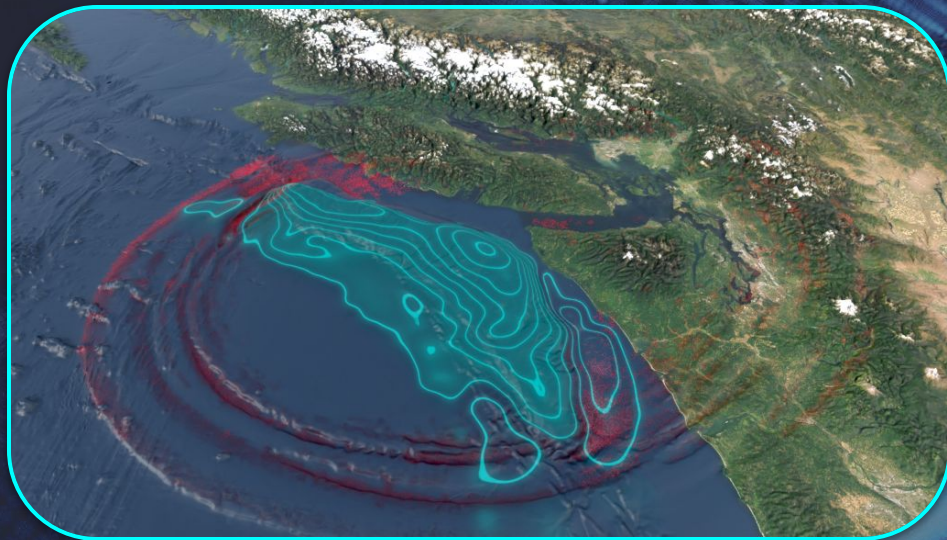




MFEM Community Workshop

A Guided Tour of MFEM GPU Kernel Optimization Techniques

John Camier - LLNL Collaborator,
Veselin Dobrev, Tzanio Kolev - LLNL
Stefan Henneking - Oden Institute, The University of Texas at Austin
Jiqun Tu - NVIDIA

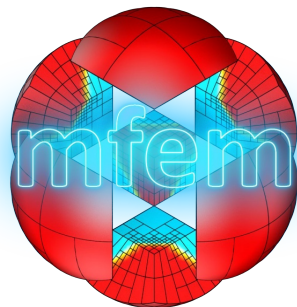


MFEM Workshop - GPU Kernel Optimizations Guided Tour

- Welcome students, new users & developers
- Exploring GPU kernel optimization strategies in MFEM

- Arbitrary order curvilinear **mesh** elements
- Arbitrary order **H1**, H(curl), H(div) and **L2** elements
- **Bilinear**/linear forms for: Galerkin, DG, etc.
- MPI-scalable assembly and linear solvers
- **GPU** acceleration on **AMD**, **NVIDIA** hardware
- Non-linear operators and non-linear solvers
- Explicit and implicit high-order time integration
- Integration with: hypre, SUNDIALS, SuperLU, PETSc, etc.

⇒ Real-time Bayesian inference at extreme scale:
A digital twin for tsunami early warning applied to the Cascadia subduction zone ^[1]



mfem.org v4.8 - Apr 2025



CEED
EXASCALE DISCRETIZATIONS

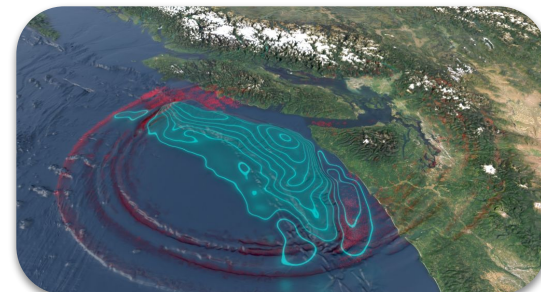
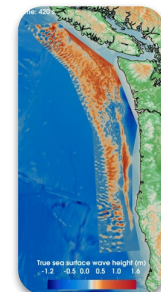
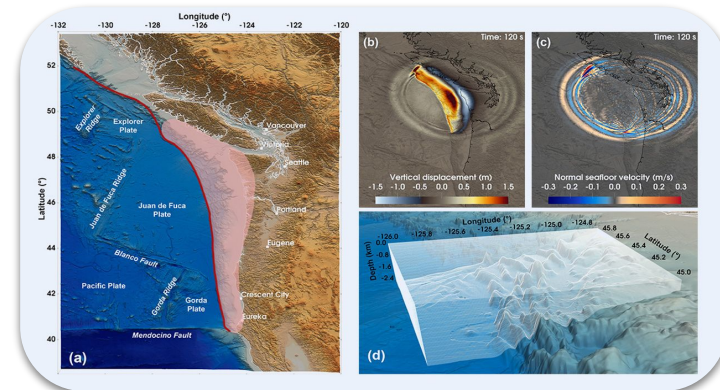
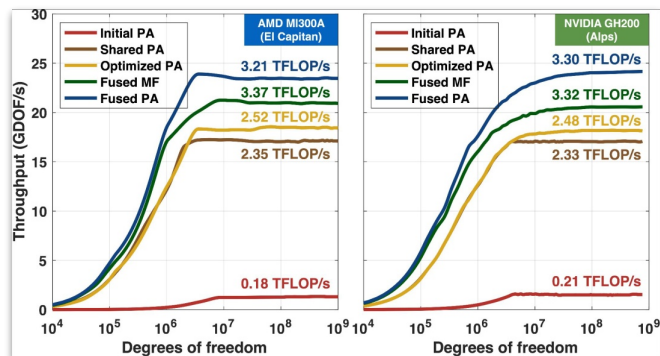


Application - A Digital Twin For Tsunami Early Warning

- Important and challenging problem
- Forecast wave heights or onshore inundation
- Produce better early warning systems for tsunamis

$$\left(A \begin{bmatrix} \vec{u} \\ p \end{bmatrix}, \begin{bmatrix} \vec{\tau} \\ v \end{bmatrix} \right) := \begin{bmatrix} 0 & (\nabla p, \vec{\tau}) \\ -(\vec{u}, \nabla v) & \langle Z^{-1} p, v \rangle_{\partial \Omega_a} \end{bmatrix}$$

- Problem size \Rightarrow memory optimizations
- Key kernels \Rightarrow performance optimizations



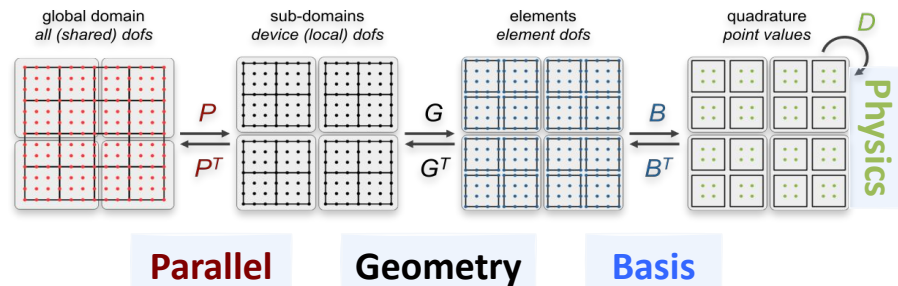
MFEM Operator Decomposition for GPU Kernels

- Partial Assembled HO Finite Element Operators

- $$A = \mathbf{P}^T \mathbf{G}^T \mathbf{B}^T \mathbf{D} \mathbf{B} \mathbf{G} \mathbf{P}$$

- Optimal memory, near-optimal FLOPs

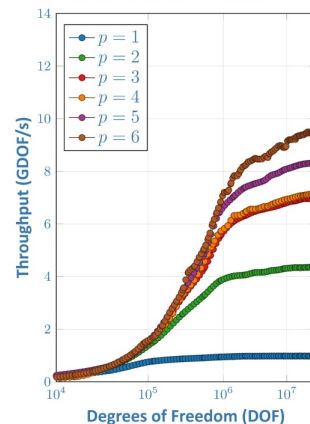
- Matrix free: no assembly of the full matrix A



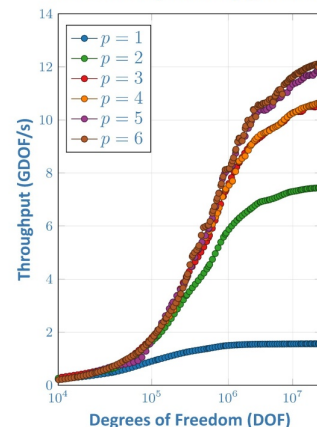
- GPU Kernel optimization: focus on $\mathbf{G}^T \mathbf{B}^T \mathbf{D} \mathbf{B} \mathbf{G}$



MFEM BP1 (standard) @ GH200



MFEM BP1 (atomics) @ GH200



MFEM GPU Kernel Overview

MFEM Features Examples Documentation Community Gallery Download

Integration

MFEM's spatial integrations are performed in the usual finite element manner by first splitting the spatial domain into a collection of non-overlapping "elements" which cover the domain. This is usually referred to as the "mesh". An integral can then be computed separately in each element and the results added together:

$$\int_{\Omega} f(x) d\Omega = \sum_i \int_{\Omega_i} f(x) d\Omega \quad (1)$$

Where Ω is the full domain and Ω_i is the domain of the i -th element. In MFEM this sum over elements is performed in classes such as the `BilinearForm` or `LinearForm` and their parallel counterparts.

MFEM Features Examples Documentation Community Gallery Download

Bilinear Form Integrators

Bilinear form integrators are at the heart of any finite element method; they are used to compute the integrals of products of basis functions over individual mesh elements (or sometimes over edges or faces). Typically each element is contained in the support of several basis functions of both the domain and range spaces, therefore bilinear integrators simultaneously compute the integrals of all combinations of the relevant basis functions from the domain and range spaces. This produces a two dimensional array of results that are arranged into a small dense matrix of integral values called a *local element (stiffness) matrix*.

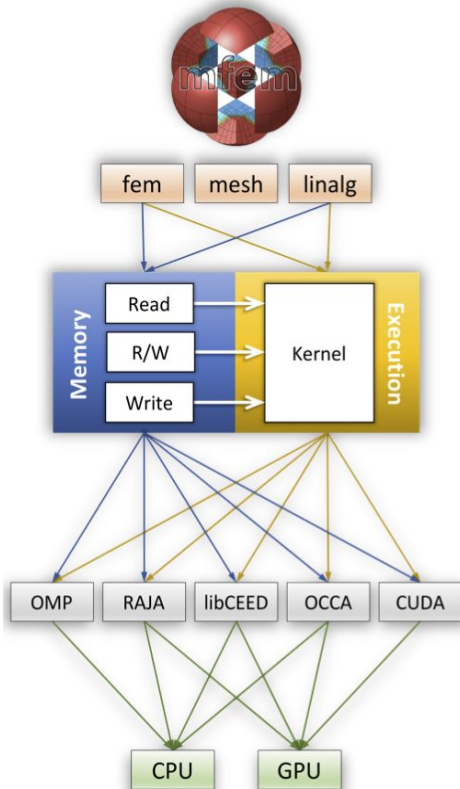
To put this another way, the `BilinearForm` class builds a global, sparse, finite element matrix, `glob_mat`, by performing the outer loop in the following pseudocode snippet whereas the `BilinearFormIntegrator` class performs the nested inner loops to compute the dense local element matrix, `loc_mat`.

```

for each elem in elements
  loc_mat = 0.0
  for each pt in quadrature_points
    for each u_j in elem
      for each v_i in elem
        loc_mat(i,j) += w(pt) * u_j(pt) * v_i(pt)
      end
    end
  end
  glob_mat += loc_mat
end
    
```

Mixed Operators

Class Name	Domain	Range	Coef.	Dimension	Operator
VectorDivergenceIntegrator	H^1_d, L^2_d	H^1, L^2	S	1D, 2D, 3D	$(\lambda \mathbf{v} \cdot \vec{\mathbf{u}}, \mathbf{v})$
GradientIntegrator	H^1	H^1_d, L^2_d	S	1D, 2D, 3D	$(\lambda \nabla \mathbf{u}, \vec{\mathbf{v}})$



MFEM Features Examples Documentation Community Gallery Download

GPU support in MFEM

MFEM relies mainly on two features for running algorithms on devices such as GPUs:

- The memory manager handles transparently the moving of data between the host (CPU) and the device (e.g. GPU),
- The `mfem::forall` function to abstract `for` loops to parallelize the execution on an arbitrary device.

```

Vector u;
Vector v;
// ...
const auto u_data = u.Read(); // Express the intent to read u
auto v_data = v.ReadWrite(); // Express the intent to read and write v

// Abstract the loop: for(int i=0; i<u.Size(); i++)
mfem::forall(u.Size(), [=] MFEM_HOST_DEVICE (int i)
{
    v_data[i] += u_data[i]; // This block of code is executed on the chosen device
});
    
```

Supported Integrators native MFEM

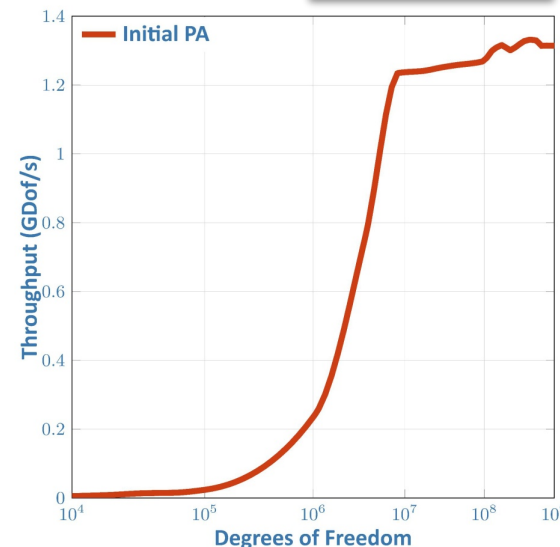
- Mass Integrator ☒
- Vector Mass Integrator ☒
- Vector FE Mass Integrator ☒
- Convection Integrator ☒
- Non-linear Convection Integrator ☒
- Diffusion Integrator ☒
- Vector Diffusion Integrator ☒
- DGTrace Integrator ☒
- Mixed Vector Gradient Integrator ☒
- Mixed Vector Curl Integrator ☒
- Mixed Vector Weak Curl Integrator ☒
- Gradient Integrator ☒
- Vector Divergence Integrator ☒
- Vector FE Divergence Integrator ☒
- Curl Curl Integrator ☒
- Div Div Integrator ☒

- **Memory:** input/outputs data management
- **Execution:** outer/inner `forall` loops
- **Kernel:** Integrator, $G^T B^T D B G$

Initial PA - Integrators Implementation

- Mixed integrator: B_{test} , B_{trial}
- Fixed order: $\{4_{\text{test}}, 5_{\text{trial}}\}$ dofs, 5 at quadrature points
- CPU development, GPU portable
- Extract: PA setup, Q function

AMD MI300A



Writing Custom Integrators

Element-wise integration arises in various places in the finite element method. A square and rectangular bilinear form operators, linear functionals, and the calcu

Type	Primary Function Needing Implementation
Square Operators	<code>BilinearFormIntegrator::AssembleElementMat</code>
Rectangular Operators	<code>BilinearFormIntegrator::AssembleElementMat</code>
Linear Functionals	<code>LinearFormIntegrator::AssembleRHSElementVect</code>

Name	C++ Expression	Formula
Jacobian Matrix	<code>const DenseMatrix &J = Trans.Jacobian()</code>	$J_{ij} = \frac{\partial y_i}{\partial x_j}$
Jacobian Determinant	<code>double detJ = Trans.Weight()</code>	$\det(J)$
Inverse Jacobian	<code>const DenseMatrix &InvJ = Trans.InverseJacobian()</code>	J^{-1}
Adjugate Jacobian	<code>const DenseMatrix &AdjJ = Trans.AdjugateJacobian()</code>	$\det(J) J^{-1}$

```
void GradientIntegrator::AssembleElementMatrix()
{
    const FiniteElement &trial_fe, const FiniteElement &test_fe,
    ElementTransformations &Ttrans, DensitiesMatrix &Gmat;

    Full matrix A

    dim = test_fe.GetDim();
    int trial_dof = &trial_fe->GetDofIndex(0);
    real c = 1;
    Vector d_col;

    dshape.SetSize(trial_dof, dim); gshape.SetSize(trial_dof, dim);
    Jadj.SetSize(dim, shape.SetSize(dim));
    elmat.SetSize(dim * dim, trial_dof);

    const IntegrationRule nInt = GetIntegrationRule(&trial_fe, test_fe, Ttrans);
    elmat = 0.0;
    elmat.col.SetSize(test_dof, trial_dof);

    for (int i = 0; i < nInt.GetNPoints(); i++)
    {
        const IntegrationPoint &p = nInt.GetIntPoint(i);
        Ttrans.SetPoint(i, p);

        CalcAdjugate(Ttrans, Jacobian(), Jadj);
        test_fe.CalcPsiShape(Ttrans, shape, &trial_fe.CalcQShape(p), dshape);
        MultiDShape Jadj, gshape;

        c = 0. weight;
        if (Q) { c = Q-Eval(i, Ttrans, ip); }
        shape = c;

        for (int d = 0; d < dim; ++d)
        {
            gshape.SetColumnReference(d, d_col);
            MultiVecShape d_col, elmat.col;

            for (int j = 0; j < trial_dof; ++j)
            {
                for (int ii = 0; ii < test_dof; ++ii)
                {
                    elmat(d = test_dof + ii, jj) = elmat.col(ii, jj);
                }
            }
        }
    }
}
```

[illegible]

GPU Kernel Optimizations - Profiling and Benchmarking

- Using Google benchmark: agile development, CPU & GPU timings
- tests/benchmarks examples

Benchmark	Time	CPU	Iterations	Dofs	MDof/s	p	version
WaveOp/1/4/160	333 ms	333 ms	2	16.4613M	49.4931/s	4	1
WaveOp/8/4/160	305 ms	305 ms	2	16.4613M	53.9399/s	4	8
WaveOp/3/4/160	187 ms	187 ms	4	16.4613M	88.1353/s	4	3
WaveOp/4/4/160	144 ms	144 ms	5	16.4613M	114.076/s	4	4
WaveOp/6/4/160	174 ms	174 ms	4	16.4613M	94.6562/s	4	6
WaveOp/9/4/160	473 ms	473 ms	2	16.4613M	34.799/s	4	9
WaveOp/13/4/160	242 ms	242 ms	3	16.4613M	67.9619/s	4	13

Apple M2 Pro



Benchmark	Time	CPU	Iterations	Dofs	MDof/s	p	version
WaveOp/1/4/160	17.4 ms	17.1 ms	41	16.4613M	960.653/s	4	1
WaveOp/8/4/160	1.83 ms	1.82 ms	385	16.4613M	9.06081k/s	4	8
WaveOp/3/4/160	0.897 ms	0.897 ms	773	16.4613M	18.3594k/s	4	3
WaveOp/4/4/160	0.703 ms	0.700 ms	983	16.4613M	23.5143k/s	4	4
WaveOp/6/4/160	1.20 ms	1.19 ms	586	16.4613M	13.8101k/s	4	6
WaveOp/9/4/160	1.07 ms	1.06 ms	656	16.4613M	15.531k/s	4	9
WaveOp/13/4/160	0.818 ms	0.810 ms	860	16.4613M	20.316k/s	4	13

AMD MI300A

- -Rpass-analysis=kernel-resource-usage [S,V,A]GPRs, ScratchSize, Occupancy, LDS Size
- --ptxas-options=-v

void mfem::HipKernel1D<mfem::RK4... void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPAGra... void mfem::HipKernel... void mf... void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPA... v void mfem::HipKerne...

5M elements, 1.3B dofs

- rocprofv3 & <https://ui.perfetto.dev>
- NVIDIA Nsight Systems/Compute



Shared Memory PA Kernel Optimizations

void mfem::HipKernel1D<mfem::RK4...

void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPAGra...

void mfem::HipKernel...

void mf...

void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPA...

void mfem::HipKerne...

AMD MI300A

B^T trial D^T B test

G^T trial G trial

B^T test D B trial

```

//Compile -c hip_d10r, int d10r, int d10r
void SmemPAGradientApply3D(const int NE,
                           const Array<real_t> &trial_bt,
                           const Array<real_t> &trial_gt,
                           const Array<real_t> &test_bt,
                           const Vector &id,
                           const Vector &dx, Vector &dy,
                           real_t a, real_t b)
{
    constexpr int MD1 = D10R > D10R ? D10R : D10R;
    const auto br_t = Reshape(trial_bt.Read(), D10R, Q1D);
    const auto gr_t = Reshape(trial_gt.Read(), D10R, Q1D);
    const auto bt = Reshape(test_bt.Read(), Q1D, Q1D);

    const auto D = Reshape(d.Read(), Q1D > Q1D ? Q1D : 3, 3, NE);
    const auto X = Reshape(x.Read(), D10R, D10R, NE, 3);
    auto Y = Reshape(y.Write(), D10R, D10R, D10R, NE);

    mfem::forall_3D(NE, Q1D, Q1D, Q1D, [=] MFEM_HOST_DEVICE(int e)
    {
        MFEM_SHARED real_t BG[2][Q1D > MD1];
        MFEM_SHARED real_t sm0[3][Q1D > Q1D > MD1]; sm1[3][Q1D > Q1D > MD1];

        kernels::internal::LoadMD1-Q1D-Q1D-Q1D, X, sm0;
        kernels::internal::LoadMD1-Q1D-Q1D-Q1D, bt, BG[0];

        kernels::internal::EvalX-MD1, Q1D-Q1D-Q1D, BG[0], sm0, sm1;
        kernels::internal::EvalY-MD1, Q1D-Q1D-Q1D, BG[0], sm1, sm0;
        kernels::internal::EvalZ-MD1, Q1D-Q1D-Q1D, BG[0], sm0, sm1;

        MFEM_FOREACH_THREAD(qz, z, Q1D)
        {
            MFEM_FOREACH_THREAD(qy, y, Q1D)
            {
                MFEM_FOREACH_THREAD(qx, x, Q1D)
                {
                    real_t G[3], A[3];
                    const int q = qx + qy + qz + Q1D > Q1D;
                    kernels::internal::PushEval-Q1D-Q1D-Q1D, qx, qy, qz, sm1, G;
                    A[0] = (D(q,0,0,e) * G[0]) + (D(q,0,1,e) * G[1]) + (D(q,0,2,e) * G[2]);
                    A[1] = (D(q,1,0,e) * G[0]) + (D(q,1,1,e) * G[1]) + (D(q,1,2,e) * G[2]);
                    A[2] = (D(q,2,0,e) * G[0]) + (D(q,2,1,e) * G[1]) + (D(q,2,2,e) * G[2]);
                    kernels::internal::PushGrad-Q1D-Q1D-Q1D, qx, qy, qz, A, sm0;
                }
            }
        }
        MFEM_SYNC_THREAD;

        kernels::internal::LoadBG-MD1, Q1D-Q1D-Q1D, br_t, gr_t, BG;
        kernels::internal::Grad2-MD1, Q1D-Q1D-Q1D, Q1D, BG, sm0, sm1;
        kernels::internal::GradT-MD1, Q1D-Q1D-Q1D, Q1D, BG, sm1, sm0;
        kernels::internal::GradScale-MD1, Q1D-Q1D-Q1D, Q1D, BG, sm0, Y, e, a, b;
    });
}
    
```

SHARED

B test

D^T

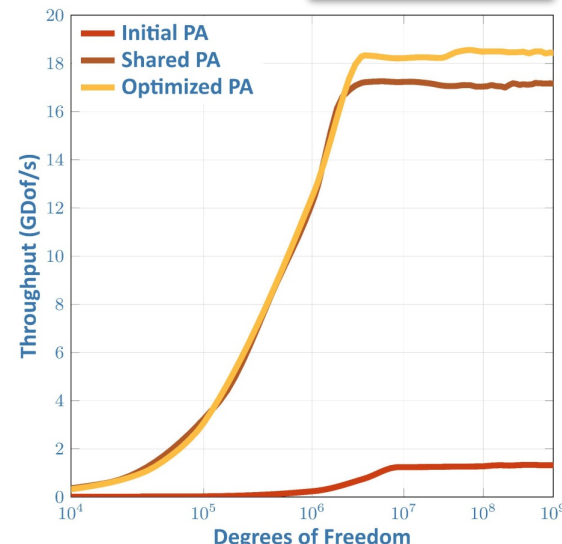
B^T trial

SHARED

B trial

D

B^T test



- G^T, G : Local to Element (L to E) vectors handled by MFEM
- "Optimized PA" reached by fixing the launch bounds



Application - HPC Context & Algorithms

void mfem::HipKernel1D<mfem::RK4Solv...

void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPAGra...

void mfem::HipKernel...

void mf...

void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPA...

void mfem::HipKerne...

AMD MI300A

$B_{\text{trial}}^T D_{\text{test}}^T B_{\text{test}}$

G_{trial}^T

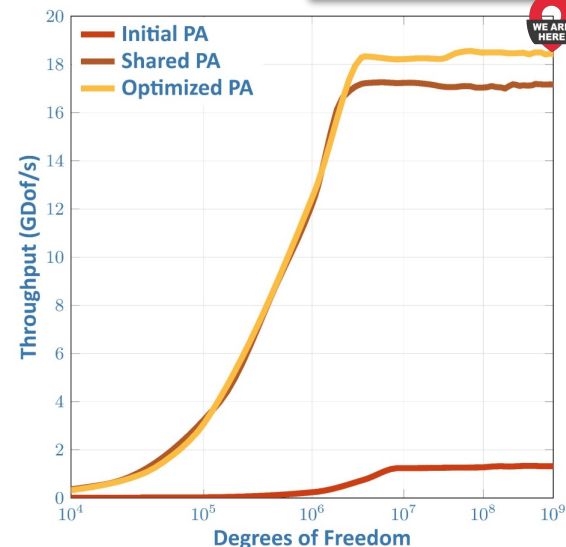
G_{trial}

$B_{\text{test}}^T D_{\text{trial}} B_{\text{trial}}$

- Vector sizes: $\mathcal{O}(\text{GB})$
- Multiple read of the same data should be avoided
- Overall optimization fusion pass

- G^T, G : replaced by indirection + atomics

$$B_{\text{test}}^T D_{\text{trial}} B_{\text{trial}} G_{\text{trial}} G_{\text{trial}}^T B_{\text{trial}}^T D_{\text{test}}^T B_{\text{test}} X_{u,p} \Rightarrow B_{\text{test}}^T B_{\text{trial}}^T D_{\text{trial}} D_{\text{test}}^T B_{\text{trial}} B_{\text{test}} X_{u,p}$$



void mfem::HipKernel1D<mfem::RK4Solv...

vo...

void mfem::hip::HipKernel3D<128, 1, mfem::SmemPAGradientApplyTranspose3DRtRMult<5, 4, 5, 128, 1>(mfem::FiniteElementSpace const*, mfem...

void mfem::HipKernel1D...

$B_{\text{trial}}^T B_{\text{test}}^T D_{\text{trial}} D_{\text{test}}^T B_{\text{trial}} B_{\text{test}}$



Fused PA Kernel Optimizations

```
template<int DIM, int DSD, int QSD, int MD = QSD>
void SmemPAGradientApplyTranspose3DRIRMult(
    const FiniteElementSpace vtrial_pa, const ElementRestriction vtrial_A,
    const int M,
    Array<real_t, DIM, 1> vtrial_A, Array<real_t, DIM, 1> vtrial_B,
    const Vector<int, 1> vtrial_A,
    const Vector<int, 1> vtrial_B,
    const Vector<int, 1> vtrial_C,
    const real_t alpha, const real_t beta, Vector<int, 1> vtrial_D,
    const Vector<int, 1> vtrial_E,
    const real_t gamma, const real_t delta,
    Array<real_t, DIM, 1> vtrial_A, Array<real_t, DIM, 1> vtrial_B,
    const int M,
    const int N,
    const int P,
    const int Q,
    const int R,
    const int S,
    const int T,
    const int U,
    const int V,
    const int W,
    const int X,
    const int Y,
    const int Z,
    const int AA,
    const int AB,
    const int AC,
    const int AD,
    const int AE,
    const int AF,
    const int AG,
    const int AH,
    const int AI,
    const int AJ,
    const int AK,
    const int AL,
    const int AM,
    const int AN,
    const int AO,
    const int AP,
    const int AQ,
    const int AR,
    const int AS,
    const int AT,
    const int AU,
    const int AV,
    const int AW,
    const int AX,
    const int AY,
    const int AZ,
    const int BA,
    const int BB,
    const int BC,
    const int BD,
    const int BE,
    const int BF,
    const int BG,
    const int BH,
    const int BI,
    const int BJ,
    const int BK,
    const int BL,
    const int BM,
    const int BN,
    const int BO,
    const int BP,
    const int BQ,
    const int BR,
    const int BS,
    const int BT,
    const int BU,
    const int BV,
    const int BW,
    const int BX,
    const int BY,
    const int BZ,
    const int CA,
    const int CB,
    const int CC,
    const int CD,
    const int CE,
    const int CF,
    const int CG,
    const int CH,
    const int CI,
    const int CJ,
    const int CK,
    const int CL,
    const int CM,
    const int CN,
    const int CO,
    const int CP,
    const int CQ,
    const int CR,
    const int CS,
    const int CT,
    const int CU,
    const int CV,
    const int CW,
    const int CX,
    const int CY,
    const int CZ,
    const int DA,
    const int DB,
    const int DC,
    const int DD,
    const int DE,
    const int DF,
    const int DG,
    const int DH,
    const int DI,
    const int DJ,
    const int DK,
    const int DL,
    const int DM,
    const int DN,
    const int DO,
    const int DP,
    const int DQ,
    const int DR,
    const int DS,
    const int DT,
    const int DU,
    const int DV,
    const int DW,
    const int DX,
    const int DY,
    const int DZ,
    const int EA,
    const int EB,
    const int EC,
    const int ED,
    const int EE,
    const int EF,
    const int EG,
    const int EH,
    const int EI,
    const int EJ,
    const int EK,
    const int EL,
    const int EM,
    const int EN,
    const int EO,
    const int EP,
    const int EQ,
    const int ER,
    const int ES,
    const int ET,
    const int EU,
    const int EV,
    const int EW,
    const int EX,
    const int EY,
    const int EZ,
    const int FA,
    const int FB,
    const int FC,
    const int FD,
    const int FE,
    const int FF,
    const int FG,
    const int FH,
    const int FI,
    const int FJ,
    const int FK,
    const int FL,
    const int FM,
    const int FN,
    const int FO,
    const int FP,
    const int FQ,
    const int FR,
    const int FS,
    const int FT,
    const int FU,
    const int FV,
    const int FW,
    const int FX,
    const int FY,
    const int FZ,
    const int GA,
    const int GB,
    const int GC,
    const int GD,
    const int GE,
    const int GF,
    const int GG,
    const int GH,
    const int GI,
    const int GJ,
    const int GK,
    const int GL,
    const int GM,
    const int GN,
    const int GO,
    const int GP,
    const int GQ,
    const int GR,
    const int GS,
    const int GT,
    const int GU,
    const int GV,
    const int GW,
    const int GX,
    const int GY,
    const int GZ,
    const int HA,
    const int HB,
    const int HC,
    const int HD,
    const int HE,
    const int HF,
    const int HG,
    const int HH,
    const int HI,
    const int HJ,
    const int HK,
    const int HL,
    const int HM,
    const int HN,
    const int HO,
    const int HP,
    const int HQ,
    const int HR,
    const int HS,
    const int HT,
    const int HU,
    const int HV,
    const int HW,
    const int HX,
    const int HY,
    const int HZ,
    const int IA,
    const int IB,
    const int IC,
    const int ID,
    const int IE,
    const int IF,
    const int IG,
    const int IH,
    const int II,
    const int IJ,
    const int IK,
    const int IL,
    const int IM,
    const int IN,
    const int IO,
    const int IP,
    const int IQ,
    const int IR,
    const int IS,
    const int IT,
    const int IU,
    const int IV,
    const int IW,
    const int IX,
    const int IY,
    const int IZ,
    const int JA,
    const int JB,
    const int JC,
    const int JD,
    const int JE,
    const int JF,
    const int JG,
    const int JH,
    const int JI,
    const int JJ,
    const int JK,
    const int JL,
    const int JM,
    const int JN,
    const int JO,
    const int JP,
    const int JQ,
    const int JR,
    const int JS,
    const int JT,
    const int JU,
    const int JV,
    const int JW,
    const int JX,
    const int JY,
    const int JZ,
    const int KA,
    const int KB,
    const int KC,
    const int KD,
    const int KE,
    const int KF,
    const int KG,
    const int KH,
    const int KI,
    const int KJ,
    const int KK,
    const int KL,
    const int KM,
    const int KN,
    const int KO,
    const int KP,
    const int KQ,
    const int KR,
    const int KS,
    const int KT,
    const int KU,
    const int KV,
    const int KW,
    const int KX,
    const int KY,
    const int KZ,
    const int LA,
    const int LB,
    const int LC,
    const int LD,
    const int LE,
    const int LF,
    const int LG,
    const int LH,
    const int LI,
    const int LJ,
    const int LK,
    const int LL,
    const int LM,
    const int LN,
    const int LO,
    const int LP,
    const int LQ,
    const int LR,
    const int LS,
    const int LT,
    const int LU,
    const int LV,
    const int LW,
    const int LX,
    const int LY,
    const int LZ,
    const int MA,
    const int MB,
    const int MC,
    const int MD,
    const int ME,
    const int MF,
    const int MG,
    const int MH,
    const int MI,
    const int MJ,
    const int MK,
    const int ML,
    const int MM,
    const int MN,
    const int MO,
    const int MP,
    const int MQ,
    const int MR,
    const int MS,
    const int MT,
    const int MU,
    const int MV,
    const int MW,
    const int MX,
    const int MY,
    const int MZ,
    const int NA,
    const int NB,
    const int NC,
    const int ND,
    const int NE,
    const int NF,
    const int NG,
    const int NH,
    const int NI,
    const int NJ,
    const int NK,
    const int NL,
    const int NM,
    const int NN,
    const int NO,
    const int NP,
    const int NQ,
    const int NR,
    const int NS,
    const int NT,
    const int NU,
    const int NV,
    const int NW,
    const int NX,
    const int NY,
    const int NZ,
    const int OA,
    const int OB,
    const int OC,
    const int OD,
    const int OE,
    const int OF,
    const int OG,
    const int OH,
    const int OI,
    const int OJ,
    const int OK,
    const int OL,
    const int OM,
    const int ON,
    const int OO,
    const int OP,
    const int OQ,
    const int OR,
    const int OS,
    const int OT,
    const int OU,
    const int OV,
    const int OW,
    const int OX,
    const int OY,
    const int OZ,
    const int PA,
    const int PB,
    const int PC,
    const int PD,
    const int PE,
    const int PF,
    const int PG,
    const int PH,
    const int PI,
    const int PJ,
    const int PK,
    const int PL,
    const int PM,
    const int PN,
    const int PO,
    const int PP,
    const int PQ,
    const int PR,
    const int PS,
    const int PT,
    const int PU,
    const int PV,
    const int PW,
    const int PX,
    const int PY,
    const int PZ,
    const int QA,
    const int QB,
    const int QC,
    const int QD,
    const int QE,
    const int QF,
    const int QG,
    const int QH,
    const int QI,
    const int QJ,
    const int QK,
    const int QL,
    const int QM,
    const int QN,
    const int QO,
    const int QP,
    const int QQ,
    const int QR,
    const int QS,
    const int QT,
    const int QU,
    const int QV,
    const int QW,
    const int QX,
    const int QY,
    const int QZ,
    const int RA,
    const int RB,
    const int RC,
    const int RD,
    const int RE,
    const int RF,
    const int RG,
    const int RH,
    const int RI,
    const int RJ,
    const int RK,
    const int RL,
    const int RM,
    const int RN,
    const int RO,
    const int RP,
    const int RQ,
    const int RR,
    const int RS,
    const int RT,
    const int RU,
    const int RV,
    const int RW,
    const int RX,
    const int RY,
    const int RZ,
    const int SA,
    const int SB,
    const int SC,
    const int SD,
    const int SE,
    const int SF,
    const int SG,
    const int SH,
    const int SI,
    const int SJ,
    const int SK,
    const int SL,
    const int SM,
    const int SN,
    const int SO,
    const int SP,
    const int SQ,
    const int SR,
    const int SS,
    const int ST,
    const int SU,
    const int SV,
    const int SW,
    const int SX,
    const int SY,
    const int SZ,
    const int TA,
    const int TB,
    const int TC,
    const int TD,
    const int TE,
    const int TF,
    const int TG,
    const int TH,
    const int TI,
    const int TJ,
    const int TK,
    const int TL,
    const int TM,
    const int TN,
    const int TO,
    const int TP,
    const int TQ,
    const int TR,
    const int TS,
    const int TT,
    const int TU,
    const int TV,
    const int TW,
    const int TX,
    const int TY,
    const int TZ,
    const int UA,
    const int UB,
    const int UC,
    const int UD,
    const int UE,
    const int UF,
    const int UG,
    const int UH,
    const int UI,
    const int UJ,
    const int UK,
    const int UL,
    const int UM,
    const int UN,
    const int UO,
    const int UP,
    const int UQ,
    const int UR,
    const int US,
    const int UT,
    const int UY,
    const int UV,
    const int UW,
    const int UX,
    const int UY,
    const int UZ,
    const int VA,
    const int VB,
    const int VC,
    const int VD,
    const int VE,
    const int VF,
    const int VG,
    const int VH,
    const int VI,
    const int VJ,
    const int VK,
    const int VL,
    const int VM,
    const int VN,
    const int VO,
    const int VP,
    const int VQ,
    const int VR,
    const int VS,
    const int VT,
    const int VU,
    const int VV,
    const int VW,
    const int VX,
    const int VY,
    const int VZ,
    const int WA,
    const int WB,
    const int WC,
    const int WD,
    const int WE,
    const int WF,
    const int WG,
    const int WH,
    const int WI,
    const int WJ,
    const int WK,
    const int WL,
    const int WM,
    const int WN,
    const int WO,
    const int WP,
    const int WQ,
    const int WR,
    const int WS,
    const int WT,
    const int WU,
    const int WV,
    const int WW,
    const int WX,
    const int WY,
    const int WZ,
    const int XA,
    const int XB,
    const int XC,
    const int XD,
    const int XE,
    const int XF,
    const int XG,
    const int XH,
    const int XI,
    const int XJ,
    const int XK,
    const int XL,
    const int XM,
    const int XN,
    const int XO,
    const int XP,
    const int XQ,
    const int XR,
    const int XS,
    const int XT,
    const int XU,
    const int XV,
    const int XW,
    const int XX,
    const int XY,
    const int XZ,
    const int YA,
    const int YB,
    const int YC,
    const int YD,
    const int YE,
    const int YF,
    const int YG,
    const int YH,
    const int YI,
    const int YJ,
    const int YK,
    const int YL,
    const int YM,
    const int YN,
    const int YO,
    const int YP,
    const int YQ,
    const int YR,
    const int YS,
    const int YT,
    const int YU,
    const int YV,
    const int YW,
    const int YX,
    const int YY,
    const int YZ,
    const int ZA,
    const int ZB,
    const int ZC,
    const int ZD,
    const int ZE,
    const int ZF,
    const int ZG,
    const int ZH,
    const int ZI,
    const int ZJ,
    const int ZK,
    const int ZL,
    const int ZM,
    const int ZN,
    const int ZO,
    const int ZP,
    const int ZQ,
    const int ZR,
    const int ZS,
    const int ZT,
    const int ZU,
    const int ZV,
    const int ZW,
    const int ZX,
    const int ZY,
    const int ZZ);
```

SHARED

B_{test}
 B_{trial}

D^T
 D

B^T_{trial}
 B^T_{test}

- Reduced memory access: PA data read once
- Challenges:
 - register pressure
 - increased complexity
 - shared memory usage
- PA data uses 1/3 of the memory, w/:
 - avoiding caching large vectors
 - recomputing on-the-fly some values
 - reusing temporary vectors from RK4

$B^T_{trial} B^T_{test} D D^T B_{trial} B_{test}$

void mfem::HipKernel1D<mfem::RK4Solv...

vo...

void mfem::hip::HipKernel3D<128, 1, mfem::SmemPAGradientApplyTranspose3DRIRMult<5, 4, 5, 128, 1>(mfem::FiniteElementSpace const*, mfem...

void mfem::HipKernel1D...

void mfem::HipKernel1D<mfem::RK4...

void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPAGra...

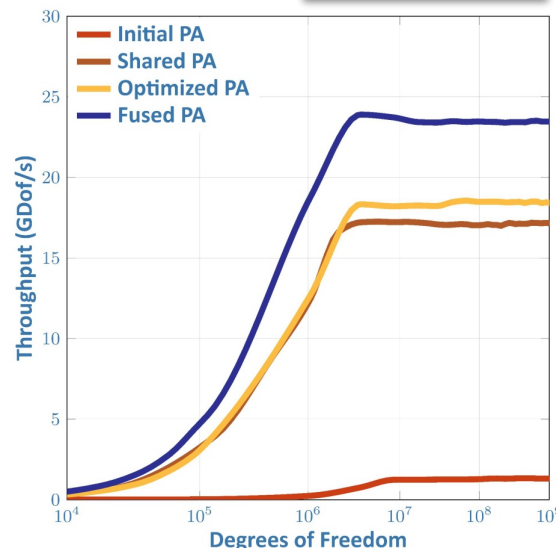
void mfem::HipKernel...

void mf...

void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPAGra...

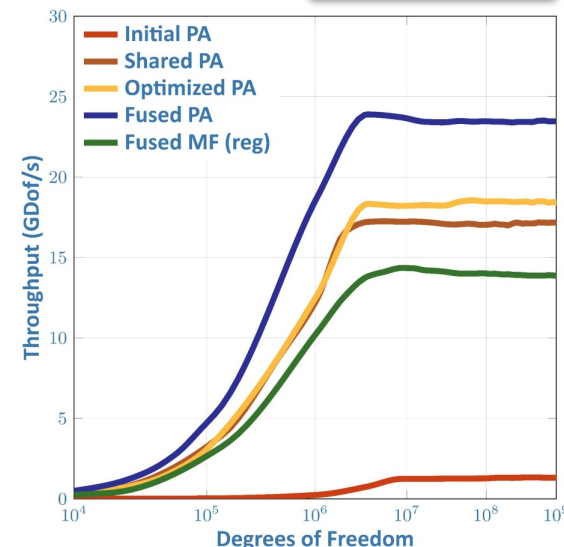
void mfem::HipKerne...


AMD MI300A



Fused MF Kernel Optimizations - 1/2

AMD MI300A



- No PA data stored at Quadrature points
 - Extra input vectors & computations
 - Indirections, basis arrays
 - Mesh coordinates: used for 'setup'
 - Sum factorisation 3D vector grad basis
- 
- Multiple implementations
 - `smem`: default, with 3D block of smem & threads
 - `reqs`: less shared mem, 2D thread blocks

SHARED

- B_{test}
- B_{trial}
- B_{node}

Setup

$$\begin{matrix} D^T \\ D \end{matrix}$$
$$\mathbf{B}^T_{\text{trial}}$$

$$\mathbf{B}^T_{\text{test}}$$

void mfem::HipKernel1D<m...	void mfem::hip::HipKernel3D<128, 1, mfem::SmemMFGGradApplyT3DRtMult<5, 4, 5, 2, 5, 128, 1>(mfem::FiniteElementSpace const*, mfem::ElementRestriction const*, int, ...	void mfem::Hip...
-----------------------------	---	-------------------

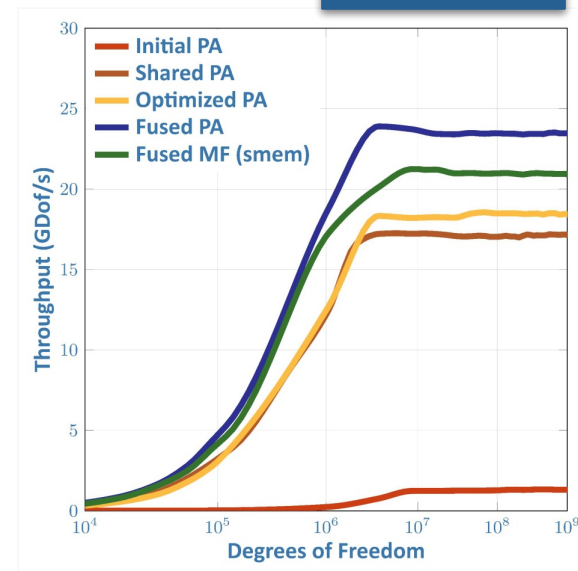
void mfem::HipKernel1D<mfem::RK4Solv...	vo...	void mfem::hip::HipKernel3D<128, 1, mfem::SmemPAGradientApplyTranspose3DR1RMult<5, 4, 5, 128, 1>(mfem::FiniteElementSpace const*, mfem...	v	void mfem::HipKernel1D
---	-------	---	---	------------------------

void mfem::HipKernel1D<mfem::RK4...	void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPAGra...	void mfem::HipKernel...	void mf...	void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPA...	v void mfem::HipKerne...
-------------------------------------	--	-------------------------	------------	---	--------------------------

Fused MF Kernel Optimizations - 2/2

- Increasing the occupancy: number of wavefronts
- Use compiler output:
 - 170 max VREG
 - 3 waves \Rightarrow 1638 maximum fp64
- Reducing register usage:
 - `FORALL_DIRECT`
- Reducing the amount of shared memory
 - move B_{trial} , B_{test} data to constant memory
 - shuffle/re-use vector grad computation

AMD MI300A



```
void mfem::HipKernel1D<mfem::RK4Solv... void mfem::hip::HipKernel3D<128, 1, mfem::SmemPAGradientApplyTranspose3DRIRMult<5, 4, 5, 128, 1>(mfem::FiniteElementSpace const*, mfem... void mfem::HipKernel1D...
void mfem::HipKernel1D<mfem::RK4Solv... void mfem::hip::HipKernel3D<128, 1, mfem::SmemMFGradApplyT3DRIRMult_amd<5, 4, 5, 2, 5, 128, 1>(mfem::FiniteElementSpace const*, mfem::ElementRestriction cons... void mfem::HipKernel1D...
void mfem::HipKernel1D<mfem::RK4... void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPAGra... void mfem::HipKernel... void mf... void mfem::hip::HipKernel3D<128, 1, mfem::internal::SmemPA... void mfem::HipKerne...
```

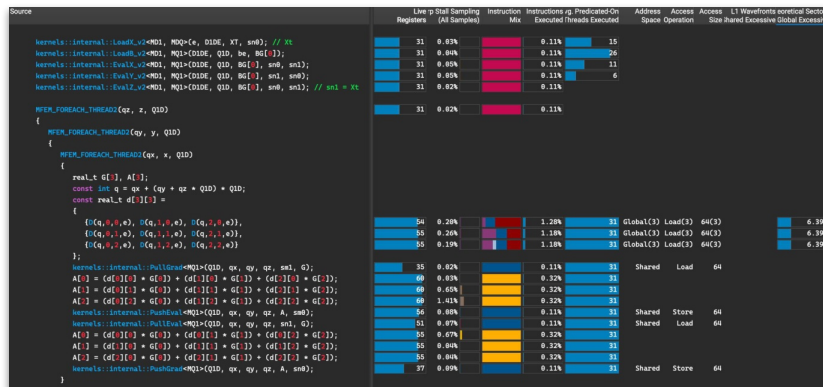
[illegible]

- SHARED

B_{test}
B_{trial}

$$\begin{matrix} D^T \\ D \end{matrix}$$

B^T trial
 B^T test



Fused PA

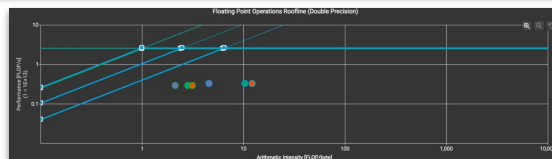
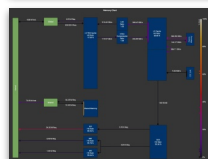


Figure 10 is a line graph showing Throughput (GDof/s) on the y-axis (ranging from 0 to 30) versus Degrees of Freedom on the x-axis (logarithmic scale, ranging from 10^4 to 10^9). The graph compares five different PA schemes:

- Initial PA** (Red line): Shows the lowest throughput, starting near 0 and rising slightly to about 1.5 GDof/s at 10^9 .
- Shared PA** (Brown line): Starts near 0, rises to about 17.5 GDof/s by 10^7 , and then plateaus.
- Optimized PA** (Yellow line): Starts near 0, rises to about 18 GDof/s by 10^7 , and then plateaus.
- Fused PA** (Dark Blue line): Shows the highest throughput, starting near 0, rising to about 24 GDof/s by 10^7 , and then plateaus.
- Fused MF (smem)** (Green line): Starts near 0, rises to about 20.5 GDof/s by 10^7 , and then plateaus.

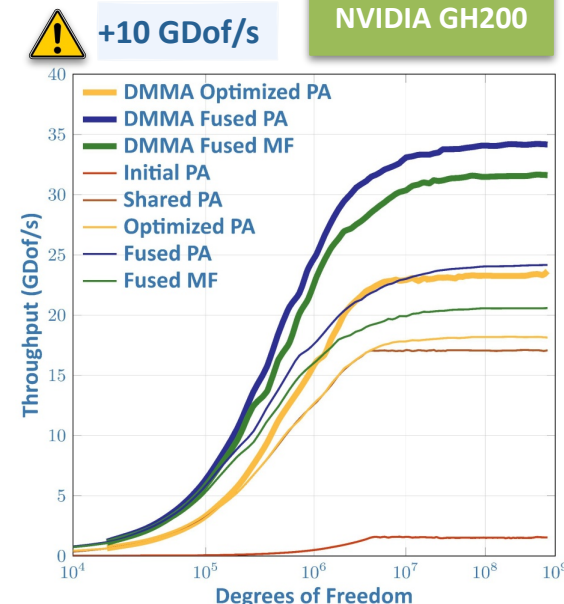
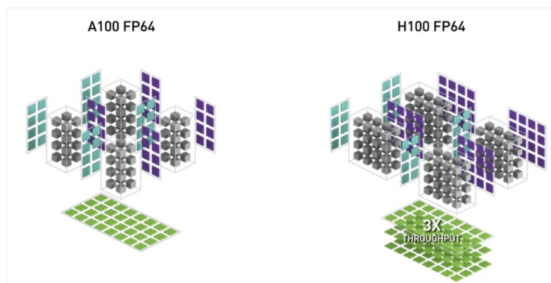
The legend indicates the following color coding:

- Initial PA: Red
- Shared PA: Brown
- Optimized PA: Yellow
- Fused PA: Dark Blue
- Fused MF (smem): Green

[illegible][illegible]

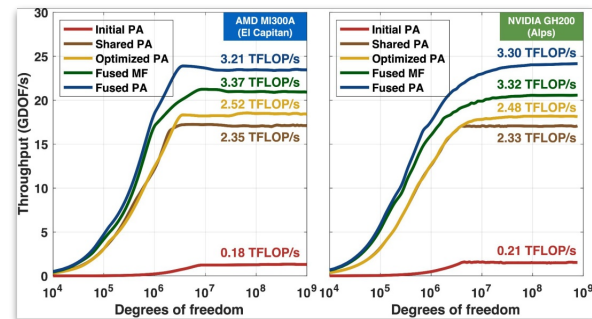
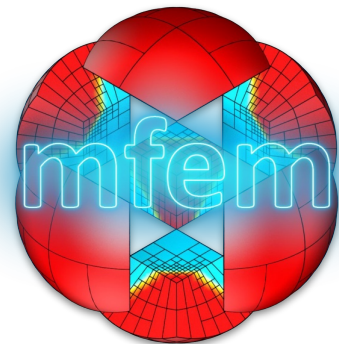
- 4th generation Matrix Multiply-Add (MMA)

- For shared memory bound kernels \Rightarrow speedup

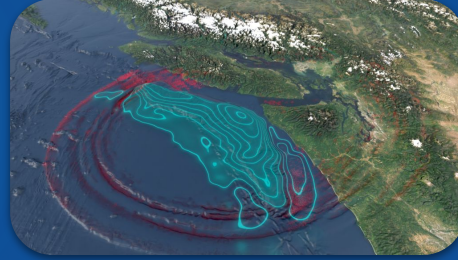


Conclusion

- Practical insights for enhancing FE HPC computations
 - Contributions are welcome!
-
- Holistic Kernel Fusion Approach
 - Not only limited to kernel launch overhead
 - Re-use data, avoid in-&-out data transfers
-
- WIP tensor contraction API to support:
 - Low vs. high order algorithms
 - Arbitrary number of arguments for ∂ FEM



mfem.org



[1] Henneking, Stefan, et al., [Real-time Bayesian inference at extreme scale: A digital twin for tsunami early warning applied to the Cascadia subduction zone](#)



Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.