 3 + 6)$ floats from global memory + local storage for $e_{i,j}$ + inner products.

Low memory footprint solver - Gray Code

Gray Code Like Indexing (Fig. 2)

x_4	x_3	x_2	x_1	edge: d	$f(d) = d/2 - 1$
0	0	1	1	3	0
0	1	0	1	5	1
0	1	1	0	6	2
1	0	0	1	9	3
1	0	1	0	10	4
1	1	0	0	12	5

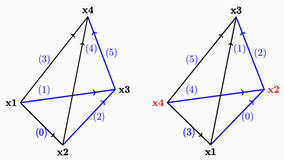
M-Scalar Products Precalculated and Stored in the 6x6 Symmetric Matrix T_M

T_M	0	1	2	...	5
0	$e_{1,2}Me_{1,2}$	$e_{1,2}Me_{1,3}$	$e_{1,2}Me_{2,3}$	...	$e_{1,2}^T Me_{3,4}$
1		$e_{1,3}Me_{1,3}$	$e_{1,3}Me_{2,3}$	...	$e_{1,3}Me_{3,4}$
2			$e_{2,3}Me_{2,3}$	...	$e_{2,3}Me_{3,4}$
...			...	...	...
5					$e_{3,4}Me_{3,4}$

Algorithm 1 Accessing M' wrt. common vertex j

1: Get Gray-code edge numbers $d_0(j), d_1(j), d_2(j)$
2: Get edge indices $k_i := \mathbf{f}_{\text{int}}(d_i) \qquad i = 0, 1, 2$
3: $M' := T_M(\underline{k}, \underline{k})$ ▷ extract 3×3 matrix

Rotating Elements into the Reference Configuration



ϕ_k	ϕ_4	ϕ_3	ϕ_2	ϕ_1
Edges	5,1,2	2,4,0	0,1,3	3,4,5
Signs	+,+,+	+,−,+	+,−,−	−,+,+

Further Memory Footprint Reduction

- Express products using elements from the main diagonal in T_M:
  $e_{1,2}^T Me_{1,3} = \frac{1}{2}(e_{1,3}^T Me_{1,3} + e_{1,2}^T Me_{1,2} - e_{2,3}^T Me_{2,3})$
- Allows to store only the main diagonal inner products in T_M.
- The memory footprint reduces to 6 floats in total per tetrahedron.
- Less memory transfers, increased computations.
- No additional code branching.
- Can be adopted to problems of the kind.

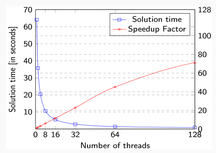
Gray code results

Tested on GeForce GTX 1080 and Intel Core i7 2.40GHz.

Implementations	# Tets	CUDA	OpenMP 8 threads
Without Gray-code	3,073,529	1.49 sec.	5.66 sec.
With Gray-code	3,073,529	0.73 sec.	3.65 sec.
Without Gray-code	24,400,999	11.48 sec.	56.63 sec.
With Gray-code	24,400,999	5.16 sec.	36.43 sec.

- CPU implementation: 33% faster.
- CUDA implementation: 50% faster.

Scaling results on KNL, Xeon Phi 1.3GHz.



- Almost linear up to 64 physical cores.
- Starts to drop with hyperthreading.

Intel's Vtune Amplifier

- Reduced non-coalesced memory accesses.
  - The number of loads dropped to 70%.
  - LLC miss count reduced to 84%.
- The number of stores dropped to 40%.

NVIDIA Visual Profiler

- High local memory overhead, 54% of total memory traffic, disappeared totally.
- The L2-bandwidth is doubled, 952.08 GB/s.
- The memory-bandwidth increased from 60% to above 90%.

Gray code results on the Nvidia shield tablet (Android)

Implementations	# Tets	CUDA	OpenMP
Without Gray-code	3,073,529	30.26 sec.	205.22 sec.
With Gray-code	3,073,529	20.30 sec.	97.19 sec.

- CPU implementation: 52% faster.
- CUDA implementation: 33% faster.

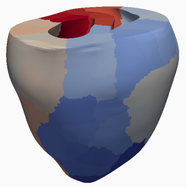
- 2.2 GHz quad ARM Cortex A15.
- Tegra K1, 192 core Kepler GPU.

- weak hardware $\Longrightarrow$ better acceleration.

Domain decomposition in CUDA

For large scale problems, the task based parallel model will run into memory difficulties. Hence a coarser decomposition of the algorithm is needed, namely a domain decomposition approach.

- The domain Ω is statically partitioned into a number of non-overlapping sub-domains Ω_i.
- Each of them is assigned to a single processor.
- Synchronization and communication of the processors is to be reduced to a minimum.
- Dynamic mapping of active subdomains to processors (GPU SMs in CUDA).



Load Balancing Strategies

SMs allocated to solve for 3 active domains.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

- **One** subdomain $\rightarrow$ **mult.** thread blocks.
- **Less** shared memory needed per thread block.
- Multiple seperate kernels: scan, scatter, compact, ...
- **Many** host synchronizations after each kernel.
- One data buffer.

SMs allocated to solve for 3 active domains.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

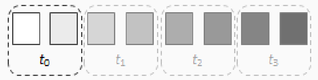
- **One** subdomain $\leftrightarrow$ **one** thread block.
- Reaches shared memory limitations per thread block.
- One **large** kernel including: scan, scatter, compact, ...
- **One** host synchronization at the termination condition.
- Dynamic memory allocation.

Data Arrangement and Granularity by CUB Library

Striped arrangement across 4 threads.

- Desirable for data movement through global memory (read/write coalescing).

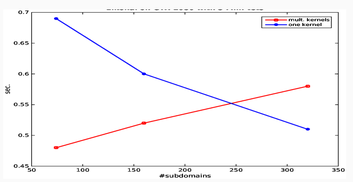
Block arrangement across 4 threads.



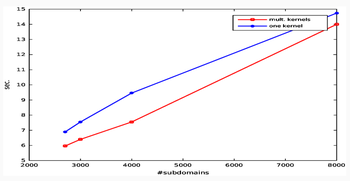
- Increased performance with the increased granularity ITEMS_PER_THREAD.

Domain decomposition results and conclusions

Eikonal DD on GTX 1080 with 3,073,529 tets.



Eikonal DD on GTX 1080 with 24,400,999 tets.



- Domain decomposition (DD) possible on a single GPU for larger examples.
- CUDA DD approach faster than without DD.
  - The reason is the shared memory version of CUB Scan algorithm.
  - For subdomains small enough the scan fits to shared memory of the block.
- One kernel approach:
  - better scaling for small example.
  - Increased granularity, take care of coalescing.
  - Much better code structure.
- Larger #subdomains $\Longrightarrow$ better load balancing.
- Poor scaling for large example.
  - Not enough GPU memory to preallocate $\Longrightarrow$ many dynamic allocations/deallocations.
- Good scaling on Titan X (8 GB memory on GTX 1080 $\rightarrow$ 12 GB on Titan X), 20% faster.

Patient-specific cardiac parametrization from Eikonal simulations

- Very high computational intensity of the bidomain equation. Impossible for the inverse problem.
- The Eikonal equation reduces significantly the computational intensity of the bidomain equation

Eikonal Equation with Material Domains

- Physiology: different heart tissues, heart chamber $\rightarrow$ m different materials.
- $\Omega \xrightarrow{tets} \overline{\Omega}_h = \bigcup_{k=1}^{m} \overline{\Omega}_k, \qquad \Omega_l \bigcap_{k \neq l} \Omega_k = \emptyset$
- $M(x)$ is constant in each tetrahedron $\tau \in \Omega_h$.
- Allow to scale $M(x)$ in each material domain Ω_k by $\gamma_k \in \mathbb{R}$.

$$\sqrt{(\nabla\phi)^T \gamma_k \cdot M \cdot (\nabla\phi)} = 1 \quad \forall x \in \Omega_k$$

- Now the excitation time $\phi(\gamma_k)$ depends on the scaling parameters $\underline{\gamma} \in \mathbb{R}^m$.

Optimization by Automatic Differentiation [3]

- Minimize

$$f(\underline{\gamma}) := \| \phi^*(\underline{x}) - \phi(\underline{\gamma}, \underline{x}) \|^2_{l^2(\omega_h)}$$

with ω_h denoting the vertices in the discretization of Ω_h and ϕ^* the measured solution.

- Steepest descent and BFGS algorithms.

Calculation of gradient $\nabla_\gamma f(\underline{\gamma})$ is done via

- automatic differentiation [dco/c++].
- Tangent and adjoint models by overloading.
- Exact gradients, up to machine accuracy.

Results and Timings

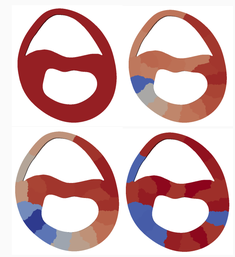
Initial : $\underline{\gamma_0} = (1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1)$.
Measured: $\underline{\gamma^*} = (1\ 1\ 1\ \ldots\ 0.2\ 0.2\ 0.2\ 0.2\ 0.2\ \ldots\ 1\ 1\ 1)$.

Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz, 252 GB RAM.

Method	Model	# Tets	1 thread	28 threads
BFGS	Tangent	3,073,529	3607 sec.	1369 sec.
BFGS	Adjoint	3,073,529	2471 sec.	678 sec.

Resulting $\underline{\gamma} = (1\ 1\ 1\ \ldots\ 0.19\ 0.2\ 0.19\ 0.2\ 0.2\ \ldots\ 1\ 1\ 1)$.

- Factor of 2.6 for the tangent and 3.6 for the adjoint model.
- Adjoint 2 times faster than Tangent already for 21 domains.



Four steps simulation of the optimization using BFGS.

Ongoing Research

1. Cluster implementation of the CUDA domain decomposition.
2. Analytic approach to compute the gradient using the Adjoint-State method.
3. Deep learning approach.

References

[1] D. Ganellari, G. Haase and G. Zumbusch, *"A massively parallel eikonal solver on unstructured meshes"*, Computing and Visualization in Science, 2018.

[2] Z. Fu, R.M. Kirby and R.T. Whitaker, *"Fast iterative method for solving the eikonal equation on tetrahedral domains"*, SIAM J. Sci. Comput., 2013.

[3] U. Naumann, *"The Art of Differentiating Computer Programs"*, SIAM, 2012.

Daniel Ganellari and Gundolf Haase

[a]Institute for Mathematics and Scientific Computing, Karl Franzens University of Graz, Austria

Supported by Erasmus Mundus JoinEUsee PENTA scholarship, FWF project F32-N18 and Mont-Blanc 3.