The FeasNewt Benchmark

Todd S. Munson
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
Email: tmunson@mcs.anl.gov

Paul D. Hovland
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
Email: hovland@mcs.anl.gov

***Abstract*— We describe the FeasNewt mesh-quality optimization benchmark. The performance of the code is dominated by three phases—gradient evaluation, Hessian evaluation and assembly, and sparse matrix-vector products—that have very different mixtures of floating-point operations and memory access patterns. The code includes an optional runtime data- and iteration-reordering phase, making it suitable for research on irregular memory access patterns. Mesh-quality optimization (or "mesh smoothing") is an important ingredient in the solution of nonlinear partial differential equations (PDEs) as well as an excellent surrogate for finite-element or finite-volume PDE solvers.**

I. Introduction

The FeasNewt mesh-quality optimization code is a useful benchmark for two reasons. First, mesh-quality optimization is an important ingredient in the solution of partial differential equations (PDEs). Second, it is a compact and portable surrogate for the solution of nonlinear PDEs using unstructured meshes. The gradient evaluation, Hessian computation, and linear solves in FeasNewt have data access patterns and floating-point computations similar to the function evaluation, Jacobian computation, and linear solves in the solution of nonlinear PDEs discretized using a finite-element or finite-volume method. More details on the application, algorithm, and computational results can be found in [1], [2], [3].

II. Mesh-Quality Optimization

Discretization methods are common techniques for computing approximate solutions to PDEs [4], [5], [6]. These methods decompose the given domain into a finite set of elements, triangles or tetrahedrons, for example, to produce a mesh used within an approximation scheme. Several factors affect the accuracy of the solution to the PDE computed: the degree of the approximation scheme and the number of elements in the mesh [7] and the quality of the mesh [8], [9]. Optimizing the quality of the mesh prior to computing the approximate solution can improve the condition number of the linear systems solved [10], reduce the time taken to compute the solution [11], and increase the numerical accuracy.

The savings in computational time from using the optimized mesh can be substantial. One application we investigated was to solve the Navier-Stokes equations for a fluid with a moderate Reynolds number containing a dilute suspension of particles [12]. The approximate solution was obtained by applying a spectral element method to a hexahedral mesh. The original mesh had many regular elements, while the optimized mesh lost much of this structure. However, the spectral element method applied required 29 hours to compute a solution when using the original mesh, but only 20 hours when using the optimized mesh, a 30% reduction in time.

The optimization problem we solve computes positions for the vertices in a given mesh to improve the average element quality according to the inverse-mean ratio metric [13], [14], a shape-quality metric measuring the distance between a trial element and an ideal element, a regular tetrahedron, for example. The objective function for the resulting optimization problem is nonconvex. A simple inexact Newton method [15], [16] with an Armijo linesearch [17] is used to solve the optimization problem. A preconditioned conjugate gradient method with a block Jacobi preconditioner is used to compute the direction [18]. Each block is symmetric and positive definite [1], [19]. Therefore, a Cholesky factorization is applied to each diagonal block when computing the preconditioner.

We implement a simple framework that reads the description of a mesh from a file, constructs the unconstrained optimization problem, calls an optimization routine, and writes the solution back to a file. When reading the mesh, we check it for inverted elements and compute the vertices on the boundary of the mesh. The computation of the boundary vertices involves applying a radix sort to the list of faces and extracting the faces that appear exactly once, since these define the boundary. Optionally, the vertices and elements in the mesh can be reordered by using a breadth-first search ordering to

improve the locality of reference before applying the optimization routine. This feature is enabled by default but can be disabled by defining the `NOREORDER` flag when compiling the code.

The average inverse mean-ratio objective function requires that a value of plus infinity be returned whenever the volume constraints are not satisfied. Therefore, if the volume of at least one element is smaller than 1.0×10^{-14}, we consider the volume constraints to be violated, and the objective function is set to plus infinity.

The gradient and Hessian of the objective function are calculated analytically by assembling the gradients and Hessians for each element function into a vector and symmetric sparse matrix. Only the upper triangular part of the Hessian matrix is stored in a block compressed sparse row format. Each 3×3 block of the Hessian matrix corresponds to a vertex-vertex interaction in the original mesh. The number of indices we need to store is reduced by using the block scheme. In order to facilitate the assembly of the Hessian matrix, once the sparsity pattern is obtained, an additional vector is calculated that tells the offset into the Hessian matrix data vector where the element Hessians are to be accumulated. The gradient and Hessian of the elements with respect to vertices fixed on the boundary of the mesh are ignored.

The code for calculating the gradient of the element function uses the reverse mode of automatic differentiation [20], [21]. The code was written and refined by hand to eliminate unnecessary operations, resulting in a more efficient gradient evaluation. The Hessian calculation uses the forward mode of differentiation on the gradient evaluation. The resulting code was written by using matrix-matrix products for efficiency. An evaluation routine that computes only the gradient of the element has also been coded that requires fewer floating-point operations than the function plus gradient evaluation. A similar routine has been provided to compute only the Hessian evaluation.

The complete algorithm is as follows.

1) Obtain a description of the mesh.
   a) Read the mesh from a file.
   b) Find the boundary vertices by computing the exposed faces in the mesh.
   c) Check the resulting mesh to ensure that the consistent orientation conditions are satisfied at the initial point.
   d) Reorder the mesh by using a breadth-first search order starting from the vertex farthest from the origin.
2) Optimize the mesh.
   a) Compute the sparsity pattern for the Hessian matrix.
   b) Perform a function and gradient evaluation at the initial point.
   c) If $\|\nabla F(x^k)\|_2 < \tau$ or the iteration limit is reached, then go to Step 3.
   d) Compute the Hessian matrix, and calculate the block diagonal preconditioner and factorization.
   e) Apply the preconditioned conjugate gradient method to solve
   $$\nabla^2 F(x^k) d^k = -\nabla F(x^k).$$
      i) Compute the current residual.
      ii) Stop if the relative residual is small enough, a direction of negative curvature is encountered, or the conjugate gradient iteration limit is reached.
      iii) Perform matrix-vector product to obtain direction.
      iv) Update iterate, residual, and got to step 2.e.ii.
   f) Perform an Armijo linesearch.
   g) Update the current iterate, objective function, and gradient, increase the iteration count, and go to Step 2.c.
3) Output the mesh.
   a) Apply an inverse permutation to put the data back into the original ordering.
   b) Write the optimized mesh to a file.

III. Performance Assessment

The most time-consuming phases of FeasNewt are the gradient evaluation (gFcn), Hessian computation and assembly (hOnly), and sparse Matrix-vector product (matmul, the computational kernel of the conjugate gradient method used for linear solves). The input meshes, as created by a mesh generator, exhibit poor spatial and temporal locality. Consequently, FeasNewt implements an optional data- and iteration-reordering transformation phase. Data is reordered by using a hypergraph breadth-first search, and iterations are reordered by using consecutive packing on the iteration hypergraph. We have compared this reordering strategy to several other data- and iteration-reordering strategies and concluded that this strategy is optimal in most cases and nearly optimal otherwise [22].

With data- and iteration-reordering enabled, sparse matrix-vector product is memory-bandwidth limited on contemporary microprocessors. Given the reasonable assumption that the Hessian and vector are too large to fit in cache, each block index of the Hessian (integer), every

nonzero value of the Hessian (real), and every element of the input vector (real) must be read and every element of the output vector (real) must be written. The length of the vectors is vd, and there are v diagonal blocks, each with $d(d+1)/2$ nonzero values, and e off-diagonal blocks, each with d^2 nonzero values, where v is the number of vertices in the mesh, e is the number of edges, and d is the number of dimensions. The number of floating-point operations is $2d^2$ for each diagonal block and $4d^2$ for each off-diagonal block. Therefore, assuming 4-byte integers and 8-byte double-precision floating-point numbers, the maximum achievable performance is

$$\frac{2d^2(v+2e)}{4e+8d(2v+v(d+1)/2+ed)} \times \text{Bandwidth} \quad (1)$$

For the three-dimensional case considered here, the limit is

$$\frac{9v+18e}{48v+38e} \times \text{Bandwidth} \quad (2)$$

For a representative mesh (ductbig), $v = 177,887$ and $e = 2,359,092$, so the maximum achievable performance on a system offering 3000 MB/s of sustained memory bandwidth is $.45 \times 3000MF/s = 1.35GF/s$. Figure 1 shows the achieved performance for the FeasNewt application, relative to this memory bandwidth limit.

Without data- and iteration-reordering, the number of cache misses increases dramatically. On a 2.2 GHz Intel Xeon with 512 KB L2 cache, the number of TLB misses increases by a factor of more than 20, with an increase of nearly two orders of magnitude in the matmul routine, and the number of L1 and L2 misses increase by 40% and 160%, respectively. The resulting increase in run time is more than a factor of 2. Figure 2 shows the effects of reordering for the three main phases.

IV. Compiling and Running FeasNewt

FeasNewt can be downloaded from `http://www.mcs.anl.gov/~tmunson/codes/fnbench.tgz`. After unpacking, it can be compiled by editing the Makefile to indicate the compiler (CC) and flags (CFLAGS). It can then be executed by using

```
./tetOpt <inputfile> <outputfile>
```

For benchmarking, `ductbig.mesh` and `foambig.mesh` in the data directory are recommended as input files and `/dev/null` is a suitable output file. The remaining mesh files (duct*, foam5, gear, hook, and tire) can be used as training data. In order to build FeasNewt with data- and iteration-reordering disabled, the NOREORDER flag should be defined, for example by using `CFLAGS=-O3 -dNOREORDER`.

FeasNewt has been compiled and run on most Unix-like systems, including Intel and AMD processors running Linux, Apple workstations running MacOS X, the IBM BlueGene/L running K42/Linux, Intel processors running Windows with cygwin, and Sun workstations running Solaris. Performance results for these various platforms appear in Table I.

V. Conclusions

FeasNewt is a portable and compact application that is both important in its own right and an excellent surrogate for nonlinear PDE simulations using unstructured meshes. Such applications are among the most difficult to optimize on modern computer architectures, with deep cache hierarchies and limited memory bandwidth. The data- and iteration-reordering in FeasNewt can be disabled, allowing one to evaluate the ability of a compiler or architecture to mitigate the effects of irregular data access patterns in such applications. The gradient and Hessian evaluations in FeasNewt include many calls to intrinsic functions such as `pow`, a characteristic shared with many scientific applications and a feature that enables one to assess the quality of the math library implementation.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

References

[1] T. S. Munson, "Mesh shape-quality optimization using the inverse mean-ratio metric," Argonne National Laboratory, Argonne, Illinois, Preprint ANL/MCS-P1136-0304, 2004.

[2] L. F. Diachin, P. Knupp, T. Munson, and S. Shontz, "A comparison of two optimization methods for mesh quality improvement," Argonne National Laboratory, Argonne, Illinois, Preprint ANL/MCS-P1239-0305, 2005.

[3] T. S. Munson, "Optimizing the quality of mesh elements," Argonne National Laboratory, Argonne, Illinois, Preprint ANL/MCS-P1260-0605, 2005.

[4] S. C. Brenner and L. R. Scott, *The Mathematical Theory of Finite Element Methods*. New York: Springer-Verlag, 2002.

[5] M. O. Deville, P. F. Fischer, and E. H. Mund, *High-Order Methods for Incompressible Fluid Flows*. Cambridge: Cambridge University Press, 2002.

[6] L. N. Trefethan, *Spectral Element Methods in MATLAB*. Philadelphia, Pennsylvania: SIAM, 2000.

[7] I. Babuška and M. Suri, "The p and h-p versions of the finite element method, basic principles and properties," *SIAM Review*, vol. 36, pp. 578–632, 1994.

[8] M. Berzins, "Solution-based mesh quality for triangular and tetrahedral meshes," in *Proceedings of the Sixth International Meshing Roundtable*. Sandia National Laboratories, 1997, pp. 427–436.

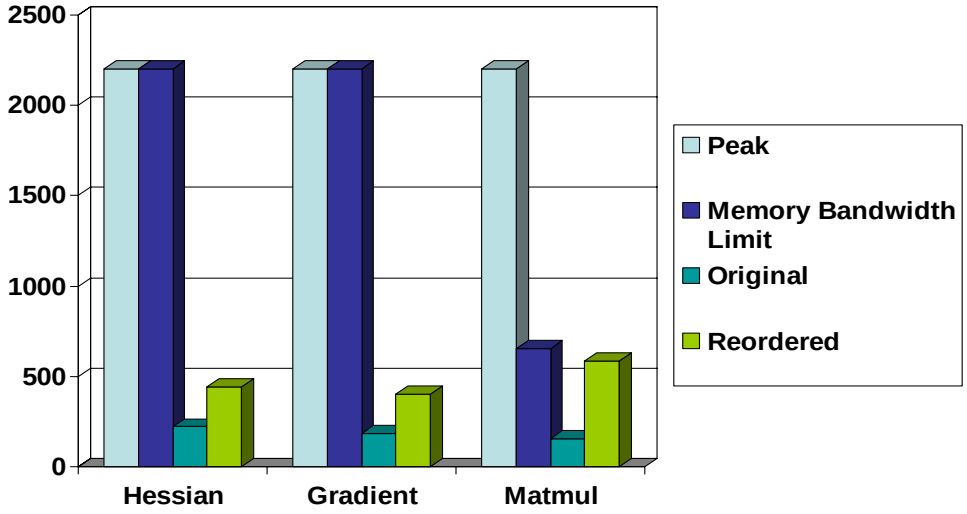

Fig. 1. Achieved performance (in MFlop/s), relative to the CPU peak and the limit imposed by the memory bandwidth of a 2.2 GHz Intel Xeon, for sparse matrix-vector multiplication, with and without data- and iteration- reordering.

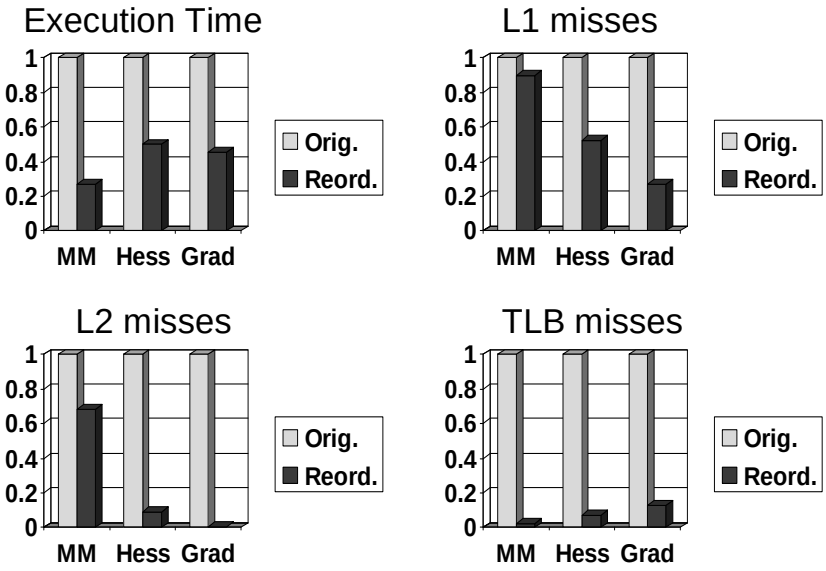

Fig. 2. Effects of data- and iteration- reordering on cache misses and run time for the matrix-vector multiplication (MM), gradient (Grad), and Hessian (Hess) phases of the FeasNewt application on a 2.2 GHz Intel Xeon processor, normalized against the original ordering.

TABLE I

PERFORMANCE OF THE FEASNEWT BENCHMARK ON SEVERAL PLATFORMS. TIMES (IN SECONDS) REPORTED ARE FOR THE DUCTBIG MESH AND EXCLUDE FILE I/O.

Processor	GHz	OS	Compiler	Flags	Runtime	MFlop/s
PowerPC G5	2.0	MacOS X	gcc 4.0	-fast	22.4	673
Opteron	1.6	Linux	gcc 3.3	-O3	27.6	546
Xeon	2.2	Linux	icc 8.1	-O3 -xN -ipo -static	33.1	455
Xeon	2.2	Linux	gcc 3.4	-O3	37.7	400
Pentium M	1.4	Windows	gcc 3.3	-O3	41.2	366
BlueGene	0.7	K42	xlc 7.0	-qbgl -qtune=440 -O5 -qarch=440	64.7	233
PowerPC G4	1.25	MacOS X	gcc 3.3	-fast -mcpu=7450	88.4	170
SPARC	0.9	Solaris	cc 5.1	-fast	106.	142
SPARC	0.9	Solaris	gcc 2.95	-O3	133.	113

[9] ——, "Mesh quality – geometry, error estimates or both?" in *Proceedings of the Seventh International Meshing Roundtable*. Sandia National Laboratories, 1998, pp. 229–237.

[10] J. Shewchuk, "What is a good linear element? Interpolation, conditioning, and quality measures," in *Proceedings of the Eleventh International Meshing Roundtable*. Sandia National Laboratories, 2002, pp. 115–126.

[11] L. Freitag and C. Ollivier-Gooch, "A cost/benefit analysis for simplicial mesh improvement techniques as measured by solution efficiency," *International Journal of Computational Geometry and Applications*, vol. 10, pp. 361–382, 2000.

[12] L. Zhang, S. Balachandar, P. Fischer, and T. Munson, Dec. 2004, Private communication.

[13] A. Liu and B. Joe, "Relationship between tetrahedron quality measures," *BIT*, vol. 34, pp. 268–287, 1994.

[14] P. Knupp, "Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities, Part II – A framework for volume mesh optimization and the condition number of the Jacobian matrix," *International Journal for Numerical Methods in Engineering*, vol. 48, pp. 1165–1185, 2000.

[15] C. T. Kelley, *Solving Nonlinear Equations with Newton's Method*. Philadelphia, Pennsylvania: SIAM, 2003.

[16] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer, 1999.

[17] L. Armijo, "Minimization of functions having Lipschitz-continuous first partial derivatives," *Pacific Journal of Mathematics*, vol. 16, pp. 1–3, 1966.

[18] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, Pennsylvania: SIAM, 2003.

[19] T. S. Munson, "Mesh shape-quality optimization using the inverse mean-ratio metric: Tetrahedral proofs," Argonne National Laboratory, Argonne, Illinois, Technical Memorandum ANL/MCS-TM-275, 2004.

[20] C. H. Bischof, P. D. Hovland, and B. Norris, "Implementation of automatic differentiation tools," *Higher-Order and Symbolic Computation*, to appear, 2004.

[21] A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Philadelphia, Pennsylvania: SIAM, 2000.

[22] M. M. Strout and P. D. Hovland, "Metrics and models for reordering transformations," in *Proceedings of the Second ACM SIGPLAN Workshop on Memory System Performance (MSP)*, June 2004, pp. 23–34.