Software Automation

L. Ridgway Scott[*], Peter Brune[†], Jeff Hammond[‡], Andy Terrel[§], Matt Knepley[¶]

November 16, 2007

1 Automatic generation of software

We describe a new pardigm in software development, which we refer to as *software automation*. This can be viewed as part of a natural evolution that has transformed systems research. The main ideas have their roots in programming languages and compilers.

We illustrate different areas of software development where this approach has proved essential. In particular, in the area of scientific software development, software automation has revolutionized an area with very demanding requirements. We give extensive examples.

Software automation has two components:

- (correct) translation from a high-level problem description to low-level executable code
- optimization of generated code.

As such, this describes the standard paradigm of a compiler for a programming language. What is different is the allowance for

- a hierarchical, or multifaceted, approach to the problem.

The typical programming language+compiler approach involves just two levels: the source language and the compiler output. In software automation, multiple levels exist, either hierarchically or in some more complex form of interaction.

The multifaceted approach of software automation facilitates leveraging inherent abstract representations of each segement of a software system. It can result in both better optimization and increased expressiveness in the problem description. In some cases, true optimization of generated code for a given segment can be performed, as opposed to just applying *ad hoc* transformations. it can both reduce cost and improve correctness of generated code.

[*]The Computation Institute and Departments of Computer Science and Mathematics, University of Chicago, Chicago, Illinois 60637-1581, USA.

[†]Department of Computer Science, University of Chicago, Chicago, Illinois 60637-1581, USA.

[‡]Department of Computer Science, University of Chicago, Chicago, Illinois 60637-1581, USA.

[§]Department of Computer Science, University of Chicago, Chicago, Illinois 60637-1581, USA.

[¶]Division of Mathematics and Computer Science, Argonne National Laboratory, Argonne, Illinois 60???, USA.

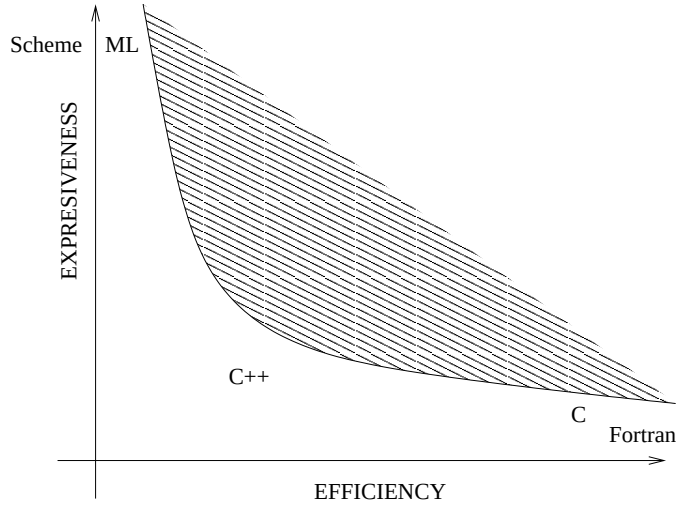

Figure 1: The conflict between efficiency and expressiveness [M. Dragichescu].

1.1 Evolution in systems research

In the past, systems research was focused in major core areas, such as

- programming languages and compilers,
- databases, and
- operating systems.

Two things have happened to transform systems research. New areas have emerged, such as embedded, pervasive and ubiquitous computing. But as well, traditional areas have expanded: data mining has augmented database research; networking and distributed computing have expanded operating systems research. We argue that the traditional model of programming languages and compilers is also expanding into a new model, which we call software automation. As with other areas of systems, this area is more closely aligned with applications than the traditional base discipline.

1.2 High-productivity computing

There is a welcome shift in emphasis from a narrow focus on high-performance computing to the need for *high productivity* computing environments. In some contexts, this means parallel computing [45] involving the continuing evolution of computer architectures. This is still the dominant issue for problems where the scale of the computation is the limiting factor. But high productivity also means that software systems allow rapid advances in areas where new natural phenomena are incorporated in models. Rapid prototyping systems are often used in such a context, but these are typically low-performance systems (such as Matlab). There is a need to have systems that allow both rapid development of new models as well as rapid deployment on novel architectures. We indicate how software automation allows these twin objectives to be approached simultaneously.

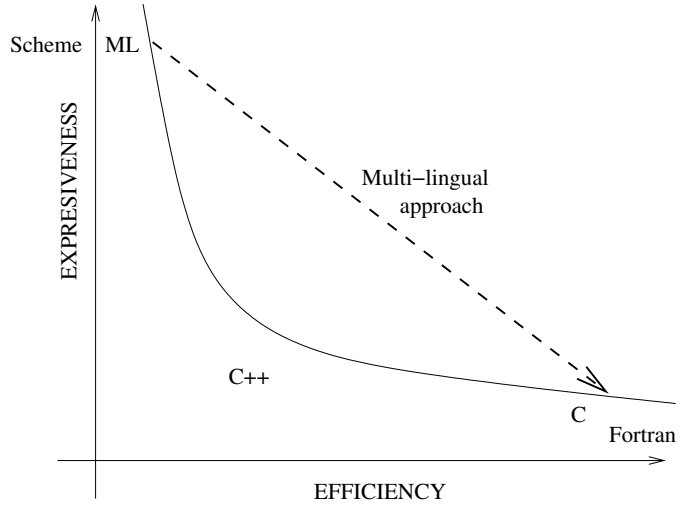

Figure 2: The multi-lingual approach to combine expressiveness and efficiency [2].

2 Efficiency–expressiveness trade-off

There is a fundamental tension between expressiveness and efficiency. Ideally the naive user could code in a high-level specification language and have it translated into efficient machine code. So far, this remains only a dream in most areas of software development. The basic problem with a single-language approach is illustrated in Figure 1.

Some causes of the efficiency–expressiveness conflict can be identified. Typical compilers perform transformations, not optimization. Optimization at compile time, even if based on realistic performance models, would be wildly expensive, so only heuristics are applied in practice. Actual performance is often data dependent. True optimization would have to involve evaluating performance, and this has been done in certain contexts [13, 15].

Many problems have a hierarchical description, and transformations at one level of description may not be appropriate for another. The full efficiency is only obtained by performing the best transformations appropriate for each level. Current language support for hierarchical abstraction is limited. You can define the abstractions that you need, but you cannot specify how to compile it.

Many large codes are themselves multidisciplinary, and the best transformations are often domain dependent. Compiler optimizations cannot be chosen to fit the different parts of an application. One must live with what the compiler writer gives you.

2.1 Multi-lingual approaches

One solution that is often used in practice is multi-lingual. Different parts of the system are developed in appropriate languages, and they are merged in an *ad hoc* fashion. In a high-level (e.g., functional) language, new functionality can be introduced that allows efficient computation at a low level, as indicated in Figure 2.

One common example of this is the efficiency gain when one 'vectorizes' Matlab code. Analysa [2] combined efficiency and expressiveness by using a functional programming lan-

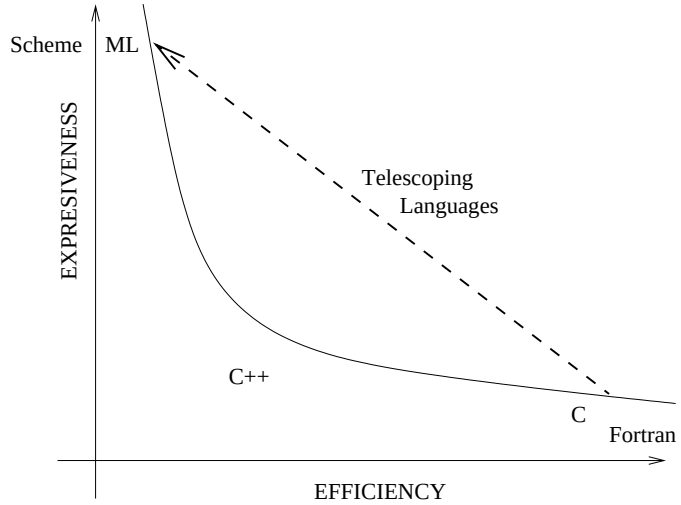

Figure 3: The telescoping language approach to combine expressiveness and efficiency [22].

guage (AlScheme, an extension of Scheme providing among other things efficient linear algebra) as a scripting language which linked with C, C++ and Fortran code (for efficiency).

There are unfortunately serious limits to linking multiple languages informally. At the moment, it lacks formal description, requires development (and maintenance) of ad hoc interface. Different memory models (allocation, garbage collection) can interact adversely at run time. Our experience gained from the development of Analysa [2] shows that this can lead to difficult debugging problems.

2.2 Telescoping languages

Ken Kennedy proposed the use of 'telescoping languages' [22] in which fully optimized low-level code would be included as high-level operations in an extended language. This may be viewed as the reverse direction to what is done with the multi-lingual approach, as depicted in Figure 3.

The Broadway compiler [18] supports domain-specific compiler optimizations. It provides compiler support for a wide range of domains and in the context of existing programming languages using a technique called 'library-level optimization' which recognizes and exploits the domain-specific semantics of software libraries.

3 Hierarchy in software development

Some problems possess a natural hierarchical structure that allows a segmentation of the code generation process. For example, we might have the following levels.

- Model description [application domain]
- Algorithm discovery [mathematical description]
- Algorithm implementation [programming language]

- Executable code [machine code: parallel, multicore]

In the typical programming language approach, the first three levels are all expressed in a single language. The compiler makes the transition from the top levels to the bottom level. However, there is potential for automation to be performed at each level. Interaction between levels can also be tested to optimize performance.

As an example of hierarchy, we consider the problem of signal processing and the solution provided by the Spiral project [41]. The hierarchical levels in this case are as follows.

- Model description [Discrete Fourier Transform: DFT]
- Algorithm discovery [Cooley-Tukey FFT]
- Algorithm implementation [FFTW]
- Executable code [BLAS, ATLAS, OSKI]

Automation has been performed at each level. For example, the Cooley-Tukey FFT algorithm can be derived automatically from an abstract definition of of the discrete Fourier transform (DFT) and its mathematical properties. But optimizations can be carried out at each level. There is no need to hand-code for specific architectures

Hierarchy is also common in other areas of scientific computing. Models are often built from simpler models:

$$\text{Laplace} \implies \text{Stokes} \implies \text{Navier-Stokes} \implies \text{Grade-2 fluids}$$

Moreover, there are natural abstractions to describe simulations of natural phenomena. Equations are the language of science and engineering:

$$E = mc^2$$

$$F = ma$$

$$-\nu\,\Delta\,\mathbf{u} + \mathbf{curl}(\mathbf{u} - \alpha\,\Delta\,\mathbf{u}) \times \mathbf{u} + \nabla\,p = \mathbf{f}$$

The natural language of an application area can be exploited both to automate the generation of code and to provide an environment to carry out optimizations.

Not all applications have simple hierarchical abstractions. In general, more complex relations will occur. We explore one of these in Section 4.

4 Solving PDE's: the FEM

Optimization of code for solving differential equations has been studied widely [31, 32, 47, 48]. Many of these approaches have been based on the finite element method (FEM). There are four distinct areas of finite element codes: function spaces, domain geometry/mesh, differential equation, and equation solution. These are not hierarchical, but rather interact in a more complex way. Each area has it own natural language and its own optimizations. But they also interact in ways that may require a type of inter-procedural analysis to obtain ideal performance.

4.1 FIAT and SyFi

The automation of code related to function spaces can be done in different ways. The principal task is to generate integrals of combinations of derivatives of finite element basis functions. There is often the need to interact with both the mesh and the PDE description.

The evaluation of finite element basis functions, and related inner-product data has been automated by FIAT [24, 25] and by SyFi [38].

4.2 Finite element form compilers

One common representation for PDE's is the language of variations forms. The variational forms must be compiled into code that is compatible with both the function space and the mesh.

The FEniCS Form Compiler (FEC) was introduced in [27] and further developed in [28]. Analysa [2] also included a form compiler.

4.3 FErari

In [26], efficient evaluation of finite element matrices was addressed. This paper posed a complex optimization problem that was further studied in [29]. In [30], a different type of optimization, based on the geometric properties of certain tensors, was explored.

The paper [23] examines the effect of using complexity-reducing relations to generate optimized code for the evaluation of finite element variational forms. The optimizations are implemented in a prototype code named FErari. The authors demonstrate that by invoking FErari as an optimizing backend to the form compiler FFC, they obtain reduced local operation counts by as much as a factor 7.9 and speedups for the assembly of the global sparse matrix by as much as a factor 2.8.

4.4 Mesh generation

All discretization methods (finite element, finite difference, finite volume, spectral element, boundary element) require a mesh. This is an industry unto itself, so we do not try to survey it extensively, but give a few examples.

Triangle [46] is a program for efficiently generating 2D triangulations and Voronoi diagrams. Features include user-specified constraints on angles and triangle areas, user-specified holes and concavities, and the economical use of exact arithmetic to improve robustness. The paper [46] discusses many of the key implementation decisions, including the choice of triangulation algorithms and datastructures, the steps taken to create and refine a mesh, a number of issuesthat arise in Ruppert's algorithm, and the use of exact arithmetic.

NETGEN [44] is an advancing front 2D/3D-Mesh generator based on abstract rules. The process of tetrahedral mesh generation is broken up into four steps: special point calculation, edge following, surface meshing, and volume mesh generation. Several techniques of mesh optimization are tested for quality.

The Bank-Holst adaptive meshing paradigm [3] is an efficient approach for parallel adaptive meshing of elliptic partial differential equations. It is designed to keep communication

costs low and to take advantage of existing sequential adaptive software.

The 'isosurface stuffing' algorithm [35] fills an isosurface with a tetrahedral mesh whose dihedral angles are bounded. It is fast and robust as it generates tetrahedra from a small number of precomputed stencils It is the first algorithm that rigorously guarantees the suitability of tetrahedra for finite element methods in domains whose shapes are substantially more challenging than boxes. If the isosurface is a smooth 2-manifold with bounded curvature, and the tetrahedra are sufficiently small, then the boundary of the mesh is guaranteed to be a geometrically and topologically accurate approximation of the isosurface.

One issue is to be able to partition a given mesh for parallel calculation The paper [39] describes an efficient approach to partitioning unstructured meshes that occur naturally in the finite element and finite difference methods. The approach makes use of the underlying geometric structure of a given mesh and finds a provably good partition in random O(n) time. It applies to meshes in both two and three dimensions. The new method has applications in efficient sequential and parallel algorithms for large-scale problems in scientific computing. This is an overview paper written with emphasis on the algorithmic aspects of the approach. Many detailed proofs can be found in companion papers.

Another issue is to maintain shape regularity as meshes are subdivided. The papers [40, 49] present efficient techniques for constructing simplicial meshes in which each simplex is small enough, according to an application specific error test, and the number of distinctly-shaped simplices in the mesh is small, depending only on the dimension of the problem. The techniques include the decomposition of simplices of a certain kind into smaller similar simplices, the refinement of a hierarchy of simplices to enforce an element size continuity condition, and the processing of such a refined hierarchy to produce a simplicial, tree-structured mesh.

5 Other scientific computing areas

Due to the mathematical nature of scientific simulation and analysis, it is natural that software automation can be applied pervasively. In section 3, we mentioned automation of signal processing code. We briefly mention other areas where automation has already reached a high level of development.

5.1 Quantum chemistry

Quantum chemistry has made major advances recently through automation. This has allowed more complex theories to be applied and has also provided a way to optimize code in novel ways. See [1, 4, 5, 42]. Also [6, 10, 11, 12, 16, 19, 34, 36, 37, 43]

We give some numbers to indicate the scale at which automation operates in the NWChem code. Table 1 lists some indicators of the automatic code generation for computation of dipole polarizability tensor single element evaluation. There are six terms per tensor evaluation, but only two unique pieces of code to compute them. Thus, the data is listed as m+n for each of the two pieces.

The 'coupled-cluster order' n is a measure of the complexity (and accuracy) of the theory being used. The number of diagrams indicates the level of complexity of the iteractions.

Table 1: Some indicators of the size of the automatically generated code for NWChem computation of dipole polarizability tensor single element evaluation. Note, there are six terms per tensor evaluation, but only two unique pieces of code. Thus, the m+n is for each of the two pieces.

Coupled-cluster order	number of diagrams	subroutines	F77 lines of code
2	12+80	29+269	3395+34302
3	21+266	42+807	6113+122463
4	23+560	40+1529	7053+313116

The number of indices in the largest tensors in the interaction terms in the computation is $2n$. Thus for $n = 4$ there are tensors of order eight, i.e., arrays with eight indices. Tensor contraction is a major part of the computation, and the automatically generated code optimizes these contractions.

5.2 Dense linear algebra

Dense linear algebra has also made major advances in terms of automation of code generation and optimization [7, 8, 9].

6 Program analysis and transformation

There are automatic tools that extract information from existing codes. Two areas are performance analysis and sensitivity analysis (a.k.a., differentiation).

6.1 Automating performance analysis

The paper [20] presents a framework for parallel performance data mining and knowledge discovery. The PerfExplorer framework is part of the authors' ongoing research into automatic parallel performance analysis. PerfExplorer adresses the need to manage large-scale data complexity using techniques such as clustering and dimensionality reduction, and the need to perform automated discovery of relevant data relationships using comparitive and correlation analysis techniques. The intended uses of the framework include, but are not limited to, benchmarking, procurement evaluation, modeling, prediction and application optimization.

6.2 Automatic Differentiation

Algorithmic, or automatic, differentiation (AD) is concerned with the accurate and efficient evaluation of derivatives for functions defined by computer programs [17]. No truncation errors are incurred, and the resulting numerical derivative values can be used for all scientific computations that are based on linear, quadratic, or even higher order approximations to nonlinear scalar or vector functions. In particular, AD has been applied to optimization,

parameter identification, equation solving, the numerical integration of differential equations, and combinations thereof. Apart from quantifying sensitivities numerically, AD techniques can also provide structural information, e.g., sparsity pattern and generic rank of Jacobian matrices.

7 Software automation in other domains

Software automation is by no means limited to scientific computing. It can be applied in any area in which there is a suitable abstraction.

7.1 Compilers

Compilers can be defined formally since they translate one language into another in a well defined way. This formalism provides a natural abstraction for automation. Not surprisingly, automatic generation of code has been performed in compiler design [14, 21].

7.2 Algebra of compiler optimization

In [33], the authors (according to their abstract) "use Kleene algebra with tests to verify a wide assortment of common compiler optimizations, including dead code elimination, common subexpression elimination, copy propagation, loop hoisting, induction variable elimination, instruction scheduling, algebraic simplification, loop unrolling, elimination of redundant instructions, array bounds check elimination, and introduction of sentinels. In each of these cases, we give a formal equational proof of the correctness of the optimizing transformation."

References

[1] Auer, A. A., Baumgartner, G., Bernholdt, D. E., Bibireata, A., Choppella, V., Cociorva, D., Gao, X., Harrison, R., Krishnamoorthy, S., Krishnan, S., Lam, C.-C., Lu, Q., Nooijen, M., Pitzer, R., Ramanujam, J., Sadayappan, P., and Sibiryakov, A. Automatic code generation for many-body electronic structure methods: the tensor contraction engine. *Molecular Physics 104* (2006), 211–228.

[2] Bagheri, B., and Scott, L. R. About Analysa. Research Report UC/CS TR-2004-09, Dept. Comp. Sci., Univ. Chicago, 2004.

[3] Bank, R. E. Some variants of the bank–holst parallel adaptive meshing paradigm. *Comput. Vis. Sci. 9*, 3 (2006), 133–144.

[4] Baumgartner, G., Auer, A., Bernholdt, D. E., Bibireata, A., Choppella, V., Cociorva, D., Gao, X., Harrison, R. J., Hirata, S., Krishanmoorthy, S., Krishnan, S., Lam, C.-C., Lu, Q., Nooijen, M., Pitzer, R. M., Ramanujam, J., Sadayappan, P., and Sibiryakov, A. Synthesis of high-performance parallel

programs for a class of ab initio quantum chemistry models. *Proceedings of the IEEE 93*, 2 (2005). special issue on "Program Generation, Optimization, and Adaptation".

[5] Bernholdt, D., Nieplocha, J., and Sadayappan, P. Raising the Level of Programming Abstraction in Scalable Programming Models. *IEEE International Conference on High Performance Computer Architecture (HPCA), Workshop on Productivity and Performance in High-End Computing (P-PHEC)*, 76–84.

[6] Bibireata, A., Krishnan, S., Baumgartner, G., Cociorva, D., Lam, C., Sadayappan, P., Ramanujam, J., Bernholdt, D., and Choppella, V. Memory-constrained data locality optimization for tensor contractions. *Lecture notes in computer science Volume 2958: Languages and Compilers for Parallel Computing*, 93–108.

[7] Bientinesi, P., Dhillon, I. S., and van de Geijn, R. A. A parallel eigensolver for dense symmetric matrices based on multiple relatively robust representations. *SIAM J. Sci. Comput. 27*, 1 (2005), 43–66.

[8] Bientinesi, P., Gunnels, J. A., Myers, M. E., Quintana-Ortí, E. S., and van de Geijn, R. A. The science of deriving dense linear algebra algorithms. *ACM Trans. Math. Softw. 31*, 1 (2005), 1–26.

[9] Bientinesi, P., Quintana-Ortí, E. S., and van de Geijn, R. A. Representing linear algebra algorithms in code: the flame application program interfaces. *ACM Trans. Math. Softw. 31*, 1 (2005), 27–59.

[10] Cociorva, D., Baumgartner, G., Lam, C.-C., Sadayappan, P., Ramanujam, J., Nooijen, M., Bernholdt, D. E., and Harrison, R. Space-time trade-off optimization for a class of electronic structure calculations. In *PLDI '02: Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation* (New York, NY, USA, 2002), ACM Press, pp. 177–186.

[11] Cociorva, D., Wilkins, J., Baumgartner, G., Sadayappan, P., Ramanujam, J., Nooijen, M., Bernholdt, D., and Harrison, R. Towards automatic synthesis of high-performance codes for electronic structure calculations: Data locality optimization. *Lecture Notes in Computer Science 2228* (2001), 237–??

[12] Cociorva, D., Wilkins, J. W., Lam, C.-C., Baumgartner, G., Ramanujam, J., and Sadayappan, P. Loop optimization for a class of memory-constrained computations. In *ICS '01: Proceedings of the 15th international conference on Supercomputing* (New York, NY, USA, 2001), ACM Press, pp. 103–113.

[13] Demmel, J., Dongarra, J., Eijkhout, V., Fuentes, E., Petitet, A., Vuduc, R., Whaley, C., and Yelick, K. Self adapting linear algebra algorithms and software. *Proceedings of the IEEE 93*, 2 (2005). special issue on "Program Generation, Optimization, and Adaptation".

[14] Fraser, C. W. *Automatic generation of code generators.* PhD thesis, 1977.

[15] Frigo, M., and Johnson, S. G. The design and implementation of FFTW3. *Proceedings of the IEEE 93*, 2 (2005). special issue on "Program Generation, Optimization, and Adaptation".

[16] Gao, X., Sahoo, S. K., Lam, C.-C., Ramanujam, J., Lu, Q., Baumgartner, G., and Sadayappan, P. Performance modeling and optimization of parallel out-of-core tensor contractions. In *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming* (New York, NY, USA, 2005), ACM Press, pp. 266–276.

[17] Griewank, A. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[18] Guyer, S. Z., and Lin, C. Broadway: a compiler for exploiting the domain-specific semantics of software libraries. *Proceedings of the IEEE 93* (2005), 342–357.

[19] Hirata, S. Tensor Contraction Engine: Abstraction and automated parallel implementation of configuration-interaction, coupled-cluster, and many-body perturbation theories. *J. Phys. Chem. A 107*, 46 (2003), 9887–9897.

[20] Huck, K. A., and Malony, A. D. Perfexplorer: A performance data mining framework for large-scale parallel computing. *sc 0* (2005), 41.

[21] Karuri, K. *A Framework for Automatic Generation of Code Optimizers*. PhD thesis, 2001.

[22] Kennedy, K., Broom, B., Chauhan, A., Fowler, R., Garvin, J., Koelbel, C., McCosh, C., and Mellor-Crummey, J. Telescoping languages: A system for automatic generation of domain languages. *Proceedings of the IEEE 93*, 2 (2005). special issue on "Program Generation, Optimization, and Adaptation".

[23] Kirby, R., and Logg, A. Benchmarking domain-specific compiler optimizations for variational forms. Preprint from The Finite Element Center, April 2007.

[24] Kirby, R. C. Algorithm 839: Fiat, a new paradigm for computing finite element basis functions. *ACM Trans. Math. Softw. 30*, 4 (2004), 502–516.

[25] Kirby, R. C. Optimizing fiat with level 3 blas. *ACM Trans. Math. Softw. 32*, 2 (2006), 223–235.

[26] Kirby, R. C., Knepley, M., Logg, A., and Scott, L. R. Optimizing the evaluation of finite element matrices. *SIAM J. Sci. Computing 27* (2005), 741–758.

[27] Kirby, R. C., and Logg, A. A compiler for variational forms. *ACM Trans. Math. Softw. 32*, 3 (2006), 417–444.

[28] Kirby, R. C., and Logg, A. Efficient compilation of a class of variational forms. *ACM Trans. Math. Softw. 33*, 3 (2007), 17.

[29] Kirby, R. C., Logg, A., Scott, L. R., and Terrel, A. R. Topological optimization of the evaluation of finite element matrices. *SIAM J. Sci. Computing 28* (2006), 224–240.

[30] Kirby, R. C., and Scott, L. R. Geometric optimization of the evaluation of finite element matrices. *SIAM J. Sci. Computing 29* (2007), 827–841.

[31] Korelc, J. Automatic generation of finite-element code by simultaneous optimization of expressions. *Theoretical Computer Science 187* (Nov 1997), 231–248.

[32] Korelc, J. Multi-language and multi-environment generation of nonlinear finite element codes. *Engineering with Computers 18* (Nov 2002), 312–327. 10.1007/s003660200028.

[33] Kozen, D., and Patron, M.-C. Certification of compiler optimizations using kleene algebra with tests. In *CL '00: Proceedings of the First International Conference on Computational Logic* (London, UK, 2000), Springer-Verlag, pp. 568–582.

[34] Krishnan, S., Krishnamoorthy, S., Baumgartner, G., Cociorva, D., Lam, C., Sadayappan, P., Ramanujam, J., Bernholdt, D., and Choppella, V. Data Locality Optimization for Synthesis of Efficient Out-of-Core Algorithms. *Proc. of the Intl. Conf. on High Performance Computing* (2003).

[35] Labelle, F., and Shewchuk, J. R. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph. 26*, 3 (2007), 57.

[36] Lam, C.-C., Cociorva, D., Baumgartner, G., and Sadayappan, P. Memory-optimal evaluation of expression trees involving large objects. In *HiPC* (1999), pp. 103–110.

[37] Lam, C.-C., Sadayappan, P., and Wenger, R. On optimizing a class of multi-dimensional loops with reductions for parallel execution. *Parallel Processing Letters 7*, 2 (1997), 157–168.

[38] Mardal, K. A., Skavhaug, O., Lines, G., Staff, G., and Odegard, A. Using Python to solve partial differential equations. *Computing in Science and Engineering* (2007).

[39] Miller, G. L., Teng, S., Thurston, W., and Vavasis, S. A. Automatic mesh partitioning. Tech. rep., Ithaca, NY, USA, 1992.

[40] Moore, D., and Warren, J. Adaptive simplicial mesh quadtrees. *Houston J. Math 21*, 3 (1995), 525–540.

[41] Püschel, M., Moura, J. M. F., Johnson, J., Padua, D., Veloso, M., Singer, B. W., Xiong, J., Franchetti, F., Gačić, A., Voronenko, Y., Chen, K., Johnson, R. W., and Rizzolo, N. SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE 93*, 2 (2005). special issue on "Program Generation, Optimization, and Adaptation".

[42] Sahoo, S. K., Krishnamoorthy, S., Panuganti, R., and Sadayappan, P. Integrated loop optimizations for data locality enhancement of tensor contraction expressions. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference* (2005), pp. 13–13.

[43] Sahoo, S. K., Krishnamoorthy, S., Panuganti, R., and Sadayappan, P. Integrated loop optimizations for data locality enhancement of tensor contraction expressions. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing* (Washington, DC, USA, 2005), IEEE Computer Society, p. 13.

[44] Schöberl, J. Netgen: An advancing front 2d/3d-mesh generator based on abstract rules. *Computing and Visualization in Science* (1997).

[45] Scott, L. R., Clark, T. W., and Bagheri, B. *Scientific Parlallel Computing.* Princeton University Press, 2005.

[46] Shewchuk, J. R. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineeri= ng*, M. C. Lin and D. Manocha, Eds., vol. 1148 of *Lecture Notes in Computer Science.* Springer-Verlag, May 1996, pp. 203–222. From the First ACM Workshop on Applied Computational Geometry.

[47] van Engelen, R., Wolters, L., and Cats, G. CTADEL: a generator of multi-platform high performance codes for PDE-based scientific applications. In *ICS '96: Proceedings of the 10th international conference on Supercomputing* (New York, NY, USA, 1996), ACM Press, pp. 86–93.

[48] Wang, P. S., Tan, H.-Q., Saleeb, A. F., and Chang, T.-Y. P. Code generation for hybrid mixed mode formulation in finite element analysis. In *SYMSAC '86: Proceedings of the fifth ACM symposium on Symbolic and algebraic computation* (New York, NY, USA, 1986), ACM Press, pp. 45–52.

[49] Zhang, S. Successive subdivisions of tetrahedra and multigrid methods on tetrahedralmeshes. *Houston J. Math 21*, 3 (1995), 541–556.