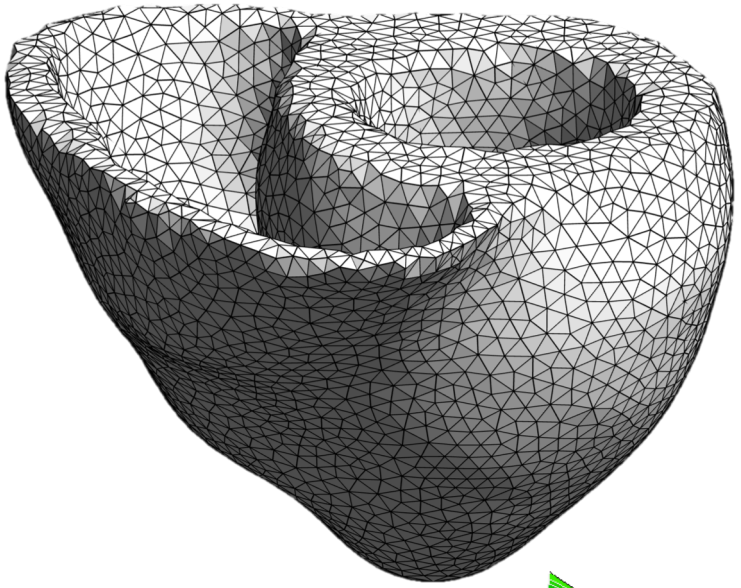
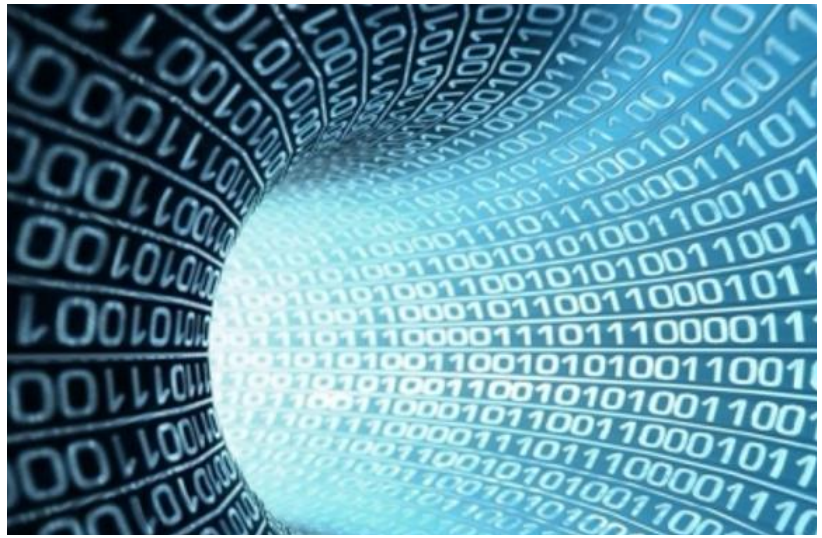
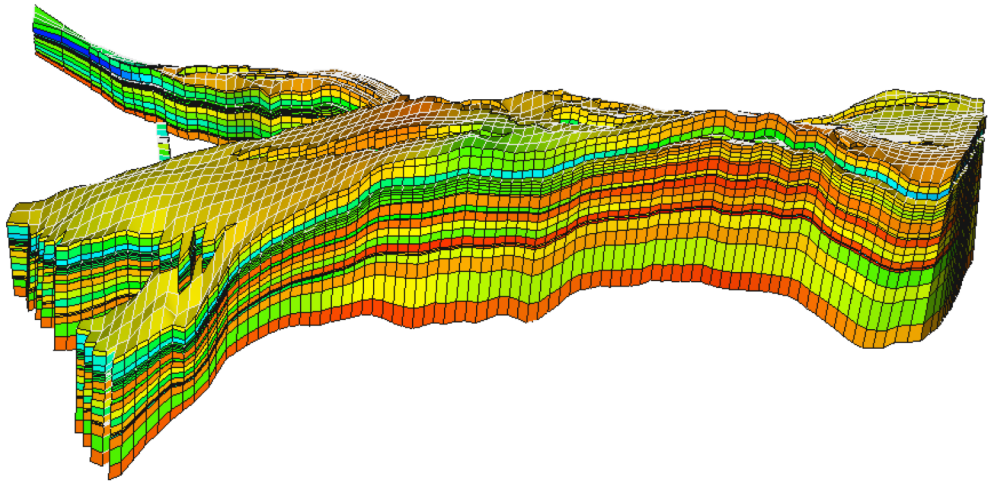
Towards High-Performance Unstructured-Mesh Computations







Xing Cai

Simula Research Laboratory

&

University of Oslo, Norway

November 29, 2023

simula

Background

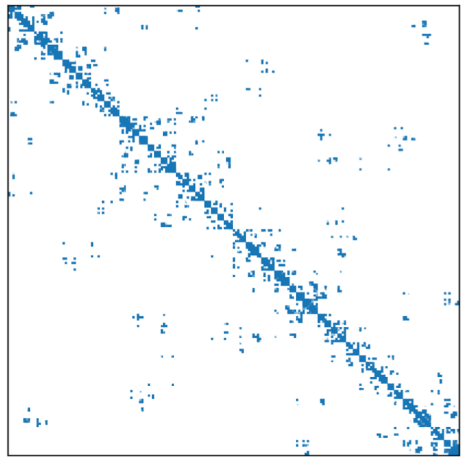
Modern processors are fast with number crunching, but the memory speeds lag behind: The "memory wall" problem

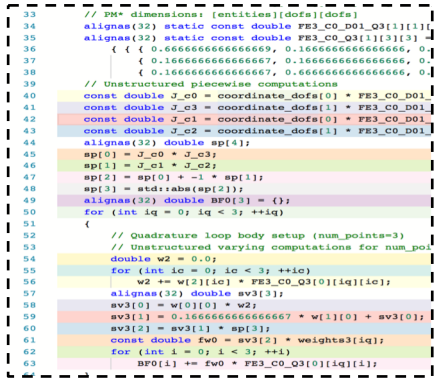
Most scientific computing applications are memory-traffic bound

Unstructured meshes are needed for modeling realistic problems, but irregular memory accesses are inevitable

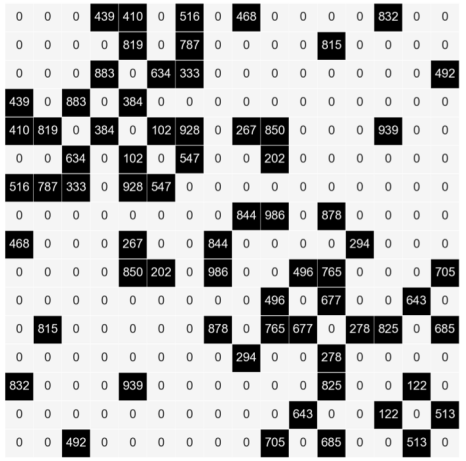
How to alleviate the challenges of achieving performance for unstructured-mesh computations?

This talk presents some recent research activities at Simula





```
// PM* dimensions: [entities][dofs][dofs]
alignas(32) static const double FE3_C0_D01_Q3[1][1][1
alignas(32) static const double FE3_C0_Q3[1][3][3] =
    { { { 0.6666666666666669, 0.1666666666666666, 0.
        { 0.1666666666666667, 0.1666666666666666, 0.
        { 0.1666666666666667, 0.6666666666666666, 0.
// Unstructured piecewise computations
const double J_c0 = coordinate_dofs[0] * FE3_C0_D01
const double J_c3 = coordinate_dofs[1] * FE3_C0_D01
const double J_c1 = coordinate_dofs[0] * FE3_C0_D01
const double J_c2 = coordinate_dofs[1] * FE3_C0_D01
alignas(32) double sp[4];
sp[0] = J_c0 * J_c3;
sp[1] = J_c1 * J_c2;
sp[2] = sp[0] + -1 * sp[1];
sp[3] = std::abs(sp[2]);
alignas(32) double BF0[3] = {};
for (int iq = 0; iq < 3; ++iq)
{
    // Quadrature loop body setup (num_points=3)
    // Unstructured varying computations for num_poi
    double w2 = 0.0;
    for (int ic = 0; ic < 3; ++ic)
        w2 += w[2][ic] * FE3_C0_Q3[0][iq][ic];
    alignas(32) double sv3[3];
    sv3[0] = w[0][0] * w2;
    sv3[1] = 0.1666666666666667 * w[1][0] + sv3[0];
    sv3[2] = sv3[1] * sp[3];
    const double fw0 = sv3[2] * weights3[iq];
    for (int i = 0; i < 3; ++i)
        BF0[i] += fw0 * FE3_C0_Q3[0][iq][i];
```



Re-ordering of mesh entities

Automated code generation for GPU computing

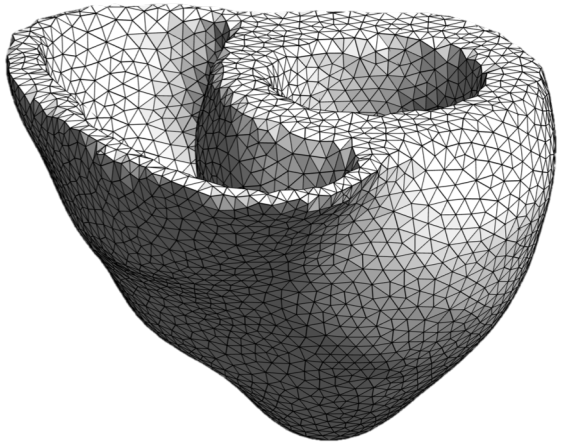
Re-purposing an ML-specific processor for unstructured mesh computation

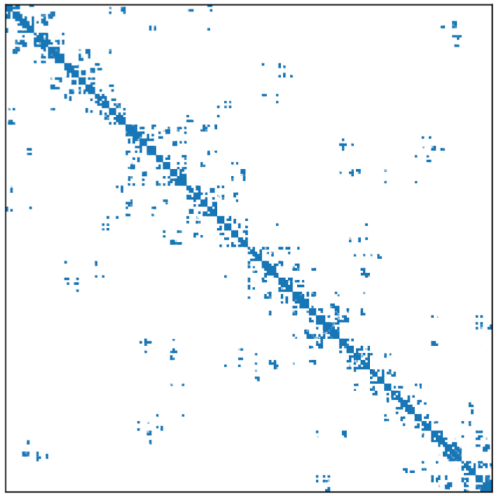
Physics-guided mesh partitioning

Detailed modeling of heterogeneous point-to-point MPI communication

Joint work with J. Trotter, A. Thune, L. Burchard, K. Hustad, J. Langguth, S. Funke, A. Rustad, S.-A. Reinemo, T. Skeie

Re-ordering of mesh entities may improve memory performance





Numerical discretization (FEM, FVM) over unstructured meshes will inevitably lead to irregular memory accesses

Proper re-ordering of the mesh entities may improve data reuse in the caches, thus reducing memory traffic

Re-ordering example 1: sparse matrix-vector multiplication

$y=Ax$

```
#pragma omp parallel for
for (int i=0; i<num_rows; i++) {
  double tmp = 0.0;
  for (int j=irows[i]; j<irows[i+1]; j++)
    tmp += A_values[j] * x[jcols[j]];
  y[i] = tmp;
}
```

Matrix A is sparse, stored in the CSR format (`irows, jcols, A_values`)

Several re-ordering strategies may improve memory performance of SpMV

- **Reverse Cuthill-McKee re-ordering**
- **Graph partitioning-based re-ordering**
- **Nested dissection**
- **Others**

Journal of Parallel and Distributed Computing 144 (2020) 189–205

Contents lists available at ScienceDirect

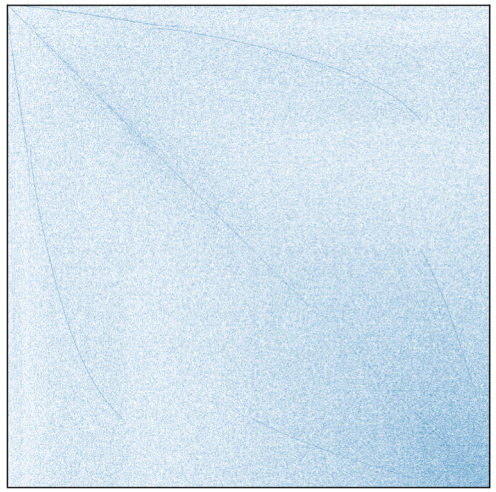
J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

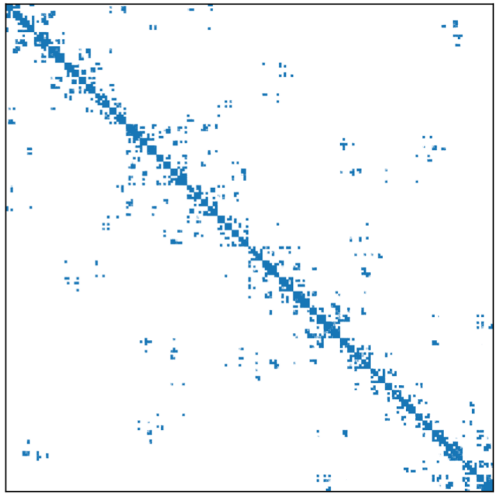
Cache simulation for irregular memory traffic on multi-core CPUs: Case study on performance models for sparse matrix–vector multiplication

James D. Trotter [a,b,*], Johannes Langguth [a], Xing Cai [a,b]

Re-ordering example 1: sparse matrix-vector multiplication



Before re-ordering



After re-ordering

	Before re-ordering	After re-ordering
SandyBridge single core	0.16 GFLOPS	0.59 GFLOPS
SandyBridge single socket	1.03 GFLOPS	3.38 GFLOPS
SandyBridge dual socket	1.98 GFLOPS	6.92 GFLOPS
SkyLake single core	0.23 GFLOPS	0.80 GFLOPS
SkyLake single socket	4.96 GFLOPS	11.65 GFLOPS
SkyLake dual socket	9.31 GFLOPS	23.33 GFLOPS

A new publication at SC23

Bringing Order to Sparsity: A Sparse Matrix Reordering Study on Multicore CPUs

Authors: James D. Trotter, Sinan Ekmekçibaşı, Johannes Langguth, Tugba Torun, Emre Düzakın, Aleksandar Ilic, Didem Unat Authors Info & Claims

SC '23: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis • November 2023 • Article No.: 31 • Pages 1–13

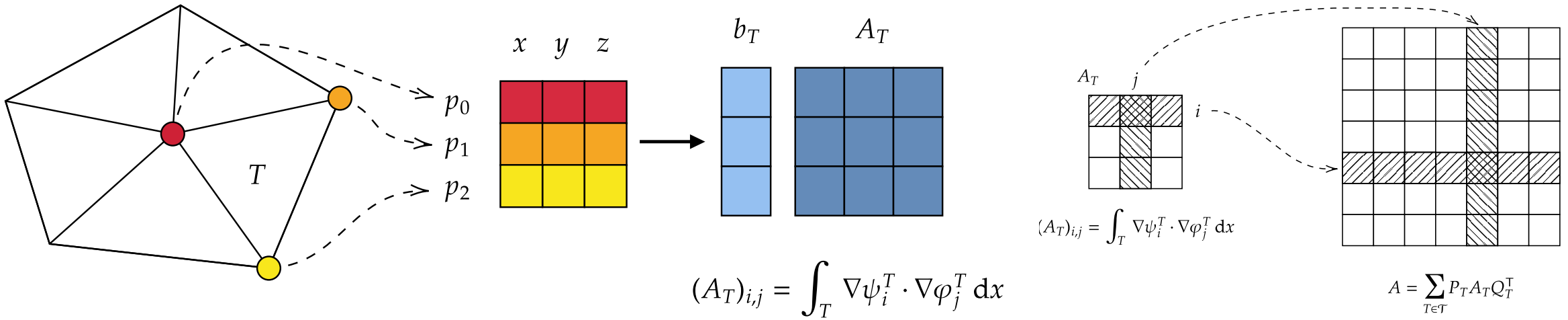
• https://doi.org/10.1145/3581784.3607046

Re-ordering example 2: finite element assembly procedure

1. Gather cell coordinates and coefficients

2. Compute element vector and matrix

3. Update global vector and matrix



1. **Reverse Cuthill-McKee re-ordering of the mesh nodes**
2. **Re-ordering of the mesh cells in ascending lexicographic order according to their node indices**

On Memory Traffic and Optimisations for Low-order Finite Element Assembly Algorithms on Multi-core CPUs

Authors: James D. Trotter, Xing Cai, Simon W. Funke Authors Info & Claims

ACM Transactions on Mathematical Software, Volume 48, Issue 2 • Article No.: 19, pp 1–31
• https://doi.org/10.1145/3503925

Re-ordering example 2: finite element assembly procedure

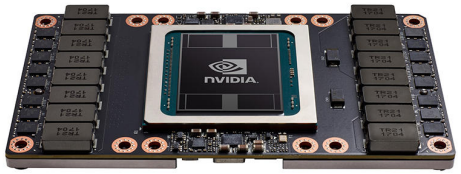
	Est. DRAM read [B/cell]		Meas. DRAM read [B/cell]	
Mesh	Best case	Worst case	Original	Reordered
Uniform mesh 1	20.1	528.0	22.0	
Uniform mesh 2	20.1	528.0	21.9	
Cardiac mesh 1	20.5	528.0	170.1	21.6
Cardiac mesh 20	20.3	528.0	250.4	22.4
Cardiac mesh 41	20.4	528.0	229.8	22.3
Cardiac mesh 44	20.4	528.0	217.9	22.0
Aneurysm mesh 3	20.0	528.0	33.7	20.6
Aneurysm mesh 4	19.9	528.0	65.2	21.2

On Memory Traffic and Optimisations for Low-order Finite Element Assembly Algorithms on Multi-core CPUs

Authors: James D. Trotter, Xing Cai, Simon W. Funke Authors Info & Claims

ACM Transactions on Mathematical Software, Volume 48, Issue 2 • Article No.: 19, pp 1–31
• https://doi.org/10.1145/3503925

Automated code generation can simplify GPU programming




```
33      // PM* dimensions: [entities][dofs][dofs]
34      alignas(32) static const double FE3_C0_D01_Q3[1][1][
35      alignas(32) static const double FE3_C0_Q3[1][3][3] =
36          { { { 0.6666666666666669, 0.1666666666666666, 0.
37              { 0.1666666666666667, 0.1666666666666666, 0.
38              { 0.1666666666666667, 0.6666666666666666, 0.
39      // Unstructured piecewise computations
40      const double J_c0 = coordinate_dofs[0] * FE3_C0_D01_
41      const double J_c3 = coordinate_dofs[1] * FE3_C0_D01_
42      const double J_c1 = coordinate_dofs[0] * FE3_C0_D01_
43      const double J_c2 = coordinate_dofs[1] * FE3_C0_D01_
44      alignas(32) double sp[4];
45      sp[0] = J_c0 * J_c3;
46      sp[1] = J_c1 * J_c2;
47      sp[2] = sp[0] + -1 * sp[1];
48      sp[3] = std::abs(sp[2]);
49      alignas(32) double BF0[3] = {};
50      for (int iq = 0; iq < 3; ++iq)
51      {
52          // Quadrature loop body setup (num_points=3)
53          // Unstructured varying computations for num_poi
54          double w2 = 0.0;
55          for (int ic = 0; ic < 3; ++ic)
56              w2 += w[2][ic] * FE3_C0_Q3[0][iq][ic];
57          alignas(32) double sv3[3];
58          sv3[0] = w[0][0] * w2;
59          sv3[1] = 0.1666666666666667 * w[1][0] + sv3[0];
60          sv3[2] = sv3[1] * sp[3];
61          const double fw0 = sv3[2] * weights3[iq];
62          for (int i = 0; i < 3; ++i)
63              BF0[i] += fw0 * FE3_C0_Q3[0][iq][i];
```

GPU programming for numerical computations can be challenging

Automated code generation can alleviate the programming challenge

Combination of high-level domain-specific language and special compiler

The FEniCS framework has used automated code generation to deploy finite element computation to CPU clusters in a user friendly way

We recently enabled automated GPU computing for FEniCS

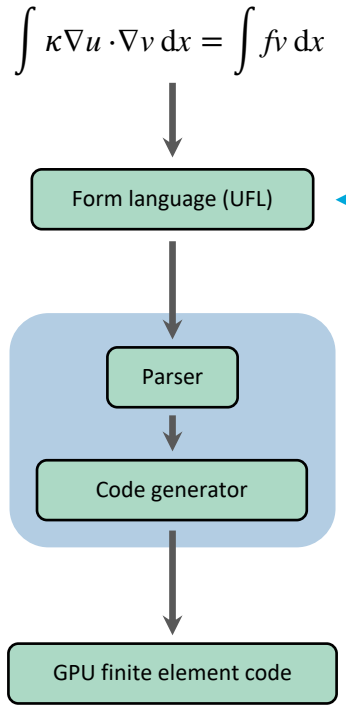
- GPU offloading of the finite element assembly procedure
- Seamless coupling with GPU-capable linear algebra backends

Automated generation of GPU-accelerated assembly code

1. Mathematical equations

$$\int \kappa \nabla u \cdot \nabla v \, dx = \int f v \, dx$$

3. Form compiler

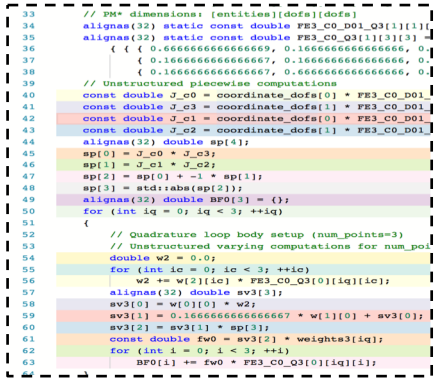


2. High-level user code (Python)

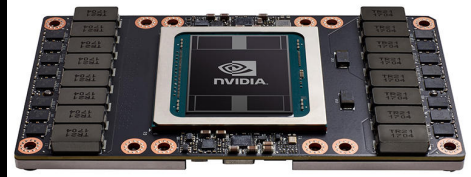
```python
elem = FiniteElement("Lagrange",tetrahedron,1)
coords = VectorElement("Lagrange",tetrahedron,1)
mesh = Mesh(coords)

V = FunctionSpace(mesh, elem)
u = TrialFunction(V)
v = TestFunction(V)
f = Coefficient(V)
kappa = Constant(mesh)

a = kappa * inner(grad(u), grad(v)) * dx
L = inner(f, v) * dx
```



```
// PM* dimensions: [entities][dofs][dofs]
alignas(32) static const double FE3_C0_D01_Q3[1][1][1] =
alignas(32) static const double FE3_C0_Q3[1][3][3] =
    { { { 0.6666666666666669, 0.1666666666666666, 0.
        { 0.1666666666666667, 0.1666666666666666, 0.
        { 0.1666666666666667, 0.6666666666666666, 0.
// Unstructured piecewise computations
const double J_c0 = coordinate_dofs[0] * FE3_C0_D01_
const double J_c3 = coordinate_dofs[1] * FE3_C0_D01_
const double J_c1 = coordinate_dofs[0] * FE3_C0_D01_
const double J_c2 = coordinate_dofs[1] * FE3_C0_D01_
alignas(32) double sp[4];
sp[0] = J_c0 * J_c3;
sp[1] = J_c1 * J_c2;
sp[2] = sp[0] + -1 * sp[1];
sp[3] = std::abs(sp[2]);
alignas(32) double BF0[3] = {};
for (int iq = 0; iq < 3; ++iq)
{
    // Quadrature loop body setup (num_points=3)
    // Unstructured varying computations for num_poi
    double w2 = 0.0;
    for (int ic = 0; ic < 3; ++ic)
        w2 += w[2][ic] * FE3_C0_Q3[0][iq][ic];
    alignas(32) double sv3[3];
    sv3[0] = w[0][0] * w2;
    sv3[1] = 0.1666666666666667 * w[1][0] + sv3[0];
    sv3[2] = sv3[1] * sp[3];
    const double fw0 = sv3[2] * weights3[iq];
    for (int i = 0; i < 3; ++i)
        BF0[i] += fw0 * FE3_C0_Q3[0][iq][i];
```



Automated GPU acceleration of finite element assembly

Performance (in Mdof/s) of matrix assembly for Poisson's equation with linear (P1) elements on dual-socket Intel Xeon Gold 6130 and AMD Epyc "Naples" 7601 CPUs, and an NVIDIA V100 GPU.

P1 elements	Xeon Gold 6130 (CPU)		Epyc "Naples" 7601 (CPU)		NVIDIA V100 (GPU)			
Mesh	FEniCS	Optimised	FEniCS	Optimised	Partial offload	Full offload	Lookup table	Rowwise
Uniform mesh 3	7.06	58.83	10.03	76.96	0.64	188.97	229.57	279.37
Cardiac mesh 1	6.60	64.38	10.46	71.48	0.39	229.78	220.57	320.67
Cardiac mesh 2	6.41	56.82	10.27	64.11	0.37	97.73	180.11	309.89
Cardiac mesh 3	6.38	59.05	10.67	69.61	0.35	85.34	165.31	292.08
Cardiac mesh 4	6.18	58.83	10.36	70.23	0.38	104.04	178.00	286.03

Best CPU performance

Best GPU performance

Parallel Computing
Volume 118, November 2023, 103051

Targeting performance and user-friendliness: GPU-accelerated finite element computation with automated code generation in FEniCS

James D. Trotter [a], Johannes Langguth [a b], Xing Cai [a c]

Automated GPU acceleration of entire FEM computation

		NVIDIA A100 (GPU)		1 Milan CPU node	
Mesh	Iterations	Time [s]	Mdof/s/it	Time [s]	Mdof/s/it
Uniform mesh 1	5	0.05	68.9	0.45	7.6
Uniform mesh 2	5	0.11	75.2	1.23	6.5
Uniform mesh 3	5	0.19	83.4	1.53	10.1
Uniform mesh 4	5	0.32	84.1	2.64	10.0
Uniform mesh 5	5	0.50	84.0	4.18	10.1
Uniform mesh 6	5	0.75	83.4	5.21	12.0
Uniform mesh 7	5	1.07	83.3	7.15	12.4
Uniform mesh 8	5	1.64	74.2	9.56	12.7
Cardiac mesh 1	6	0.26	88.1	2.04	11.1
Cardiac mesh 2	7	0.49	84.5	3.80	10.8
Cardiac mesh 3	7	0.57	82.5	4.10	11.4
Cardiac mesh 4	8	0.87	83.3	6.31	11.5

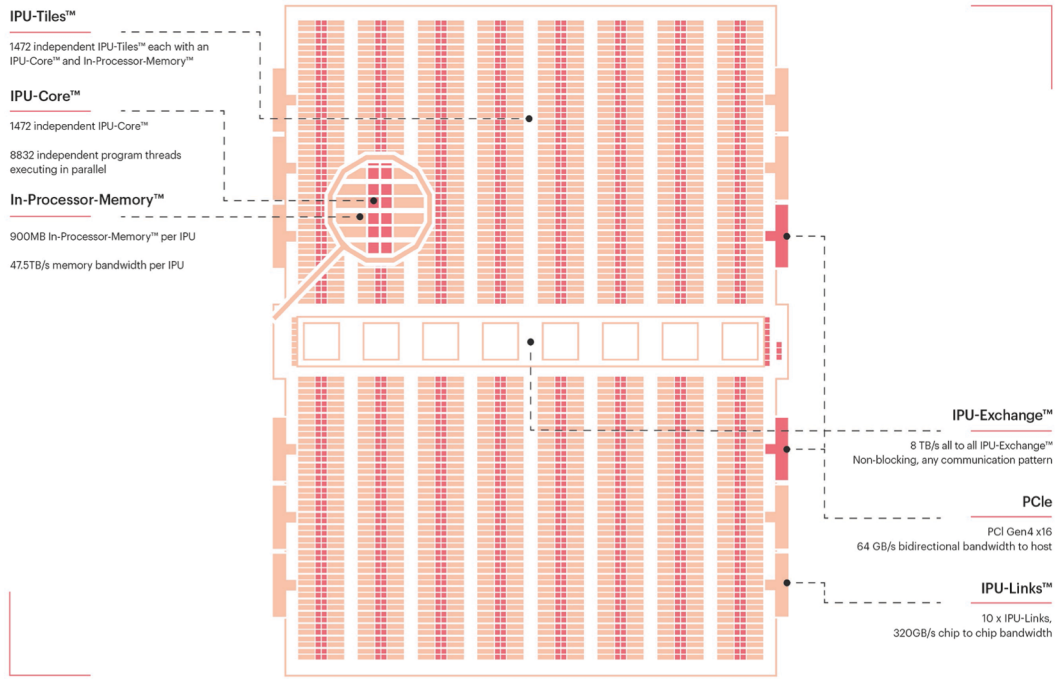
Parallel Computing
Volume 118, November 2023, 103051

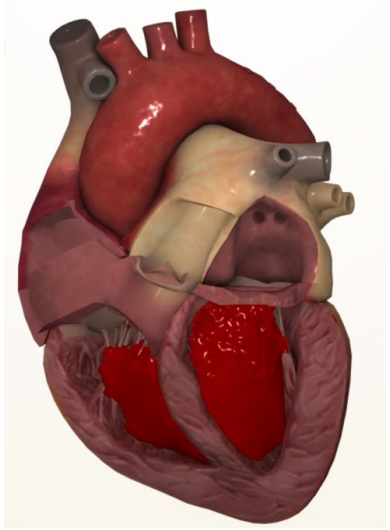
Targeting performance and user-friendliness: GPU-accelerated finite element computation with automated code generation in FEniCS

James D. Trotter [a], Johannes Langguth [a b], Xing Cai [a c]

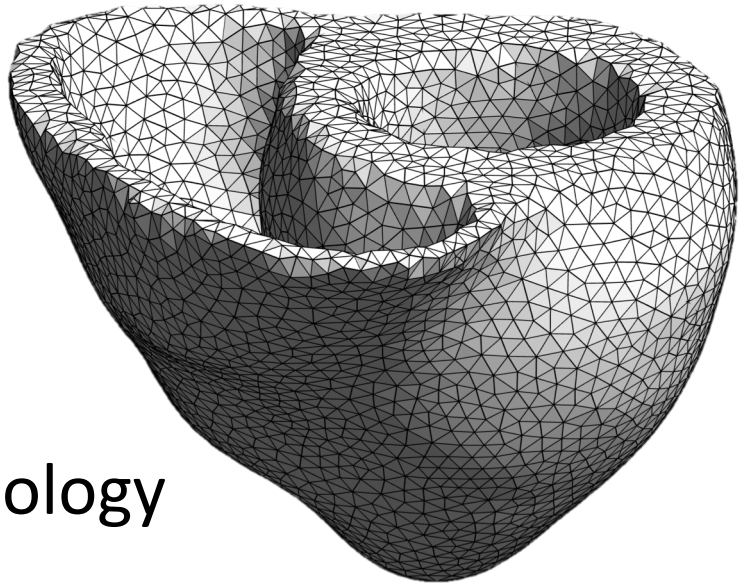
Re-purposing Graphcore intelligence processing units (IPUs)

- Massively parallel architecture
  1472 tiny cores per chip
  Each core has its private SRAM
  No chip-level shared memory
- Special programming style
  The computation needs to be formulated as a "dataflow" graph
    Nodes: small computational tasks for the cores
    Edges: flow of data between the nodes
  Inter-core communication is "implied"
- IPUs: originally designed for ML
  Simula ported several scientific computing applications to IPUs:
  heart simulation, graph analytics, sequence alignment, etc.





A simple model of cardiac electrophysiology

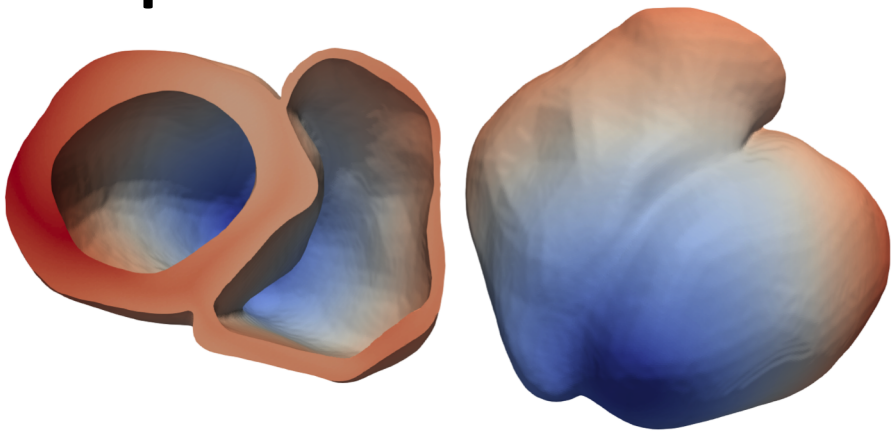


- The monodomain model of cardiac electrophysiology

$$\frac{\partial V_m}{\partial t} = \frac{-I_{\text{ion}}}{C_m} + \nabla \cdot (\mathbf{D} \nabla V_m)$$

- Operator splitting results in a “PDE” part and an “ODE part
  - PDE part: a 3D diffusion equation
  - ODE part: a system of nonlinear ODEs at every mesh entity
- Also subject of the JHPCN project between Simula & U. Tokyo

Porting a simple cardiac simulator to Graph IPUs



In comparison with using Nvidia's A100 GPUs

- The PDE part of the cardiac simulator runs faster on IPUs
- The ODE part is slower on IPUs

Performance comparison between GC200 IPUs and A100 GPUs, related to a monodomain simulation using the `heart04` mesh with 25,000 ODE steps and 100,000 PDE steps.

IPUs	Total time	PDE part	ODE part	GPUs	Total time	PDE part	ODE part
1	76.57 s	19.75 s	56.82 s	1	37.40 s	27.34 s	10.05 s
2	38.99 s	10.55 s	28.44 s	2	18.77 s	13.26 s	5.51 s
4	20.06 s	5.73 s	14.33 s	4	9.75 s	6.89 s	2.86 s
8	10.71 s	3.49 s	7.22 s	8	9.54 s	8.81 s	
16	6.02 s	2.26 s	3.76 s				

Front. Phys., 30 March 2023
Sec. Statistical and Computational Physics
Volume 11 - 2023 |
https://doi.org/10.3389/fphy.2023.979699

This article is part of the Research Topic
Heterogeneous Computing in Physics-Based Models
View all 5 Articles ›

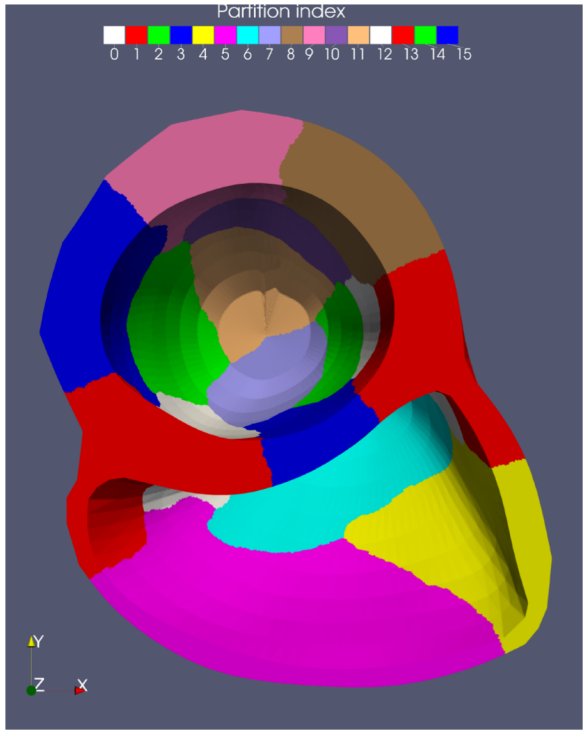
Enabling unstructured-mesh computation on massively tiled AI processors: An example of accelerating *in silico* cardiac simulation

Luk Burchard[1]* Kristian Gregorius Hustad[1] Johannes Langguth[1,2]
Xing Cai[1,3]

Physics-guided mesh partitioning

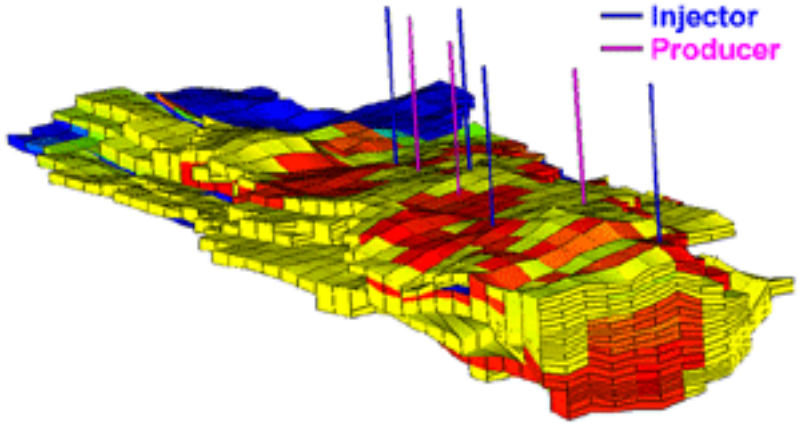
Mesh partitioning is the first step of parallel computing

- Partitioning an unstructured computational mesh is non-trivial
- Mesh partitioning affects the parallelization overhead, may also impact the numerical performance
- When mesh entities have heterogeneous connectivity strength, the partitioning problem needs special care



Example: Parallel solution of the “black-oil” reservoir model

$$
\begin{cases}
\dfrac{\partial}{\partial t}(\phi s_o \rho_o^o) = \nabla \cdot \left(\dfrac{KK_{ro}}{\mu_o} \rho_o^o \nabla \Phi_o \right) + q_o, \\
\dfrac{\partial}{\partial t}(\phi s_w \rho_w) = \nabla \cdot \left(\dfrac{KK_{rw}}{\mu_w} \rho_w \nabla \Phi_w \right) + q_w, \\
\dfrac{\partial(\phi \rho_o^g s_o + \phi \rho_g s_g)}{\partial t} = \nabla \cdot \left(\dfrac{KK_{ro}}{\mu_o} \rho_o^g \nabla \Phi_o \right) + \nabla \cdot \left(\dfrac{KK_{rg}}{\mu_g} \rho_g \nabla \Phi_g \right) + q_o^g + q_g, \\
\Phi_\alpha = p_\alpha + \rho_\alpha \wp z, \\
s_o + s_w + s_g = 1, \\
p_w = p_o - p_{cow}, \\
p_g = p_o + p_{cog},
\end{cases}
$$



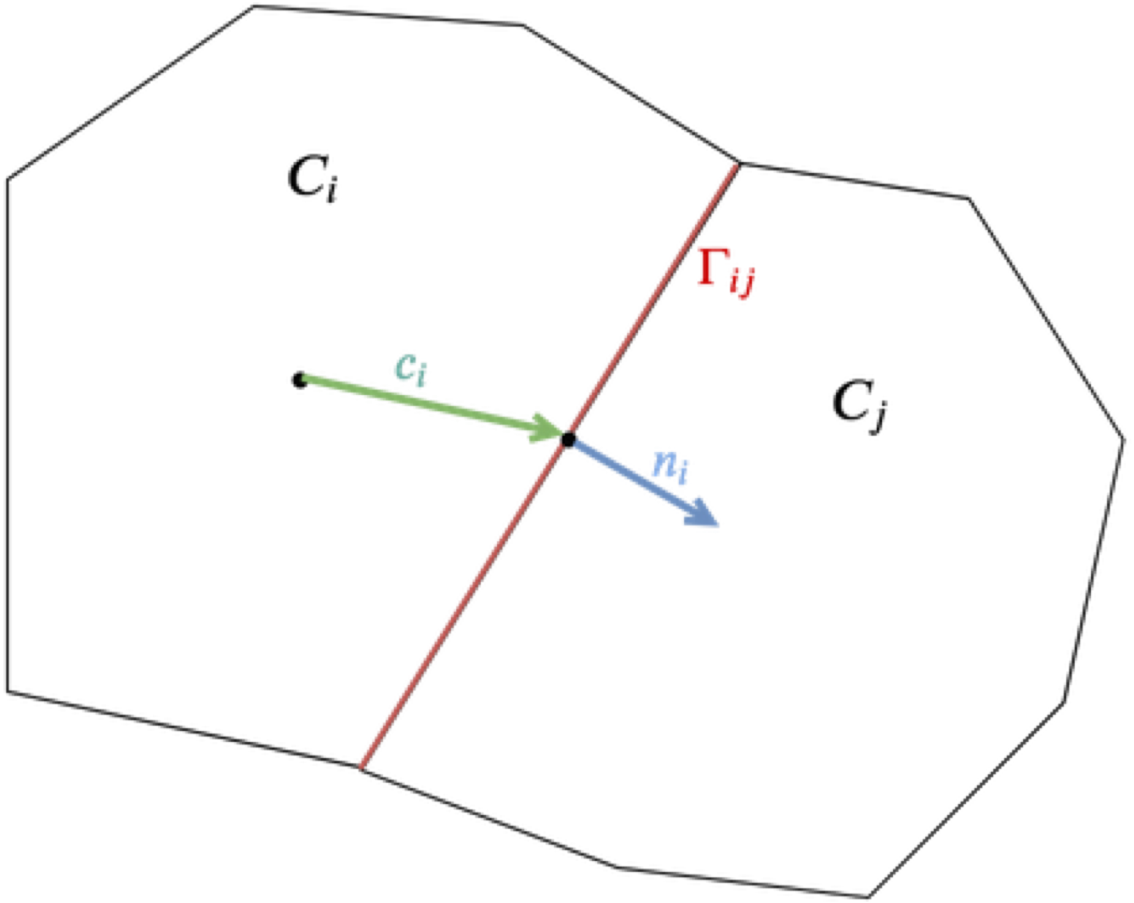
- **Large-scale reservoir simulations require parallelization**
- **Parallel preconditioners are essential for the iterative linear solvers, but their effectiveness is sensitive to mesh partitioning**
- **Mesh partitioning must therefore balance between parallel efficiency and numerical efficiency**

Transmissibility as a measure of numerical connectivity

- Cell-centered finite volume discretization
- Transmissibility across the boundary of two computational cells is a good measure of numerical connectivity

$$T_{ij} = m_{ij}|\Gamma_{ij}|\left(\frac{\|\vec{c}_i\|^2}{\vec{n}_i K_i \vec{c}_i} + \frac{\|\vec{c}_j\|^2}{\vec{n}_j K_j \vec{c}_j}\right)^{-1}$$

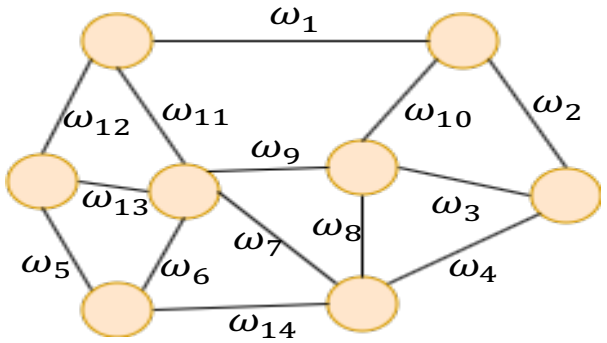
- Tightly connected cells should ideally not be divided between two subdomains

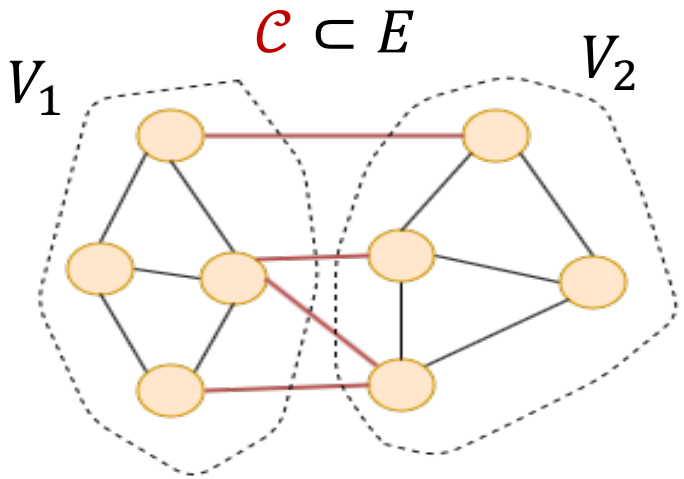


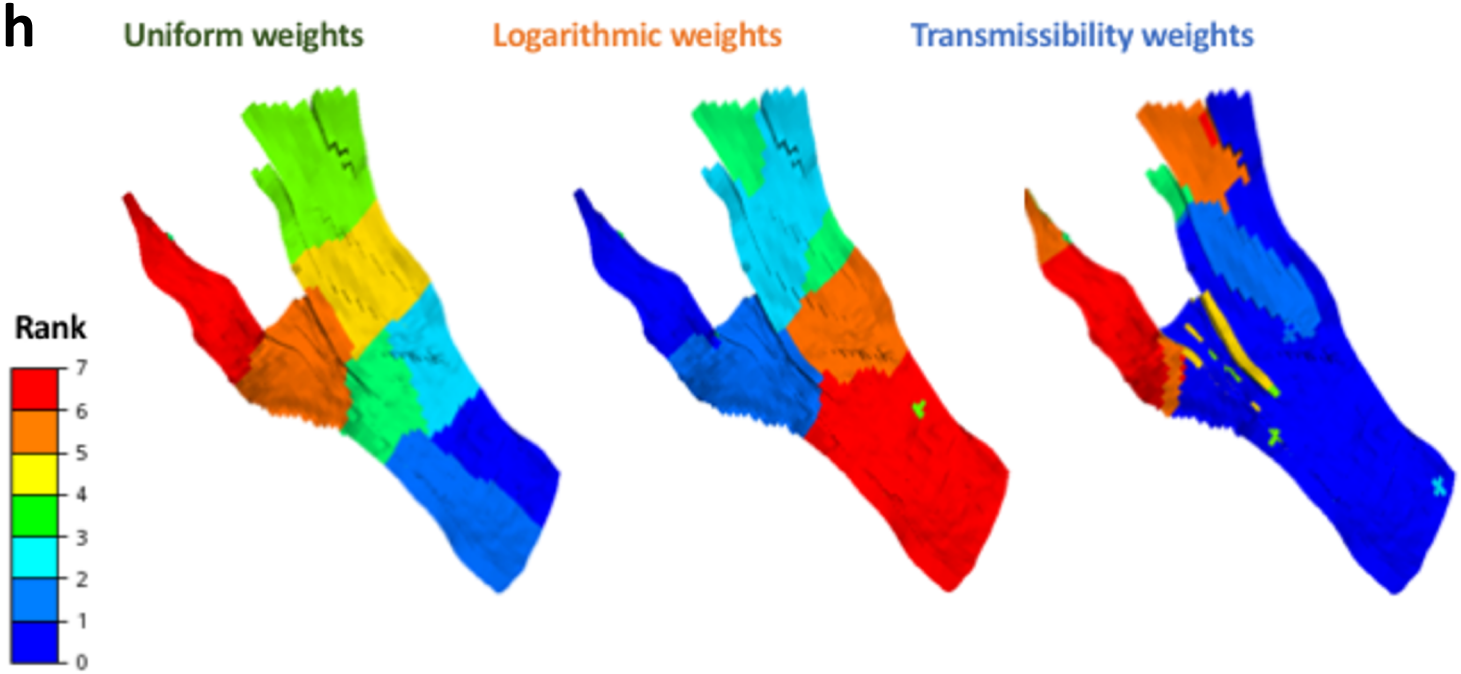
Three graph partitioners used for mesh partitioning

- The computational mesh is first translated to a graph, each cell becomes a vertex
- If two cells share a face, the two corresponding vertices are connected by an edge in the graph

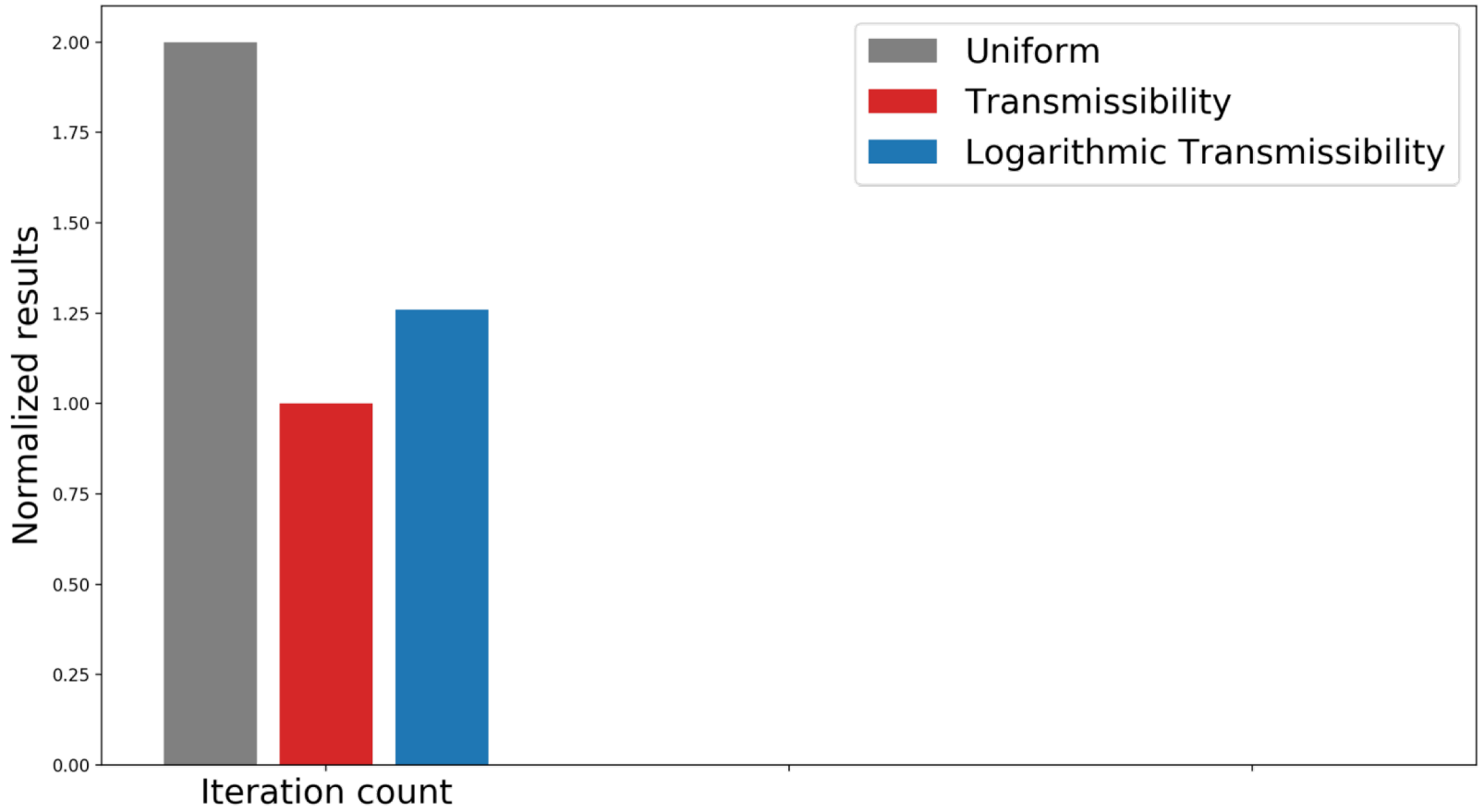
1. Each edge has uniform weight
2. Each edge is weighted by the transmissibility itself
3. Each edge is weighted by the logarithmic of transmissibility



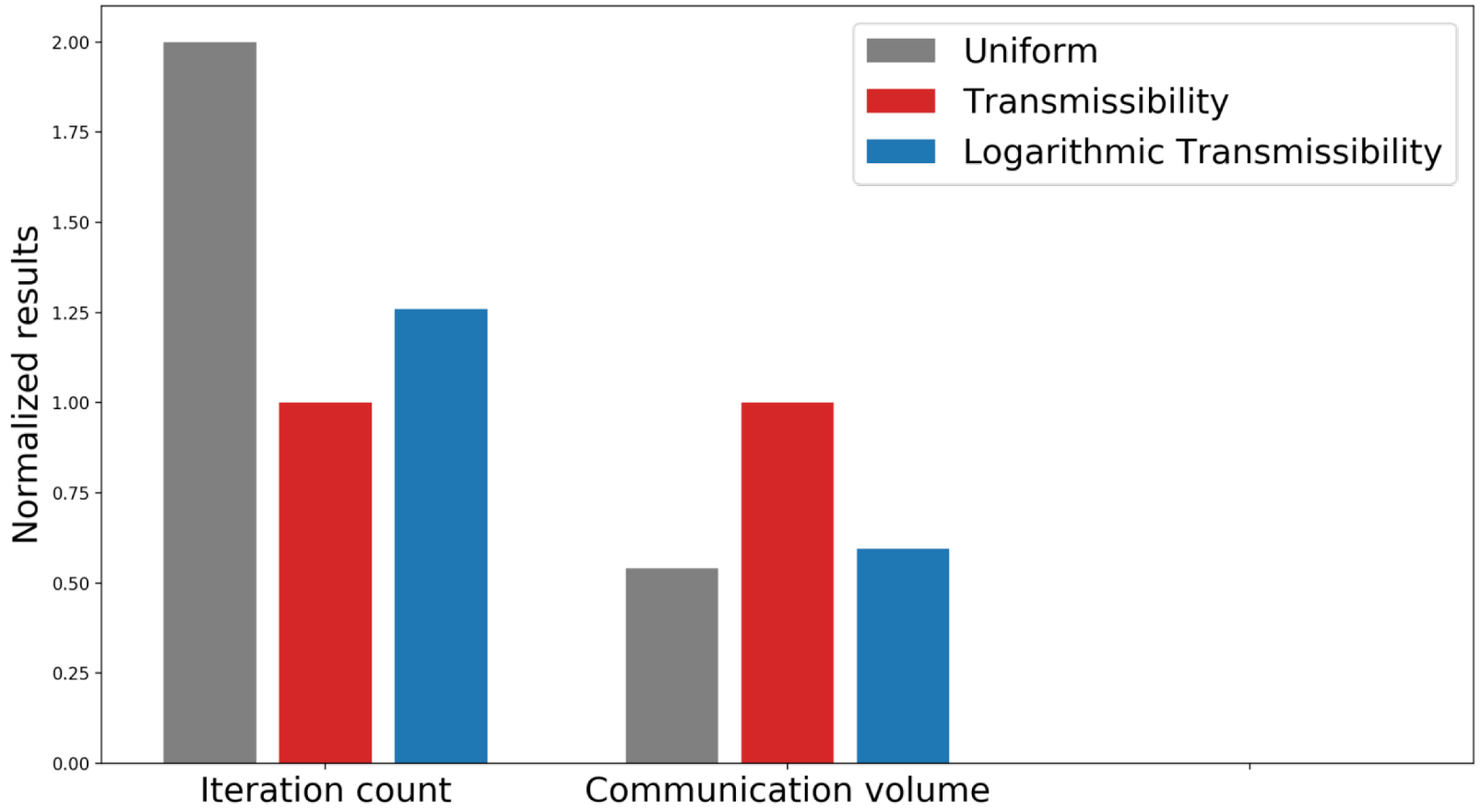




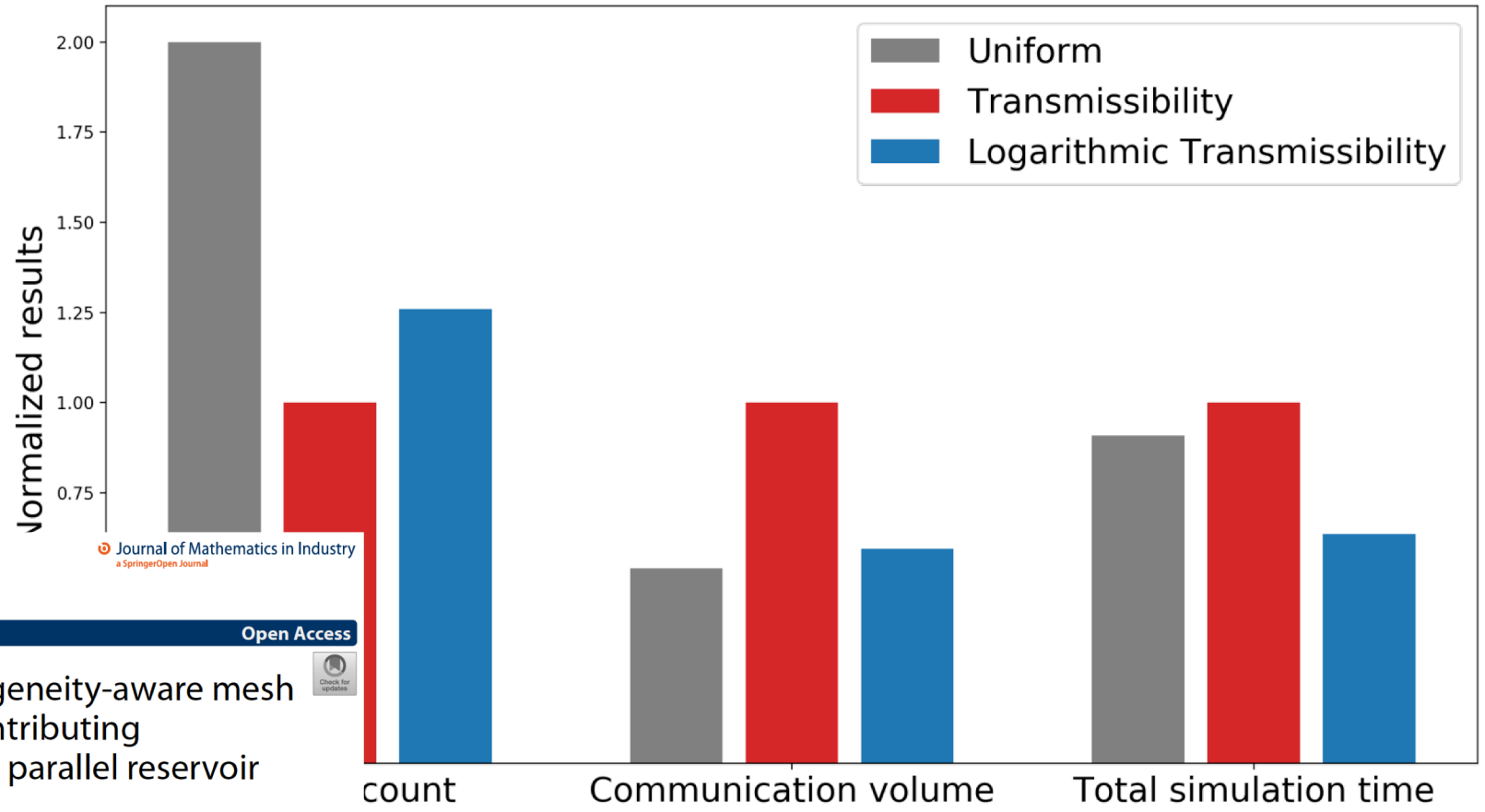
Comparing the edge-weight schemes using 64 MPI processes



Comparing the edge-weight schemes using 64 MPI processes



Comparing the edge-weight schemes using 64 MPI processes



Thune et al. *Journal of Mathematics in Industry* (2021) 11:12
https://doi.org/10.1186/s13362-021-00108-5

Journal of Mathematics in Industry
a SpringerOpen Journal

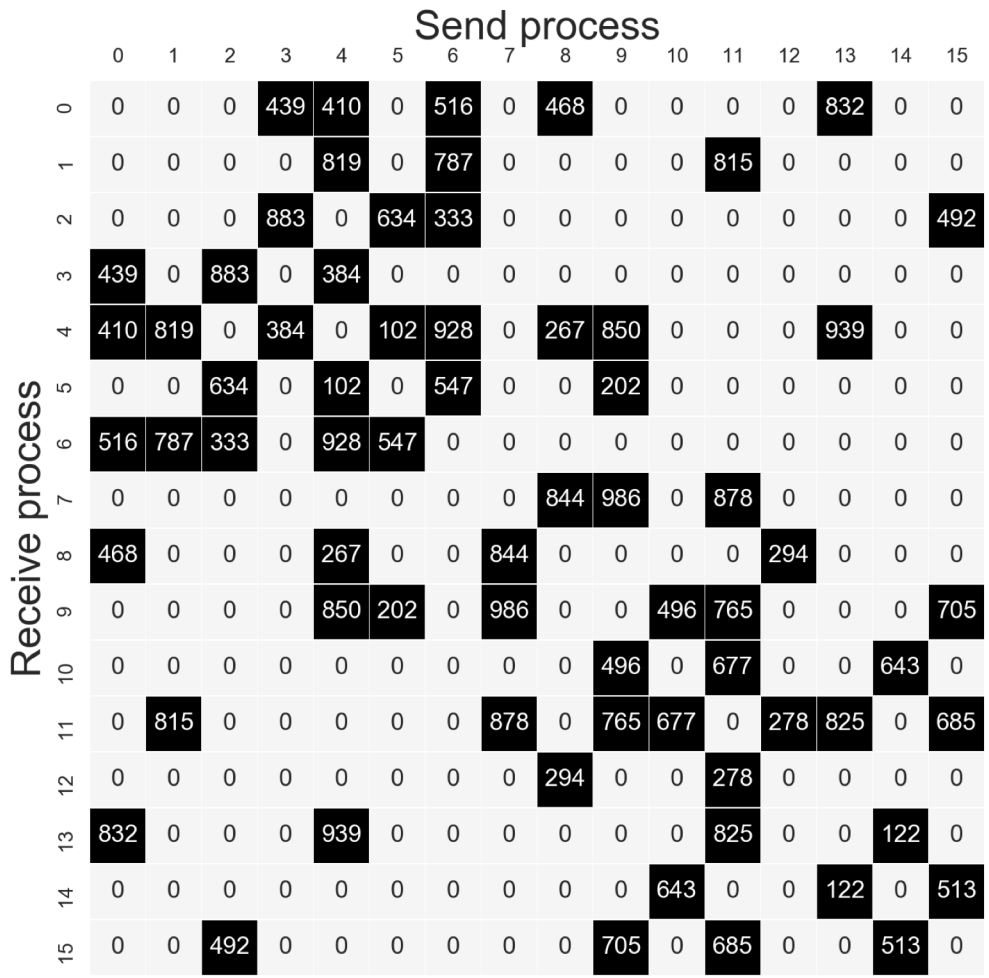
RESEARCH Open Access

On the impact of heterogeneity-aware mesh partitioning and non-contributing computation removal on parallel reservoir simulations

Andreas Thune[1,2*], Xing Cai[1,2] and Alf Birger Rustad[3]

Detailed modeling of communication overhead

- The inter-connect on a parallel system is often heterogeneous
- The actual process-to-process communication is also heterogeneous
- Detailed understanding of the communication overhead is important
- State-of-the-art models have weaknesses
- We have developed new models



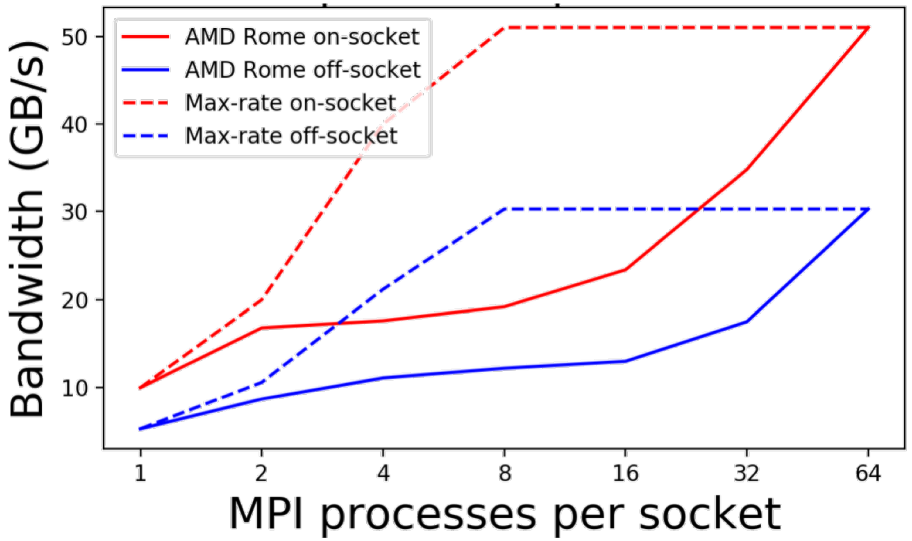
The state-of-the-art models have several weaknesses

Postal model:

$$T(s) = \tau + \frac{s}{BW}$$

The max-rate model [Gropp et al. 2016]:

$$T(s,N) = \tau + \frac{N \cdot s}{\min(N \cdot BW_{SP}, BW_{max})}$$



Our new model is based on a “staircase” principle

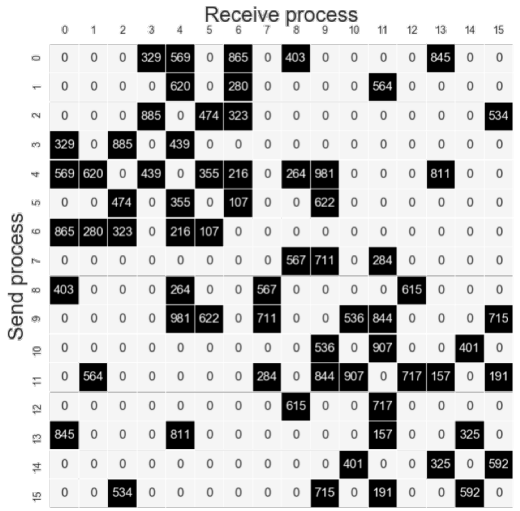
Staircase model

$$t_0^{recv} = \frac{N \cdot s_0}{BW_{MP}(N)},$$

$$t_i^{recv} = t_{i-1}^{recv} + \frac{(N-i)\cdot(s_i - s_{i-1})}{BW_{MP}(N-i)},$$

$$T_i = \tau + \max(t_i^{recv}, t_i^{send})$$

N	BW_{MP}
1	$BW_{MP}(1)$
2	$BW_{MP}(2)$
3	$BW_{MP}(3)$
4	$BW_{MP}(4)$
5	$BW_{MP}(5)$



Our new model is based on a “staircase” principle

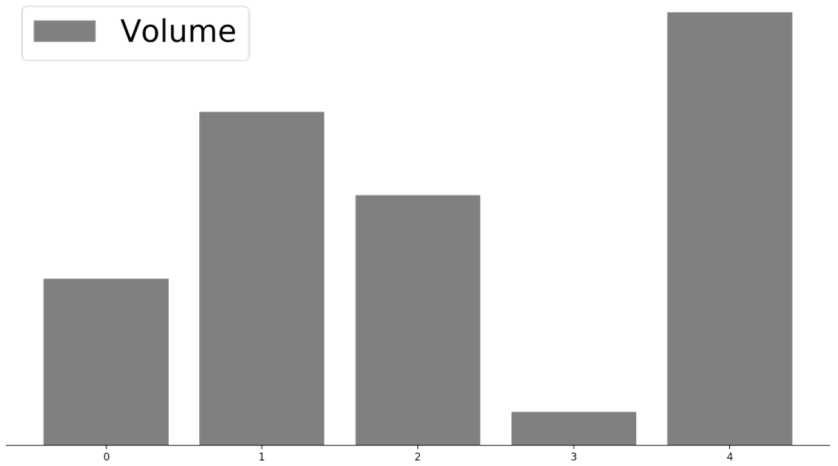
Staircase model

$$t_0^{recv} = \frac{N \cdot s_0}{BW_{MP}(N)},$$

$$t_i^{recv} = t_{i-1}^{recv} + \frac{(N-i)\cdot(s_i - s_{i-1})}{BW_{MP}(N-i)},$$

$$T_i = \tau + \max(t_i^{recv}, t_i^{send})$$

N	BW_{MP}
1	$BW_{MP}(1)$
2	$BW_{MP}(2)$
3	$BW_{MP}(3)$
4	$BW_{MP}(4)$
5	$BW_{MP}(5)$



Our new model is based on a “staircase” principle

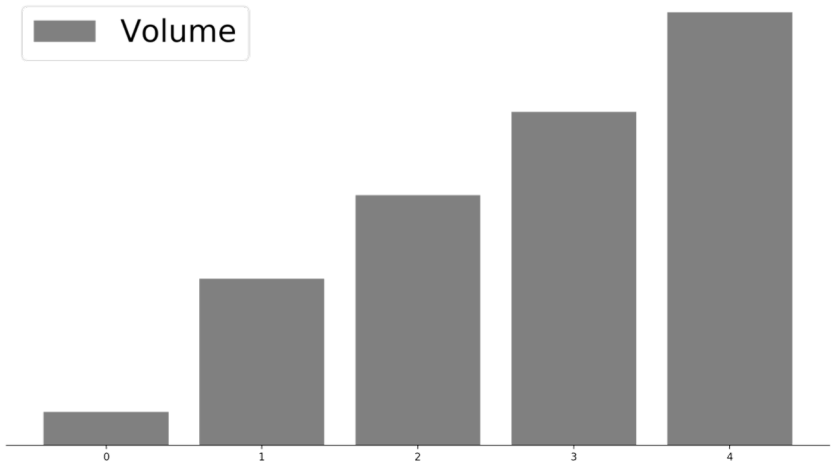
Staircase model

$$t_0^{recv} = \frac{N \cdot s_0}{BW_{MP}(N)},$$

$$t_i^{recv} = t_{i-1}^{recv} + \frac{(N-i)\cdot(s_i - s_{i-1})}{BW_{MP}(N-i)},$$

$$T_i = \tau + \max(t_i^{recv}, t_i^{send})$$

N	BW_{MP}
1	$BW_{MP}(1)$
2	$BW_{MP}(2)$
3	$BW_{MP}(3)$
4	$BW_{MP}(4)$
5	$BW_{MP}(5)$



Our new model is based on a “staircase” principle

Staircase model

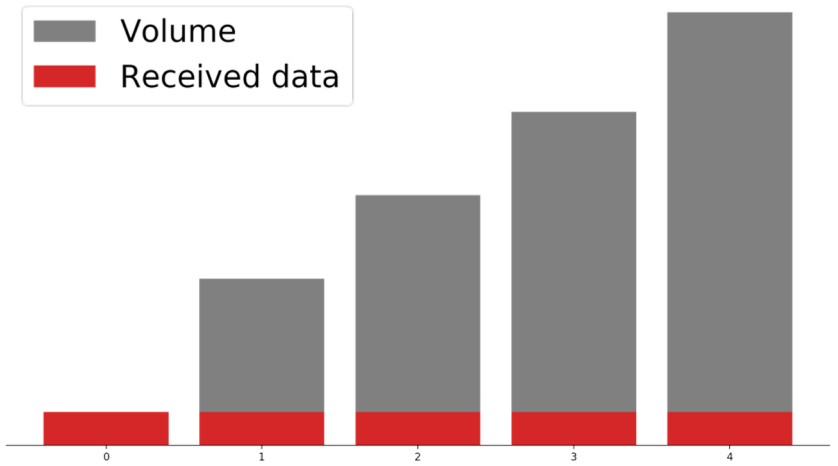
$$t_0^{recv} = \frac{N \cdot s_0}{BW_{MP}(N)},$$

$$t_i^{recv} = t_{i-1}^{recv} + \frac{(N-i)\cdot(s_i - s_{i-1})}{BW_{MP}(N-i)},$$

$$T_i = \tau + \max(t_i^{recv}, t_i^{send})$$

N	BW_{MP}
1	$BW_{MP}(1)$
2	$BW_{MP}(2)$
3	$BW_{MP}(3)$
4	$BW_{MP}(4)$
5	$\mathbf{BW_{MP}(5)}$

$$t_0^{recv} = \frac{5 \cdot s_0}{BW_{MP}(5)},$$



Our new model is based on a “staircase” principle

Staircase model

$$t_0^{recv} = \frac{N \cdot s_0}{BW_{MP}(N)},$$

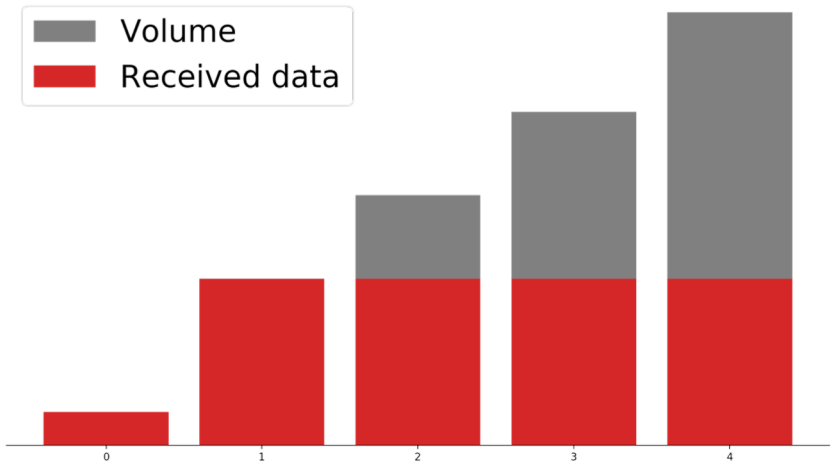
$$t_i^{recv} = t_{i-1}^{recv} + \frac{(N-i)\cdot(s_i - s_{i-1})}{BW_{MP}(N-i)},$$

$$T_i = \tau + \max(t_i^{recv}, t_i^{send})$$

N	BW_{MP}
1	$BW_{MP}(1)$
2	$BW_{MP}(2)$
3	$BW_{MP}(3)$
4	$\mathbf{BW_{MP}(4)}$
5	$BW_{MP}(5)$

$$t_0^{recv} = \frac{5 \cdot s_0}{BW_{MP}(5)},$$

$$t_1^{recv} = t_0^{recv} + \frac{4\cdot(s_1 - s_0)}{BW_{MP}(4)},$$



Our new model is based on a “staircase” principle

Staircase model

$$t_0^{recv} = \frac{N \cdot s_0}{BW_{MP}(N)},$$

$$t_i^{recv} = t_{i-1}^{recv} + \frac{(N-i)\cdot(s_i - s_{i-1})}{BW_{MP}(N-i)},$$

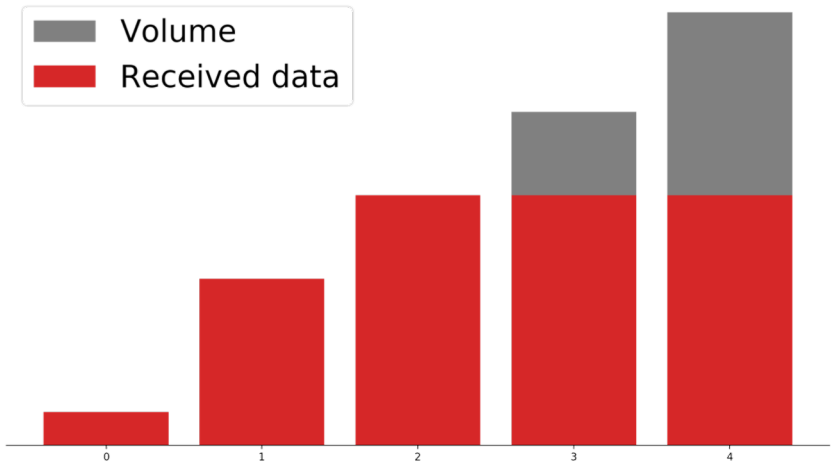
$$T_i = \tau + \max(t_i^{recv}, t_i^{send})$$

N	BW_{MP}
1	$BW_{MP}(1)$
2	$BW_{MP}(2)$
3	$\mathbf{BW_{MP}(3)}$
4	$BW_{MP}(4)$
5	$BW_{MP}(5)$

$$t_0^{recv} = \frac{5 \cdot s_0}{BW_{MP}(5)},$$

$$t_1^{recv} = t_0^{recv} + \frac{4\cdot(s_1 - s_0)}{BW_{MP}(4)},$$

$$t_2^{recv} = t_1^{recv} + \frac{3\cdot(s_2 - s_1)}{BW_{MP}(3)},$$



Our new model is based on a “staircase” principle

Staircase model

$$t_0^{recv} = \frac{N \cdot s_0}{BW_{MP}(N)},$$

$$t_i^{recv} = t_{i-1}^{recv} + \frac{(N-i)\cdot(s_i - s_{i-1})}{BW_{MP}(N-i)},$$

$$T_i = \tau + \max(t_i^{recv}, t_i^{send})$$

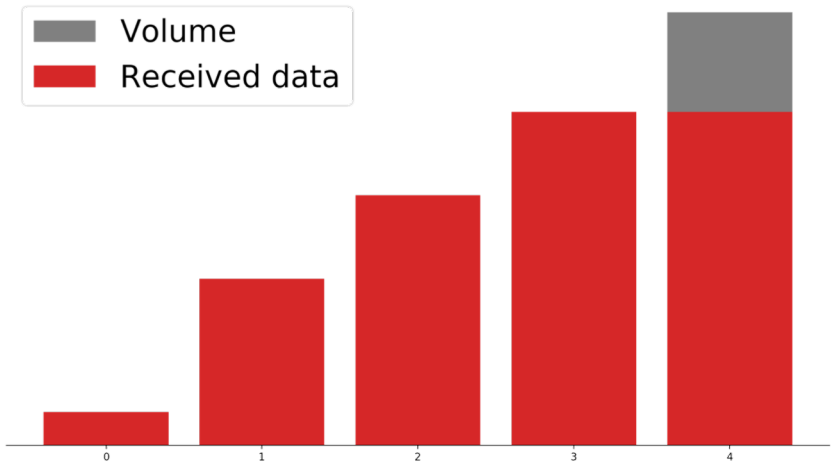
N	BW_{MP}
1	$BW_{MP}(1)$
2	$\mathbf{BW_{MP}(2)}$
3	$BW_{MP}(3)$
4	$BW_{MP}(4)$
5	$BW_{MP}(5)$

$$t_0^{recv} = \frac{5 \cdot s_0}{BW_{MP}(5)},$$

$$t_1^{recv} = t_0^{recv} + \frac{4\cdot(s_1 - s_0)}{BW_{MP}(4)},$$

$$t_2^{recv} = t_1^{recv} + \frac{3\cdot(s_2 - s_1)}{BW_{MP}(3)},$$

$$t_3^{recv} = t_2^{recv} + \frac{2\cdot(s_3 - s_2)}{BW_{MP}(2)},$$



Our new model is based on a “staircase” principle

Staircase model

$$t_0^{recv} = \frac{N \cdot s_0}{BW_{MP}(N)},$$

$$t_i^{recv} = t_{i-1}^{recv} + \frac{(N-i)\cdot(s_i - s_{i-1})}{BW_{MP}(N-i)},$$

$$T_i = \tau + \max(t_i^{recv}, t_i^{send})$$

N	BW_{MP}
1	**$BW_{MP}(1)$**
2	$BW_{MP}(2)$
3	$BW_{MP}(3)$
4	$BW_{MP}(4)$
5	$BW_{MP}(5)$

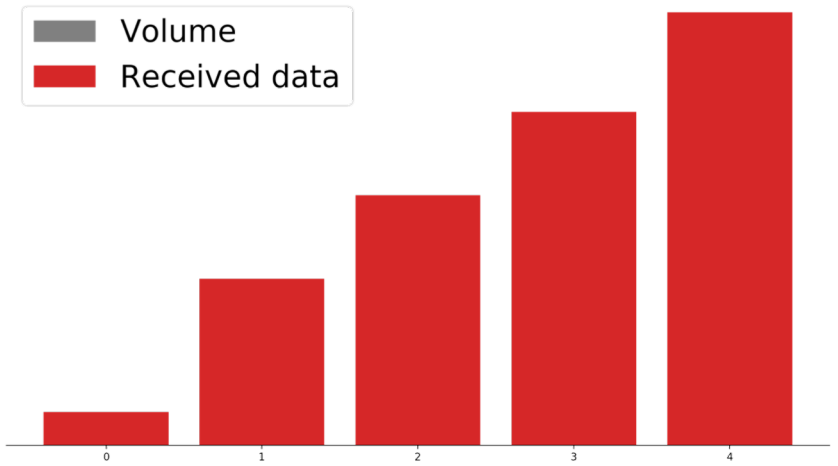
$$t_0^{recv} = \frac{5 \cdot s_0}{BW_{MP}(5)},$$

$$t_1^{recv} = t_0^{recv} + \frac{4\cdot(s_1 - s_0)}{BW_{MP}(4)},$$

$$t_2^{recv} = t_1^{recv} + \frac{3\cdot(s_2 - s_1)}{BW_{MP}(3)},$$

$$t_3^{recv} = t_2^{recv} + \frac{2\cdot(s_3 - s_2)}{BW_{MP}(2)},$$

$$t_4^{recv} = t_3^{recv} + \frac{(s_4 - s_3)}{BW_{MP}(1)},$$



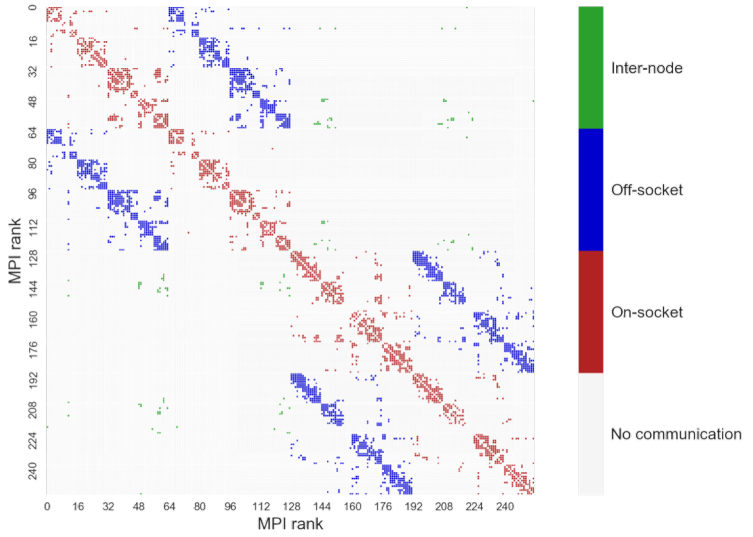
The Staircase model works in more general cases when messages are mixed intra-socket, inter-socket, and inter-node

Mixed intra-node bandwidth estimate:

$$BW_{MP}^{mix}(N,\theta_i) = \frac{\theta_i}{N} BW_{MP}^{off}(N) + \frac{1-\theta_i}{N} BW_{MP}^{on}(N)$$

General case estimate:

$$T_i^{total} = T_i^{intra-node} + T_i^{inter-node}$$



Journals & Magazines > IEEE Transactions on Parallel... > Volume: 34 Issue: 5

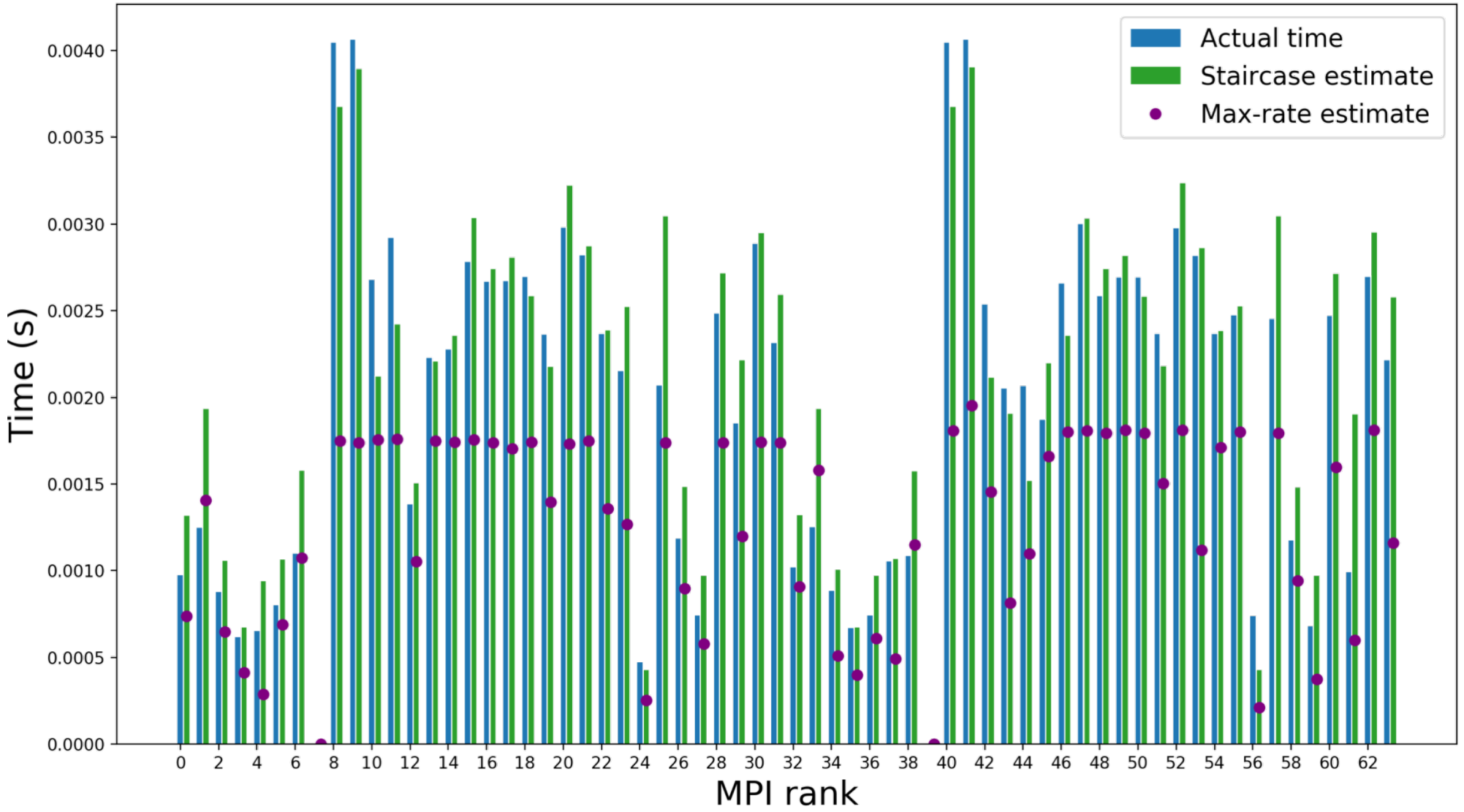
Detailed Modeling of Heterogeneous and Contention-Constrained Point-to-Point MPI Communication

Publisher: IEEE

Cite This PDF

Andreas Thune ; Sven-Arne Reinemo ; Tor Skeie ; Xing Cai All Authors

Example of detailed modeling of communication overhead



Concluding remarks

New research needed for mesh re-ordering & partitioning

Further developments of automated code generation for accelerated computing

Important international collaborations

- **JHPCN project with U. Tokyo**
- **SparCity project (EuroHPC)**

RESEARCH-ARTICLE OPEN ACCESS

Modelling Data Locality of Sparse Matrix-Vector Multiplication on the A64FX

Authors: Sergej Breiter, James D. Trotter, Karl Fürlinger Authors Info & Claims

SC-W '23: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis • November 2023 •
Pages 1334–1342 • https://doi.org/10.1145/3624062.3624198

東京大学
THE UNIVERSITY OF TOKYO

SPAR
CITY