High-Performance Python-based Simulations of Pressure and Temperature Waves in a Trace Gas Sensor

Brian Brennan and Robert C. Kirby
Department of Mathematics
Baylor University
One Bear Place #97328
Waco, TX 76798-7328
Email: b_brennan@baylor.edu, robert_kirby@baylor.edu

John Zweck and Susan E. Minkoff
Department of Mathematical Sciences
University of Texas at Dallas
800 West Campbell Road
Richardson, TX 75080-3021
Email: zweck@utdallas.edu, sminkoff@utdallas.edu

***Abstract*—We present a coupled model of temperature and pressure waves applicable to photoacoustic trace gas sensors. We discretize this model with finite elements using the Python-based FEniCS project.To validate the generated code, we observe optimal convergence rates to a plane wave solution in one and two dimensions. Using FEniCS' seamless access to MPI-based parallelism, we observe nearly-optimal speedup using four cores of a workstation, although not beyond. Finally, we use the automated mesh adaptivity of FEniCS to optimize the computation of heat flux along a portion of the boundary, arriving at comparable accuracy to uniform refinement using a factor of forty fewer cells.**

I. Introduction

Currently, research on trace gas sensors is focused on the development of portable, efficient, and cost-effective sensor technologies that can be deployed in networks for large scale monitoring of carbon dioxide and atmospheric pollutants, as well as for non-invasive disease diagnosis using breath analysis [1], [2], [3]. One such trace gas sensor is the Quartz-Enhanced Photo-Acoustic Spectroscopy (QEPAS) sensor which employs a quartz tuning fork to detect the weak acoustic pressure waves generated by the interaction of laser radiation with a trace gas [4]. More specifically, QEPAS sensors are based on the following physical mechanisms. A laser generates optical radiation at a specific absorption wavelength of the gas to be detected. The laser beam is directed between the tines of the tuning fork (see Figure 1). The optical energy that is absorbed by the trace gas is transformed into vibrational energy of the gas molecules, generating a temperature disturbance. If the interaction between the laser and the trace gas is sinusoidally modulated, this temperature disturbance is in the form of a thermal wave. In addition, vibrational to translational energy conversion processes in the gas molecules result in the generation of a weak acoustic pressure wave, which can be detected by the tuning fork. To amplify the signal detected by the tuning fork, the modulation frequency of the laser is chosen so as to excite a resonant vibration in the tuning fork. Finally, since quartz is a piezoelectric material, this mechanical vibration is converted to an electric current that can be measured. Because the entire process is linear, the measured current is proportional to the concentration of the trace gas. In some experimental regimes, the tuning fork can also be used to directly detect the thermal wave via the pyroelectric effect [5].

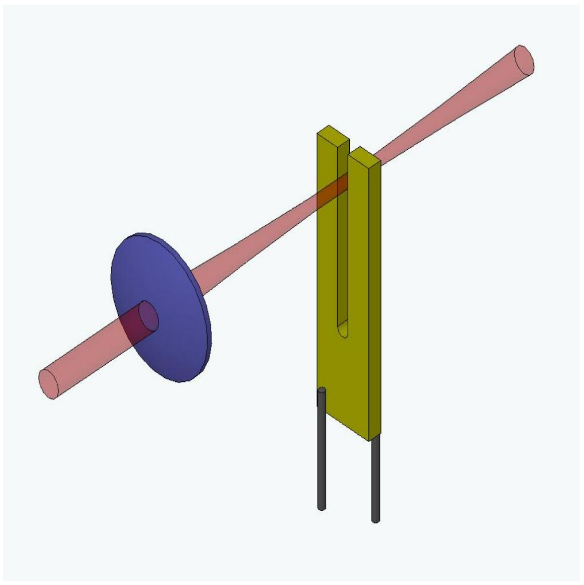

Fig. 1. Schematic diagram of the experimental setup for a QEPAS sensor showing the tuning fork (the U-shaped bar with two tines), two attached wires, and the laser source focused between the tines of the tuning fork.

To date, all mathematical models of QEPAS sensors [6], [7], [8], [9] have included damping in the model using an ad-hoc approach that involves making experimental measurements using the actual tuning fork being modeled. A major goal of our ongoing research is to develop a more realistic model of the damping in a QEPAS sensor. The primary source of damping is viscous damping of the fluid in a boundary layer surrounding the tuning fork. Therefore, in this paper we begin the development a computational model of a QEPAS sensor that realistically incorporates the effects of viscous damping using a parameter that depends on the physical properties of the fluid and is independent of the particular choice of tuning fork. The model is based on a coupled system of partial differential equations for the acoustic pressure and the temperature of the fluid that was derived by Morse and Ingard [10]. This coupled system generalizes the classical acoustic wave and heat equations.

The purpose of this paper is to develop some of the high-performance computational tools required to compute numerical solutions of the coupled pressure-temperature equations. We verify the correctness of the numerical implementation and study its performance using an artificial plane-wave solution. In future work, we will further develop the model and apply the computational tools developed here to compute solutions of the pressure-temperature equations for a QEPAS sensor.

We employ the Python interface to FEniCS to automate the process of solving the coupled pressure-temperature equations using the finite element method [11]. Our simulation results show that even in two spatial dimensions we need to invoke high-performance linear solver tools to compute the solution on a sufficiently fine mesh. FEniCS offers convenient access to the PETSc, Trilinos/Epetra, and uBlas linear algebra libraries [12], [13], [14].

Because boundary layer phenomena are expected to play a role in the performance of the sensor, it will be important to use a much finer mesh near the surface of the tuning fork than away from it. To automate the mesh refinement proccess, we make use of the automated goal-oriented error control algorithm implemented in FEniCS, which adaptively refines the mesh so as to minimize the error in a quantity of interest. In this paper, we choose the quantity of interest to be a local average of pressure or the gradient of the temperature on the surface of the tuning fork, since these quantities determine the vibration of the tuning fork.

II. Mathematical Model

The interaction of laser radiation with the trace gas generates an acoustic pressure wave, $\mathcal{P}$, and a thermal disturbance, $\mathcal{T}$. To model the effects of viscous damping and thermal conduction in the gas Morse and Ingard [10], derived a coupled system of pressure-temperature equations which generalizes the standard acoustic wave and heat equations. These equations are given by

$$\frac{\partial}{\partial t}\left(\mathcal{T}-\frac{\gamma-1}{\gamma\alpha}\mathcal{P}\right)-\ell_h c\Delta\mathcal{T}=\mathcal{S} \tag{1a}$$

$$\gamma\left(\frac{\partial^2}{\partial t^2}-\ell_v c\frac{\partial}{\partial t}\Delta\right)(\mathcal{P}-\alpha\mathcal{T})-c^2\Delta\mathcal{P}=0. \tag{1b}$$

Here ℓ_v and ℓ_h are characteristic lengths associated with the effects of fluid viscosity and thermal conduction, respectively, c is sound speed, γ is the ratio of the specific heat of the gas at constant pressure to that at constant volume, and $\alpha=\left(\frac{\partial P}{\partial T}\right)_v$ is the rate of change of ambient pressure with respect to ambient temperature at constant volume.

We model the interaction between the laser and the trace gas using the source term, $\mathcal{S}$, in equation (1a) given by

$$\mathcal{S}(x,t)=C\exp\left(-\frac{2[(x-x_s)^2+(z-z_s)^2]}{\sigma^2}\right)\exp(-i\omega t) \tag{2}$$

where C is a constant that is proportional to the concentration of the trace gas to be detected, (x_s, z_s) are the coordinates of the axis of the cylindrically symmetric Gaussian power profile of the laser beam, σ is the beam width and ω is the frequency of the periodic interaction between the laser radiation and the trace gas. The modulation frequency, ω, is chosen so as to excite a resonant vibration in the tuning fork. Since $\mathcal{S}(\mathbf{x},t)=S(\mathbf{x})\exp(-i\omega t)$ is periodic in time, so are the pressure and temperature. Substituting $\mathcal{P}(\mathbf{x},t)=P(\mathbf{x})\exp(-i\omega t)$ and $\mathcal{T}(\mathbf{x},t)=T(\mathbf{x})\exp(-i\omega t)$ into equations (1) we obtain the coupled system of Helmholtz equations

$$i\beta\omega\left(T-\frac{\gamma-1}{\gamma\alpha}P\right)+\beta\ell_h c\Delta T=S \tag{3a}$$

$$i\gamma(\omega^2-\ell_v ci\omega\Delta)(P-\alpha T)-ic^2\Delta P=0 \tag{3b}$$

where $\beta=\frac{\alpha^2\gamma^2\omega}{\gamma-1}$. Since complex numbers are not implemented in FEniCS, we separate equations (3a) and (3b) into real and imaginary parts. Setting $T=T_1+iT_2$ and $P=P_1+iP_2$, we obtain a system of four partial differential equations of the form $Au=b$, where

$$A=\begin{pmatrix} -\beta\ell_h c\Delta & \beta\omega & 0 & -\alpha\gamma\omega^2 \\ -\beta\omega & -\beta\ell_h c\Delta & \alpha\gamma\omega^2 & 0 \\ \alpha\gamma\ell_v c\omega\Delta & -\alpha\gamma\omega^2 & -\gamma\ell_v c\omega\Delta & \gamma\omega^2+c^2\Delta \\ \alpha\gamma\omega^2 & \alpha\gamma\ell_v c\omega\Delta & -(\gamma\omega^2+c^2\Delta) & -\gamma\ell_v c\omega\Delta \end{pmatrix}. \tag{4}$$

Note that equations (1) and (3) are valid in all spatial dimensions. In this paper, we compute solutions in 1D, 2D, and 3D.

III. Automating the Finite Element Method

The FEniCS Form Compiler FFC [15] automates an optimal stiffness matrix assembly algorithm by reading in a variational form, written in common mathematical notation, for just-in-time compilation. Assembling the global stiffness matrix involves a for loop over the elements of the mesh in which we calculate the local contribution due to each element and then add it into the global stiffness matrix. While simple in nature, the assembly of the stiffness matrix often takes up a significant portion of the run time. So efficiency in the assembly routine becomes a crucial goal, and the FEniCS project provides both generality and efficiency.

The first step of any finite element method is to translate the given differential equation into the corresponding variational problem: find $u\in V$ such that

$$a(u,v)=L(v)\qquad\forall v\in\widehat{V} \tag{5}$$

where V is the trial space and $\hat{V}$ is the test space [16], [17]. For example, for the Poisson equation $-\Delta u=f$, the bilinear form is defined by

$$a(u,v)=\int_\Omega\nabla u\cdot\nabla v dx=\langle\nabla u,\nabla v\rangle.$$

This definition of the variational form can be directly passed into the FEniCS Form Compiler in Python. The entire finite element assembly process described above is then automated in FEniCS. In Figure 2 we see a basic example of this code for the Poisson problem with Dirichlet conditions on the left and right sides of the square and homogenous Neumann conditions on the top and bottom.

```
from dolfin import *

# Create mesh and define function space
mesh = UnitSquareMesh(32, 32)
V = FunctionSpace(mesh, "Lagrange", 1)

# Define Dirichlet boundary (x = 0 or x = 1)
def boundary(x):
    return x[0] < DOLFIN_EPS or x[0] > 1.0 - DOLFIN_EPS

# Define boundary condition
u0 = Constant(0.0)
bc = DirichletBC(V, u0, boundary)

# Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
f = Expression('exp(-(pow(x[0],2) + pow(x[1],2))')
a = inner(grad(u), grad(v))*dx
L = f*v*dx

# Compute solution
u = Function(V)
solve(a == L, u, bc)
```

Fig. 2. FEniCS code for the Poisson equation on a square mesh. Note that in FEniCS code the spatial coordinates x and y are referenced with $x[0]$ and $x[1]$ respectively.

For our model, the bilinear form corresponding to (4) is

$$\begin{aligned} a(u,v) = {} & \beta\ell_h c\langle\nabla T_1,\nabla v_1\rangle + \beta\omega\langle T_2, v_1\rangle \\ & - \alpha\gamma\omega^2\langle P_2, v_1\rangle - \beta\omega\langle T_1, v_2\rangle \\ & + \beta\ell_h c\langle\nabla T_2,\nabla v_2\rangle + \alpha\gamma\omega^2\langle P_1, v_2\rangle \\ & - \alpha\gamma\ell_v c\omega\langle\nabla T_1,\nabla v_3\rangle - \alpha\gamma\omega^2\langle T_2, v_3\rangle \\ & + \gamma\ell_v c\omega\langle\nabla P_1,\nabla v_3\rangle + \gamma\omega^2\langle P_2, v_3\rangle \\ & - c^2\langle\nabla P_2,\nabla v_3\rangle + \alpha\gamma\omega^2\langle T_1, v_4\rangle \\ & - \alpha\gamma\ell_v c\omega\langle\nabla T_2,\nabla v_4\rangle - \gamma\omega^2\langle P_1, v_4\rangle \\ & + c^2\langle\nabla P_1,\nabla v_4\rangle + \gamma\ell_v c\omega\langle\nabla P_2,\nabla v_4\rangle \end{aligned}$$

and $L(v) = \langle S, v_1\rangle$.

In Figure 3 we can see that the FEniCS code for our model closely resembles that of the basic Poisson problem.

If FEniCS is built with a parallel linear algebra back end, such as PETSc or Trilinos, we can quickly implement high performance tests in Python using the Message Passing Interface (MPI). The mesh is automatically partitioned over the processors, using tools such as Scotch or ParMetis, which means the Python code is unchanged as we switch between serial and parallel runs.

IV. Numerical Results in Serial

In this section, we verify our FEniCS implementation of the finite-element solution of the pressure-temperature equations by comparison to a plane-wave solution of equations (3) derived by Morse and Ingard [10]. Here we will be using a dual eight-core Intel Xeon E5-2680 machine running at 2.7GHz with 128GB of RAM. This machine is running Linux Mint 13 (Maya). All results for plane-wave solutions are on the interval $\Omega_{1D} = [0, 0.25]$, the square $\Omega_{2D} = [0, 0.25] \times [0, 0.25]$ or the cube $\Omega_{3D} = [0, 0.25] \times [0, 0.25] \times [0, 0.25]$ where each length is measured in meters.

```
from dolfin import *

# Assign values to physical parameters

# Import mesh file and define boundary conditions

# Define mixed function space
V = FunctionSpace(mesh, "Lagrange", 1)
W = MixedFunctionSpace([V,V,V,V])

# Define variational problem
(T1,T2,P1,P2) = TrialFunction(W)
(v1,v2,v3,v4) = TestFunction(W)

S = Expression('exp(-(pow(x[0],2) + pow(x[1],2))')

a = ( B*lh*c*inner(grad(T1),grad(v1))
+B*w*inner(T2,v1)-a*g*w*w*inner(P2,v1)
-B*w*inner(T1,v2)+B*lh*c*inner(grad(T2),grad(v2))
+a*g*w*w*inner(P1,v2)-a*g*lv*c*inner(grad(T1),grad(v3))
-a*g*w*w*inner(T2,v3)+g*lv*c*w*inner(grad(P1),grad(v3))
+g*w*w*inner(P2,v3)-c*c*inner(grad(P2),grad(v3))
+a*g*w*w*inner(T1,v4)-a*g*lv*c*inner(grad(T2),grad(v4))
-g*w*w*inner(P1,v4)+c*c*inner(grad(P1),grad(v4))
+g*lv*c*w*inner(grad(P2),grad(v4)) )*dx

L = S*v1*dx

# Compute solution
u = Function(W)
problem = LinearVariationalProblem(a, L, u, bcs)
solver = LinearVariationalSolver(problem)
solver.solve()
(T1,T2,P1,P2) = u.split()
```

Fig. 3. FEniCS code for the variational problem $a(u, v) = L$ corresponding to the pressure-temperature model in equations (3) with appropriate boundary conditions defined by `bcs`. The mixed function space `W` is the Cartesian product $V \times V \times V \times V$.

If we assume that the pressure wave in free-space is of the form

$$P(\mathbf{x}) = e^{i\mathbf{k}\cdot\mathbf{x}}, \tag{6}$$

where $\mathbf{k}$ is the wave vector, and set $S = 0$ in equation (3a) then the temperature, T, is the plane wave given by

$$T(\mathbf{x}) = \frac{i\omega(\gamma-1)}{(i\omega-\ell_h ck^2)\gamma\alpha}e^{i\mathbf{k}\cdot\mathbf{x}}, \tag{7}$$

where $k = |\mathbf{k}|$. Inserting equations (6) and (7) into equation (3) and dividing by $e^{i\mathbf{k}\cdot\mathbf{x}}$, we obtain a quadratic equation for k^2 whose solution is given by

$$k^2 = \frac{i\omega^2}{2\Omega c^2}\,\frac{1 - i\Upsilon - i\gamma\Omega \mp Q}{1 - i\gamma\Upsilon}, \tag{8}$$

where $\Omega = \frac{\omega}{c}\ell_h$, $\Upsilon = \frac{\omega}{c}\ell_v$ and

$$Q = \sqrt{(1 - i\Upsilon + i\gamma\Omega)^2 - 4i(\gamma-1)\Omega}.$$

The two signs in the definition of k correspond to particular physical modes. The minus sign represents the *propagational* mode while the plus sign represents the *thermal* mode. Here we have chosen to work with the equations corresponding to the *propagational* mode.

Most of the work up to this point has been independent of the spatial dimension. FEniCS allows us to run the very same code in 1D, 2D or 3D simply by changing to an appropriate mesh and respecifying the boundary conditions. In order to restrict our free-space plane-wave solution to a problem on a bounded domain, we can use the exact solutions `exact_T1`, `exact_T2`, `exact_P1`, `exact_P2` on a given 1D, 2D or 3D mesh to enforce the appropriate Dirichlet boundary conditions. This is easily accomplished in FEniCS with the commands

```
    bc1 = DirichletBC(W.sub(0), exact_T1, boundary)
    bc2 = DirichletBC(W.sub(1), exact_T2, boundary)
    bc3 = DirichletBC(W.sub(2), exact_P1, boundary)
    bc4 = DirichletBC(W.sub(3), exact_P2, boundary)
    bcs = [bc1, bc2, bc3, bc4]
```

where `W.sub(0)`, `W.sub(1)`, `W.sub(2)` and `W.sub(3)` represent the to be approximated sub-solutions `T1`, `T2`, `P1`, and `P2`, respectively, on the mixed vector space `W`.

To mimic a realistic problem, the following set of physical parameters will be used for all tests in this paper:

$$\begin{aligned} \ell_h &= \ell_v = 10^{-6} \text{ m} \\ c &= 300 \text{ m/s} \\ \omega &= 3.3 \times 10^4 \text{ Hz} \\ \gamma &= 1.4 \\ \alpha &= 8.8667 \text{ Pa/K}. \end{aligned}$$

Now that we have an exact solution in hand, we can verify the accuracy of our method while also checking that it is converging to the exact solution at the expected rate. For polynomials of order p, we expect the error to be $\mathcal{O}(h^{p+1})$ in L^2. We will use the Multifrontal Massively Parallel Solver (MUMPS) to solver all of our linear systems in serial and parallel [19], [20]. MUMPS uses a variation of Gaussian elimination to solve large, sparse linear equations. Implementing MUMPS in FEniCS requires replacing the `LinearVariationalSolver()` in Figure 3 with the following code segment.

```
A, b = assemble_system(aa, L, bcs)
solver = PETScLUSolver("mumps")
solver.solve(A, u.vector(), b)
```

First we consider the 1D problem. In Figure 4 we plot the relative error

$$\text{Relative Error}(u) = \frac{\|u - u_h\|}{\|u\|}, \tag{9}$$

where u represents the exact solution T_1, T_2, P_1 or P_2 and u_h is the corresponding finite element solution. We see that for piecewise linear basis functions the method is converging quadratically. After N is about 10^5, the error begins to increase slightly. We are not sure of the origin of this, but it is likely due to round off error and ill-conditioning of the stiffness matrix. Likewise, Figure 5 shows the method is $\mathcal{O}(h^3)$ if we switch to piecewise quadratic basis functions.

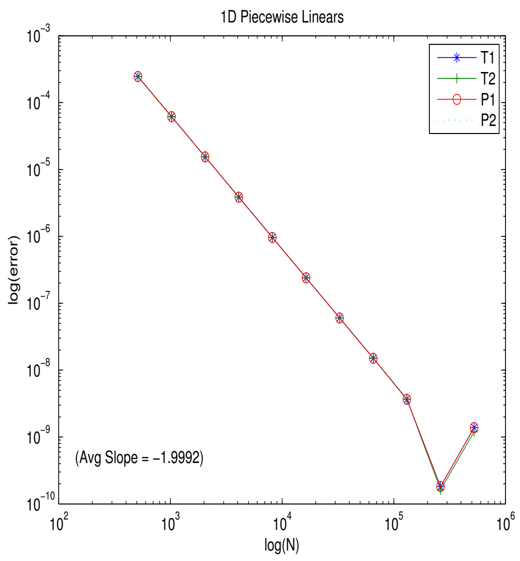

Fig. 4. log - log plot of the relative error in the solution for the 1D `T1`, `T2`, `P1`, and `P2` plane-wave solutions with respect to the mesh refinement parameter N. Here we are using piecewise linear Lagrange basis functions.

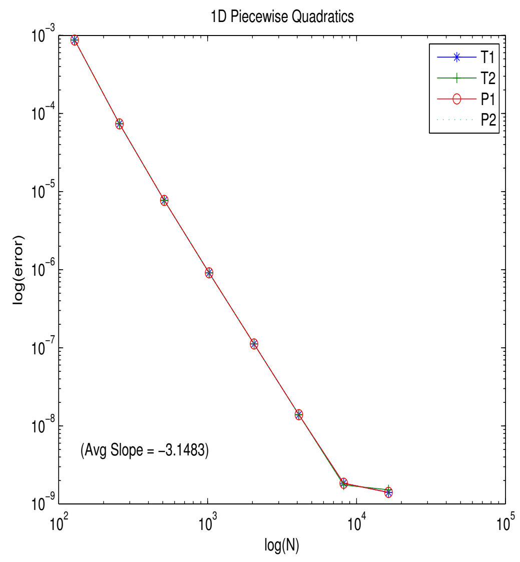

Fig. 5. log - log plot of the relative error in the solution for the 1D `T1`, `T2`, `P1`, and `P2` plane-wave solutions with respect to the mesh refinement parameter N. Here we are using piecewise quadratic Lagrange basis functions.

In two dimensions, the matrix is much larger and the required factorization of this matrix exhausts the available memory. Nevertheless, Figure 6 shows that when we can solve the 2D problem, the method converges quadratically for piecewise linear basis functions.

As we eventually move to the full 3D problem, these same memory constraints will become even greater and thus so does the need for and efficient, high-performance method.

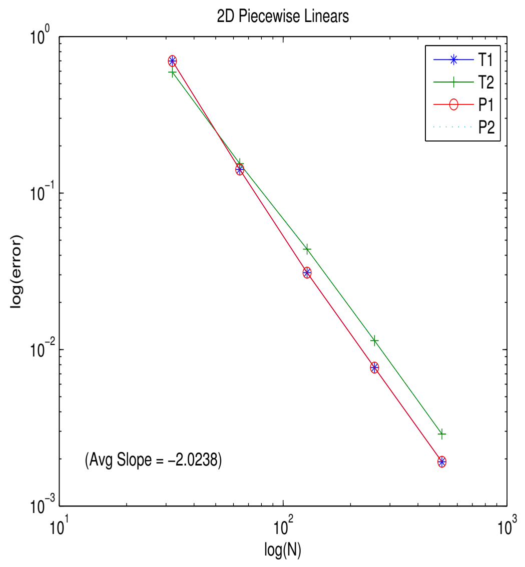

Fig. 6. log - log plot of the relative error in the MUMPS solution for the 2D `T1`, `T2`, `P1`, and `P2` plane-wave solutions with respect to the mesh refinement parameter N. Here we are using piecewise linear Lagrange basis functions.

V. Preliminary Intra-Node Parallel Results

When it is built with a parallel linear algebra backend, like PETSc or Trilinos/Epetra, FEniCS supports parallel computing on both shared memory machines and clusters. In order to avoid these potential complications, FEniCS has been designed such that the same code used for serial tests can be run in parallel using the following command.

```
mpirun -n 16 myprogram.py
```

While MUMPS is effective in serial, its major advantage is scalability on clusters. Our goal here is to demonstrate that FEniCS runs in MPI and to investigate possible speedup using multiple cores on our workstation. Obtaining optimal speedup using all cores is not likely for a direct solver, although MUMPS will enable us to scale well to distributed memory clusters in the near future.

In Figure 7 we can see that for the 256×256 square mesh, the speedup ratio increases as we apply more cores. We are even achieving almost optimal performance up to four cores before additional processors become less and less effective. Gaussian elimination requires quite a bit of memory so this is to be expected. We expect to see better performance as we move to a distributed memory cluster.

Next, we increase the size of the problem to a 512×512 mesh. By refining the mesh, we are giving more work to each processor which should yield better performance in terms of speedup. In Figure 8 we see that the speedup ratio more closely fits the optimal speedup line than it did for previous test. Also, the curve quickly flattens out after five processors rather than slowly curving off. This tells us that we are achieving optimal performance for up to five cores but that MUMPS is not able to utilize more cores on the same bus.

If we increase the problem size to a 1024×1024 mesh we see in Figure 9 that the speedup ratio increases with the number of processors rather than levelling off as it did for the smaller problems. This test is the largest 2D problem we are able to run on this workstation but we expect that when applied to larger problems, the speedup in our MUMPS solver will continue to scale with the number of processors.

We now begin to explore the 3D model where our issues with memory are even more severe. Hence we are limited to the $32 \times 32 \times 32$ mesh where Figure 10 shows very similar speedup results to the 2D cases. Again, we are achieving optimal speedup up to four cores.

For each of the tests in this section we have uniformly refined the mesh by a factor of two in each spatial dimension. Considering the strict memory limitations of the current implementation, we will next explore adaptive mesh refinement.

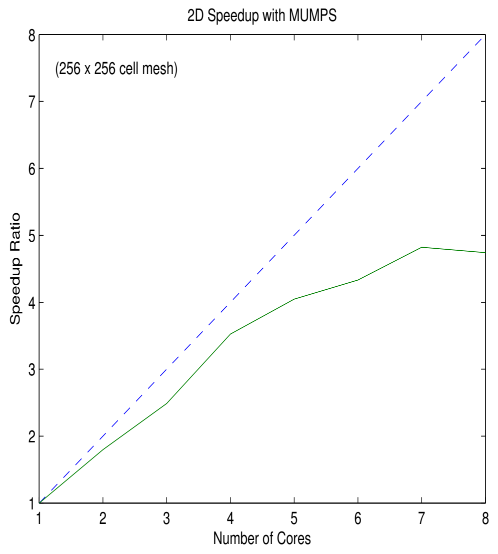

Fig. 7. Plot of the parallel speedup ratio on the 256×256 mesh.

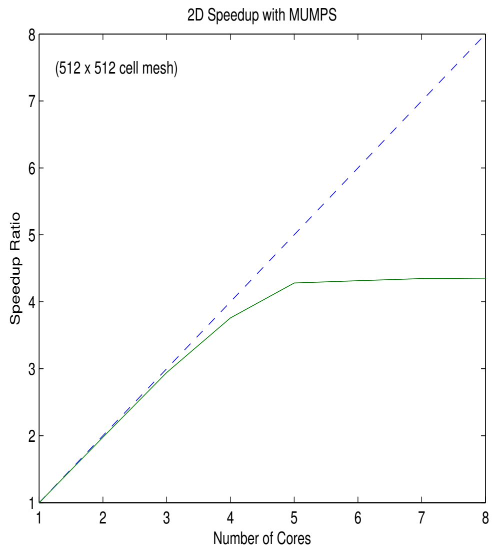

Fig. 8. Plot of the parallel speedup ratio on the 512×512 mesh.

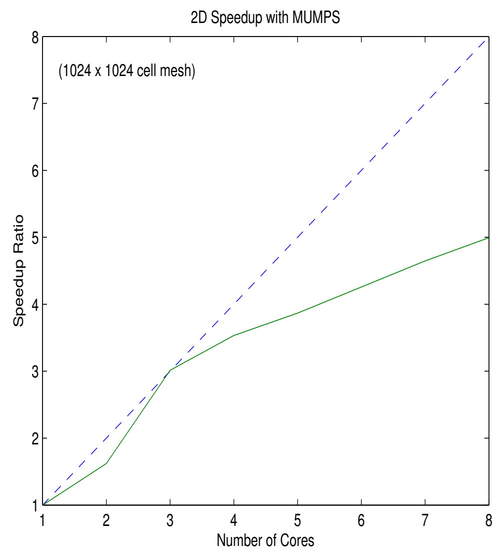

Fig. 9. Plot of the parallel speedup ratio on the 1024×1024 mesh.

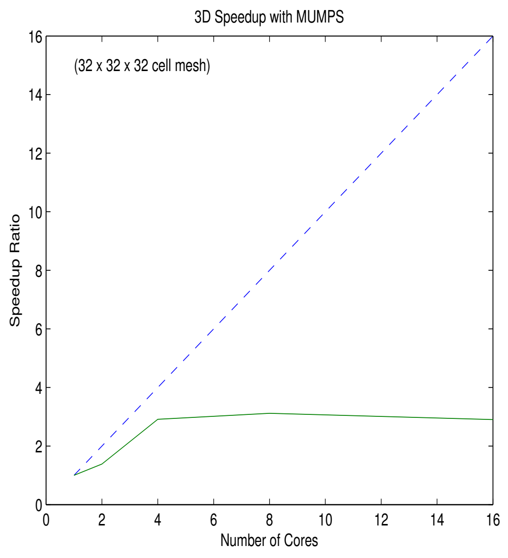

Fig. 10. Plot of the parallel speedup ratio on the $32 \times 32 \times 32$ mesh.

VI. Adaptive Mesh Refinement

We should remember that the goal is not simply to solve for the temperature and pressure waves on the entire domain but rather to efficiently approximate the heat flux as well as the magnitude of the pressure wave on the surface of the tuning fork.

Rather than uniform refinement, some kind of adaptive algorithm could be useful. An important strategy, developed in [21], is called *goal-oriented* adaptivity. By solving dual variational problems, Oden and Prudhomme were able to generate an adaptive algorithm that optimizes the error in some quantity of interest. In the simplest case, this quantity of interest is a linear functional of the computed solution. For example, we could consider the local average of the heat flux

$$M = \int_{\Gamma} \nabla T_1 \cdot \mathbf{n} \, ds, \tag{10}$$

where Γ is a section of the boundary and T_1 is the real part of the temperature.

Goal-oriented adaptivity gives an error estimate on the quantity $M(T_1) - M(T_{1,h})$ where $T_{1,h}$ is the computed solution, and it requires solving the dual variational problem

$$a^*(z, v) = M(v) \tag{11}$$

for all test functions v. The solution z acts as a weighting function for the elementwise residuals in the error estimates. When z is small, even a large residual might contribute very little to the actual error in M. In the case of linear partial differential equations, the dual variational problem just amounts to solving a linear system with the transpose of the original stiffness matrix.

The details of implementing goal-oriented estimation are rather delicate and beyond the scope of this paper, but fortunately FEniCS provides tools to automate the process. The high-level representation of the weak form in FEniCS can be used to automatically derive the dual variational problem, and the details of this implementation can be found in [22].

From the user's perspective, getting started on goal-oriented adaptivity in FEniCS is quite simple. This is done, while still using a MUMPS solver, simply by replacing the line

```
solver = PETScLUSolver("mumps")
```

with the commands

```
M = inner(nabla_grad([u0]), n) * ds(1)
solver = AdaptiveLinearVariationalSolver(problem, M)
prm = solver.parameters["linear_variational_solver"]
prm["linear_solver"] = "mumps"
```

where all nodes on the subdomain of interest Γ are marked as 1 rather than the default 0 which explains the $ds(1)$ in the definition of the goal functional M.

Suppose we are working on a square mesh while letting Γ in the definition of our goal functional (10) be the right edge of this square. First we will refine the mesh uniformly and calculate the value of the goal functional on each mesh. We see in Table VI that from $N = 64$ on through $N = 512$, we are converging linearly to the goal functional with respect to relative error

$$\text{Relative Error} = \frac{|M - M_h|}{|M|} \tag{12}$$

where M is the exact value of the goal functional and M_h is the computed value.

Now suppose we set the initial mesh of the adaptive code be an 8×8 square for the same Γ used in the uniform refinement test. The adaptive algorithm identifies and automatically refines the region near the right edge (see Figure 11). After 50 iterations of adaptive refinement the relative error in the goal functional is 1.78×10^{-2} with a final mesh of just $13,188$

N	Relative Error	Number of Cells	Run Time (s)
8	1.87	128	0.027
16	1.93	512	0.042
32	1.59×10^{-1}	2048	0.076
64	1.23×10^{-1}	8192	0.277
128	6.36×10^{-2}	32768	1.43
256	3.18×10^{-2}	131072	8.37
512	1.59×10^{-2}	524288	57.11

TABLE I. RELATIVE ERROR RESULTS IN THE GOAL FUNCTIONAL (10) ON THE RIGHT EDGE OF THE SQUARE THROUGH UNIFORM MESH REFINEMENT.

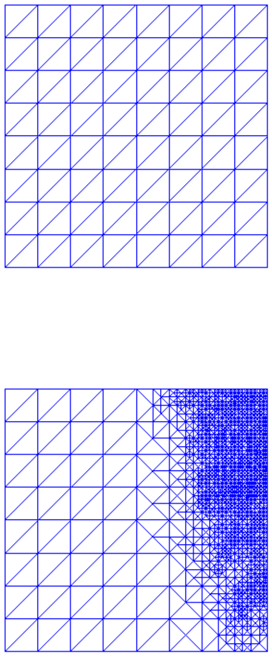

Fig. 11. The initial 8×8 mesh (top) along with the adaptively refined mesh (bottom) for the goal functional defined by (10).

cells and a complete run time of 24.75 seconds. This is a huge improvement over the uniform mesh refinement which required over $500,000$ cells to achieve approximately the same level of accuracy.

This preliminary result suggests that goal-oriented adaptive refinement will play an important role in future work on approximating solutions of equations (3) with the Gaussian source (2) around the 3D surface of the tuning fork.

VII. Conclusion

In this paper, we have seen how the Python interface of FEniCS has provided a powerful tool for the simulation of a nontrivial system of differential equations modeling trace gas sensors. The high-level syntax makes developing and validating a complex model relatively straightforward. However, the code generation and interface to high-performance tools means that the result is not merely a prototype – only mild changes to the boundary conditions and mesh source are required to change between dimensions, and the same code, with suitably chosen solver parameters, runs seamlessly in MPI-based parallelism.

Several topics will occupy our future study. Our study of parallel scaling, currently limited to simple workstation-based MPI jobs, has only recently begun. We are currently configuring FEniCS to run on the parallel cluster at Baylor University, on which we will be able to push MUMPS much further. We also plan to study iterative methods, including block-preconditioners, that should have lower memory requirements and scale to larger problems than direct methods. We hope to adapt techniques like those in [23].

VIII. Acknowledgments

The first two authors were supported by NSF grant CCF-1117794.

References

[1] A. Kosterev, G. Wysocki, Y. Bakhirkin, S. So, R. Lewicki, M. Fraser, F. Tittel, and R. F. Curl, "Application of quantum cascade lasers to trace gas analysis," *Appl. Phys. B*, vol. 90, pp. 165–176, 2007.

[2] Y. Zhang, J. A. Smith, A. Michel, M. Baeck, Z. Wang, J. Fast, and C. Gmachl, "Coupled monitoring and modeling of air quality and regional climate during the 2008 Beijing olympic games." San Francisco: American Geophysical Union, Fall Meeting 2009, 2009.

[3] M. McCurdy, Y. Bakhirkin, G. Wysocki, R. Lewicki, and F. Tittel, "Recent advances of laser-spectroscopy based techniques for applications in breath analysis," *Journal of Breath Research*, vol. 1, no. 1.

[4] A. Kosterev, Y. Bakhirkin, R. Curl, and F. Tittel, "Quartz-enhanced photoacoustic spectroscopy," *Optics Letters*, vol. 27, pp. 1902–1904, 2002.

[5] N. Petra, J. Zweck, S. Minkoff, A. Kosterev, and J. D. III, "Modeling and design optimization of a resonant optothermoacoutstic trace gas sensor," *SIAM J Appl Math*, vol. 71, pp. 309–332, 2001.

[6] M. Wojcik, M. Phillips, B. Cannon, and M. Taubman, "Gas-phase photoacoustic sensor at 8.41 m using quartz tuning forks and amplitude-modulated quantum cascade lasers," *Appl. Phys. B*, vol. 11, pp. 307–313, 2006.

[7] N. Petra, J. Zweck, A. Kosterev, S. Minkoff, and D. Thomazy, "Theoretical analysis of a quartz-enhanced photoacoustic spectroscopy sensor," *Applied Physics B: Lasers and Optics*, vol. 94, no. 4, pp. 673–680, 2009.

[8] S. Firebaugh, E. Terray, and L. Dong, "Optimization of resonator radial dimensions for quartz enhanced photoacoustic spectroscopy systems," *Proc. SPIE 8600, Laser Resonators, Microresonators, and Beam Control XV, 86001S*, 2013.

[9] H. Yi, K. Liu, S. Sun, W. Zhang, and X. Gao, "Theoretical analysis of off beam quartz-enhanced photoacoustic spectroscopy sensor," *Optics Communications*, 2012.

[10] P. Morse and K. Ingard, *Theoretical acoustics*. Princeton University Press.

[11] A. Logg, K. Mardal, and G. Wells, *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[12] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. McInnes, B. Smith, and H. Zhang, "PETSc users manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.4, 2013.

[13] M. Heroux, R. Bartlett, V. Howle, R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams, "An Overview of Trilinos," Sandia National Laboratories, Tech. Rep. 2003-2927, 2003.

[14] *BOOST C++ Libraries*, `http://www.boost.org`. [Online]. Available: http://www.boost.org

[15] R. Kirby and A. Logg, "A compiler for variational forms," *ACM Trans. Math. Software*, vol. 32, pp. 417–444, 2006.

[16] D. Braess, *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*. Cambridge: Cambridge University Press, 2001.

[17]

[18] G. Karypis and V. Kumar, "Metis: Unstructured graph partitioning and sparse matrix ordering system, version 4.0," http://www.cs.umn.edu/~metis, University of Minnesota, Minneapolis, MN, 2009.

[19] P. Amestoy, I. Duff, J. Koster, and J.-Y. L'Excellent, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

[20] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, "Hybrid scheduling for the parallel solution of linear systems," *Parallel Computing*, vol. 32, no. 2, pp. 136–156, 2006.

[21] J. Oden and S. Prudhomme, "Goal-oriented error estimation and adaptivity for the finite element method," *Computers and Mathematics with Applications*, vol. 41, no. 5, pp. 735–756, 2001.

[22] M. Rognes and A. Logg, "Automated goal-oriented error control i: Stationary variatonal problems," *SIAM Journal on Scientific Computing*, vol. 35, no. 3, pp. 173–193, 2013.

[23] V. Howle and R. Kirby, "Block preconditioners for finite element discretization of incompressible flow with thermal convection," *Numerical Linear Algebra with Applications*, vol. 19, no. 2, pp. 427–440, 2012.