AFRL-AFOSR-VA-TR-2018-0346

A Deep-Learning Approach towards Auto-Tuning CFD Codes

Wu-Chun Feng
VIRGINIA POLYTECHNIC INST AND STATE UNIVERSITY

09/18/2018
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA2
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
03-07-2018	Final	February 2017 to February 2018

4. TITLE AND SUBTITLE

A Deep-Learning Approach towards Auto-Tuning CFD Codes

5a. CONTRACT NUMBER

FA9550-17-1-0205

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Feng, Wu-Chun

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

VIRGINIA POLYTECHNIC INSTITUTE & STATE UNIVERSITY (VIRGINIA TECH)
1880 PRATT DR STE 2006
BLACKSBURG VA 24060-3580

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

USAF, AFRL DUNS 143574726
AF OFFICE OF SCIENTIFIC RESEARCH
875 N. RANDOLPH ST. ARLINGTON VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Heterogeneous computing systems are increasingly becoming the norm in high-performance computing (HPC). For the June 2018 TOP500 List, the majority of computational power on the TOP500 comes from systems containing heterogeneous computing devices, e.g., CPUs, GPUs, APUs, and Xeon Phis. However, significant hurdles impede a domain scientists' ability to extract high performance out of such heterogeneous devices, including (1) selecting appropriate algorithm(s) for the target heterogeneous device, (2) setting runtime parameters, and (3) configuring hardware relative to some evaluation metric, e.g., performance, power, or energy efficiency. Furthermore, given the diversity of HPC systems, domain scientists want their software codes to be portable across many computing systems and to understandably have some measure of "future-proofing" of their software codes, even as the underlying hardware continues to rapidly evolve. Thus, our project studies, analyzes, and synthesizes machine-learning and deep-learning approaches that expose the parameters as "knobs" and tune them dynamically during the simulation so as to optimize for the metric of interest, whether it be performance, power, energy efficiency, numerical accuracy, or fidelity of the flow physics.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Wu-chun Feng
U	U	U	UU		19b. TELEPHONE NUMBER *(Include area code)* +1-540-231-1192

Reset

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE. Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE. State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATES COVERED. Indicate the time during which the work was performed and the report was written, e.g., Jun 1997 - Jun 1998; 1-10 Jun 1996; May - Nov 1998; Nov 1998.

4. TITLE. Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER. Enter all contract numbers as they appear in the report, e.g. F33615-86-C-5169.

5b. GRANT NUMBER. Enter all grant numbers as they appear in the report, e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER. Enter all program element numbers as they appear in the report, e.g. 61101A.

5d. PROJECT NUMBER. Enter all project numbers as they appear in the report, e.g. 1F665702D1257; ILIR.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER. Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER. Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/ monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.

A Deep-Learning Approach Towards Auto-Tuning CFD Codes

PI: W. Feng, wfeng@vt.edu, 540-231-1192
Co-PIs: Eric de Sturler, Christopher Roy, Danesh Tafti (Virginia Tech)
Jack Edwards, Hong Luo, Frank Mueller (North Carolina State University)

Reporting Period: 02/15/2017 – 02/14/2018

Heterogeneous computing systems are increasingly becoming the norm in high-performance computing (HPC). For the June 2018 TOP500 List, for example, more than 50% of the computational power on the TOP500 comes from systems containing heterogeneous computing devices, e.g., CPUs, GPUs, APUs, and Xeon Phis. However, significant hurdles impede a domain scientists' ability to extract high performance out of such heterogeneous devices, including (1) selecting appropriate algorithm(s) for the target heterogeneous device, (2) setting runtime parameters, and (3) configuring hardware relative to some evaluation metric, e.g., performance, power, or energy efficiency. Due to this complexity, our research for this one-year project has focused on applying machine learning or deep learning to auto-select or auto-configure algorithms for the underlying hardware. Furthermore, given the diversity of HPC systems, ranging from the aforementioned heterogeneous computing systems to the more homogeneous ones that make up the rest of the supercomputers on the TOP500 list, domain scientists want their software codes to be portable across many computing systems and to understandably have some measure of "future-proofing" of their software codes, even as the underlying hardware continues to rapidly evolve. For example, advanced simulation codes, like computational fluid dynamics (CFD), come with their own complex choices, such as time-stepping algorithms, linear and nonlinear solvers, choices in physics models and discretization, and so on. Hence, the total number of parameters and possible combinations of parameter choices that determine the performance of a simulation is daunting. Specifically, exploring hardware, software, and algorithmic design choices often requires time-consuming simulations and while some brute-force auto-tuning support has been proposed in the past, the results are heuristic and often narrow in scope, i.e., only applicable to a particular problem. Consequently, our one-year project proposed to study, analyze, and synthesize machine-learning approaches (and where appropriate, deep-learning approaches as well) that expose the various parameters as "knobs" and tune them dynamically during the simulation so as to optimize for the metric of interest, whether it be performance, power, energy efficiency, numerical accuracy, or fidelity of the flow physics.

1. Research and Educational Activities

This section outlines the research and educational activities that were facilitated by this grant award on deep learning as applied to computational fluid dynamics (CFD).

1.1 Overall Goal

Rapid changes just between accelerator generations further impose massive manual efforts on tuning and sometimes even algorithmic redesign. Our hypothesis is that a combination of cutting-edge design-, compile-, and run-time technologies can mitigate these problems but only if the application domain is limited, e.g., computational fluid dynamics (CFD).

1.2 General Research Objectives

The objective of this work is to develop a software infrastructure specifically targeted at machine-learning (ML), and where appropriate, the more narrow area of deep learning (DL), for the auto-tuning computational fluid dynamics (CFD) codes that lifts the burden of retargeting simulation codes to constantly changing and increasingly specialized accelerator platforms.

To this end, the following tasks have been accomplished:

1. Generalized fine-grained auto-tuning for CFD codes, including
    a. a semi-automated approach to intelligently auto-tune scientific codes via machine learning (ML) or deep learning (DL);
    b. a fine-grained auto-tuning framework for scientific codes that allows different optimizations to be applied;
    c. an evaluation of the auto-tuning framework with Intel's compiler and multiple platforms; and
    d. results indicating that fine-grained auto-tuning via machine learning (ML) or deep learning (DL) outperforms traditional coarse-grained approaches significantly

2. Application-specific CPU/GPU parallelism strategies, including
    a. auto-tuning INFLOW3D by estimating errors associated with models for subgrid-scale chemical production in turbulent combustion;
    b. auto-tuning RDGFLO3D CFD code by
        i. identifying parameters of the unstructured-mesh based RDGFLO3D code for auto-tuning;
        ii. assessing machine-leaning (ML) capabilities for automatically exploring the tuning space; and
        iii. We determined their effectiveness in terms of performance improvements
    c. auto-tuning GenIDLEST CFD code [Tafti01, Tafti10] by
        i. identifying parameters of the GenIDLEST incompressible flow solver code for auto-tuning, including orthogonality of the fluid grid, total number of grid blocks comprising the global domain, convergence threshold of

the linear solvers, number of computational nodes working on the problem, computational cores per node, L3 cache size, and memory bandwidth and

ii. using supervised ML to predict the optimal level of granularity in the substructure, minimizing time to solution for a range of problem types

d. auto-tuning SENSEI CFD code [Derla13] by

i. identifying tuning parameters for the three highest-occupying kernels and

ii. using the AUMA framework [Falch16] for the machine-learning optimization of run-time performance

1.3 Partnership

The project is a multi-institutional collaborative partnership between Virginia Tech and North Carolina State University.

1.4 Research Activities

Activities are reported in several parts:

[1] Generalized fine-grained auto-tuning for CFD codes

a. HPCTuner: We created a fine-grained compilation framework, HPCTuner. HPCTuner compiles hot computation loops, one at a time, with different compilation flag sets. A lightweight profiling phase employs Caliper [Boehm16] for loop runtime measurement. The loops accounting for most significant portion of runtime are identified as hot loops. Each hot loop is subsequently outlined into a new source file, so that they can be compiled independently with different compilation flags (compilation configurations). We select 32 optimization flags to construct an extremely large compilation configuration space (CCS) for 2^{32} flag combinations, where each flag may have more than two valid values (typically on/off). We then sample 1000 compilation configurations out of CCS to generate 1000 code variants for the entire program and collect fine-grained per-loop runtime information for each variant. The next critical step is to utilize per-loop runtime information to compose a faster binary, where per-loop compilation configurations are selected automatically.

b. iML: We created a generalized iterative machine-learning approach, internally dubbed iML, that performed fine-grained auto-tuning of the threadblock size (in only two dimensions) on a canonical lid-driven cavity CFD code to deliver as much as a 20% improvement without changing the lid-driven cavity CFD code. We tested our iterative approach across a myriad of machine-learning (ML) algorithms, e.g., random forest, classification

and regression trees, k-nearest neighbors, multi-layer perceptron, and support vector machine. Future work includes (1) creating a general iterative framework for machine learning, (2) improving the performance of our iterative approach, specifically reducing the number of samples needed to learn while improving the performance to be deployed at run time, (3) increasing the volume and variety of tunable parameters beyond the thread-block size, and (4) expanding the volume and variety of applications with which to test the efficacy of our approach.

[2] Application-specific CPU/GPU parallelism strategies

a. Auto-tuning INCOMP3D: One component of our original AFOSR Basic Research Initiative (BRI) effort resulted in the development of an implicit flow solver, known as INCOMP3D, suitable for conducting large-scale, large-eddy simulations (LES) of incompressible flows on structured, simply-connected, multi-block meshes. Hybrid CPU/GPU parallelization strategies were developed, as were means of efficiently performing block incomplete LU decompositions using a combination of coarse- and fine-grained parallelism using CUDA and OpenACC paradigms. While it is possible to use machine learning and deep data-mining to optimize the performance of this code via alterations in the algorithmic parameters, we instead focused on leveraging CPU/GPU fine-grained parallelism strategies developed in the earlier work to tackle a formidable problem in LES modeling estimation of errors associated with models for subgrid-scale chemical production in turbulent combustion. Specifically, we developed a data-mining tool that accepts input from a time-sequenced, multi-resolution large-eddy simulation of a turbulent bluff-body stabilized flame. In this strategy, coupled large-eddy simulations are conducted on a sequence of successively-coarsened meshes. Data from the finest mesh represents the truth model. The velocity field from the finest mesh may be filtered to the coarse-mesh levels and used to form a source term that constrains the coarse-mesh velocity field. This ensures a strong correlation among eddy structures at the different mesh levels. Flow properties are outputted at specific coarse mesh locations, at all neighboring coarse mesh cells, and at all finer-mesh locations that lie within the coarse-mesh neighborhood. This data is then written at different instances in time throughout an iteration sequence for use in the data-mining activity.

b. Auto-tuning RDGFLO3D: The effect of different renumbering methods for elements on the performance of the GMRES+LU-SGS methods in the RDGFLO3D has been investigated. Theoretical results are rare or even do not exist in this context. Some renumbering methods work surprisingly well in practice. Unfortunately, the best methods might be hardware-dependent. Three common renumbering techniques are considered in this work, which are Reverse Cuthill-McKee renumbering, Hyperplane renumbering, and vectoriza-

tion renumbering. The reverse Cuthill-McKee renumbering is widely used to minimize the bandwidth of the resulting matrix in the finite element formulation, which proves to be efficient for the GMRES+ILU method. However, the vectorization of the LU-SGS method is not certain using the reverse Cuthill-McKee renumbering. The objective of the hyperplane renumbering is to provide a balance between lower and upper matrices. Its main advantage is that the LU-SGS method is vectorizable for each hyperplane. The idea of the vectorization renumbering is to minimize the number of groups which can be vectorized, and thus to achieve the maximum efficiency for vectorization.

c. Auto-tuning GenIDLEST: This code is a general-purpose incompressible flow solver code with many physical models for different classes of problems. Common to all the different classes of problems is the solution of linear systems, which consumes 50-90% of the execution time. The linear solutions are carried out by preconditioned Krylov methods. Typically, the conjugate gradient (CG) method is used for symmetric systems and the bi-conjugate gradient–stabilized (BiCGSTAB) method is used for non-symmetric systems. Other Krylov based solvers such as GMRES(M), rBiCGSTAB and rGCROT are also available. The last two recycle the Krylov subspace vectors between time steps [5] when the system matrix remains fixed and only the right-hand side vector changes between time steps. To accelerate convergence, the Richardson method or SSOR is used as an iterative preconditioner. The pre-conditioners are applied to a sub-structured system. The objective is to build a model using supervised machine learning (ML) to predict the optimal level of granularity in the sub-structure, which minimizes time to solution for a range of problem types. Each problem type is defined by seven input features: orthogonality of the fluid grid; total number of grid blocks comprising the global domain; convergence threshold of the linear solvers (typically [1e-5, 1e-8]); number of computational nodes working on the problem; computational cores per node; L3 cache size, and memory bandwidth (typically on the order of 50 to 100 GB/s).

d. Auto-tuning SENSEI CFD code is a multi-block, structured grid, cell-centered, second-order finite volume solver developed using agile programming techniques and written in Fortran 03/08. The code base integrates the work of several projects into a single system for performing error estimation and grid adaptation and for providing an easily extensible common framework for future development. In addition, it utilizes a neew formulation of the method of manufactured solutions ideally suited for solvers, which are based on discretizations of the weak form of the governing equations of interest. To tune the SENSEI code for high performance, we leverage a machine-learning model from the AUMA framework [Falch16] to tune six (6) application-specific parameters in the three (3) most-executed kernels.

1.5 Results from the Reporting Period

In a single year of funding, research activities have produced numerous software artifacts, paper drafts, and publications. The following can be reported:

1. Fine-grained auto-tuning: With the collected per-loop runtime information, we first estimate the best overall runtime by aggregating the minimum runtime of each loop and the "left-over" code (i.e., non-hot code regions). Then, we evaluate the following methods to compose the final executable on several CPU generations of the Intel compiler.
   a. G.realized uses the per-loop best compilation configurations to compile each region.
   b. FR uses different compilation configurations for each region. Each compilation configuration is randomly selected from the 1000 generated compilation configuration. Note that there is no need for per-loop runtime measurement in this approach.
   c. CFR, our method, filters out compilation configurations associated with bad per-loop performance, and then performs FR.
   d. R.origin performs coarse-grained random selection on the origin program, which is the state of the art, i.e., it is agnostic of regions.
   e. O3.caliper compiles the origin program with –O3 and Caliper instrumentation. This allows us to assess the overhead of Caliper.

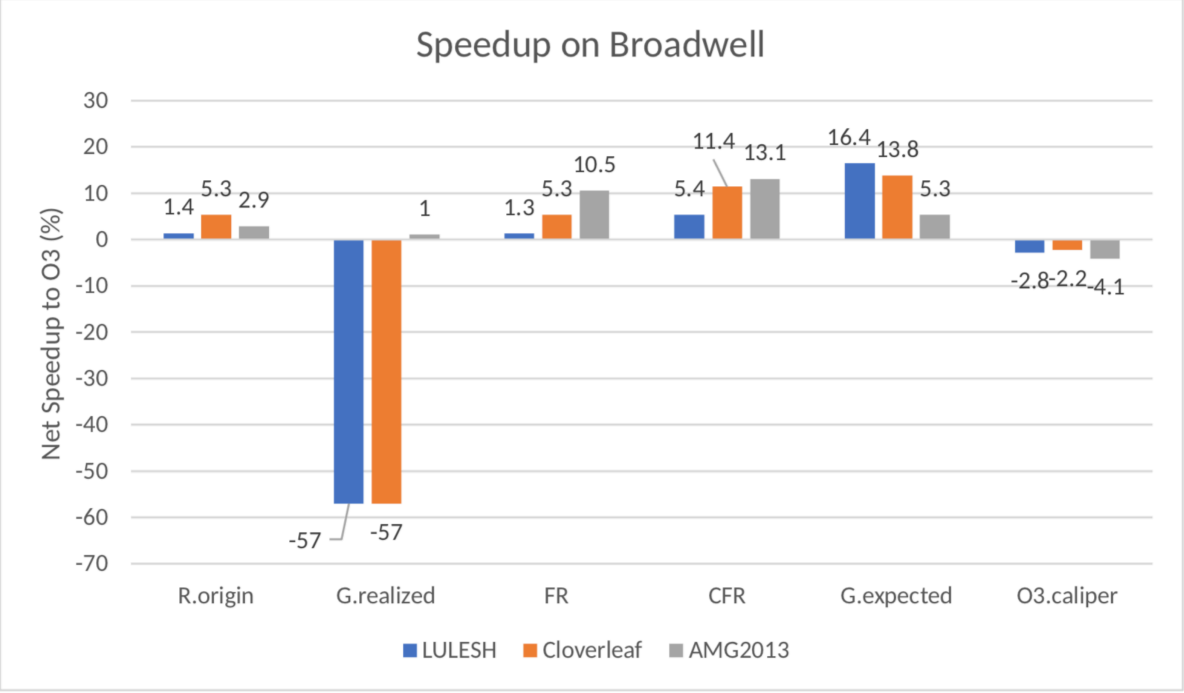

Figure 1: Results for Different Auto-Tuning Flags on Intel Broadwell Architecture

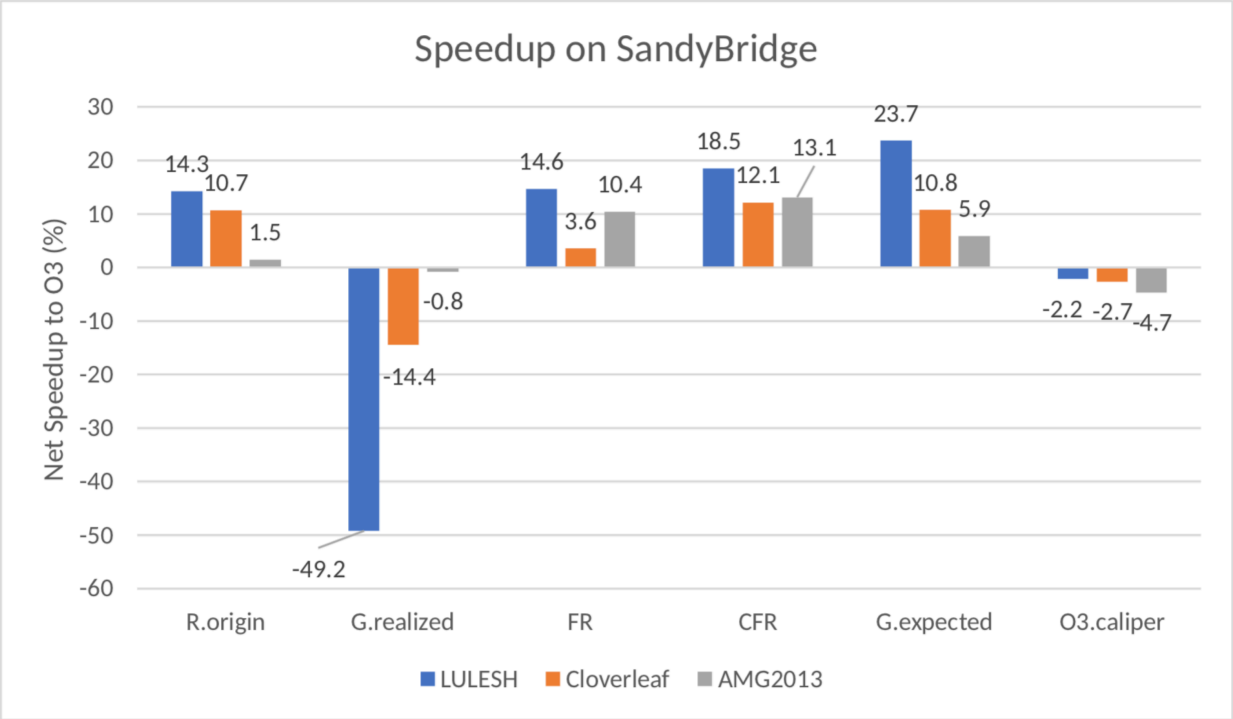

Figure 2: Results for Different Auto-Tuning Flags on Intel Sandy Architecture

Our results in Figures 1 and 2 depict speedup relative to –O3 compilation (the common default) on the y-axis over each optimization scheme, including our CFR auto-tuning, for the benchmarks Lulesh, Cloverleaf, and AMG2013. Each figure depicts a different hardware platform, namely Intel Broadwell and Intel SandyBridge. The results show that G.realized often results in significant slowdowns while the state-of-the-art method R.origin can improve performance on most occasions and our CFR method can improve performance on our selected modern scientific applications with up to 20% improvement. We note that our method is effective and platform-independent and can benefit from machine learning to reduce tuning overhead.

2. Application-specific CPU/GPU parallelism strategies:

   a. INCOMP3D: The data-mining code reads this data and then computes correlations of flow properties and functions of flow properties (such as the chemical species production rates). The fine-mesh properties, filtered to the coarse-mesh levels, represent the truth model to which calculated coarse-mesh results can be compared. If the function represents the outcome of a model for a subgrid process that could be applied at a coarse mesh level, then one can directly calculate the error between the actual result

and the outcome of the model. Considering, for example, an L2 norm of a vector of species production rates as a representative function, one could calculate this function using production-rate data filtered from the finest mesh and production-rate data evaluated using data present at the coarse mesh. The ratio (or difference) of these values represents the degree at which the coarse-mesh function approximates the truth model. Figure 3 shows a representative example. The pink region represents the effect of unresolved fluctuations in inhibiting chemical activity. The black line represents the desired response of a perfect model for the L2 norm, which would collapse the data about this line. Deviations from this line represent model form errors, and considering the range of such errors, it might be concluded that simple models for chemical production may have little chance of predicting the fine-mesh response.

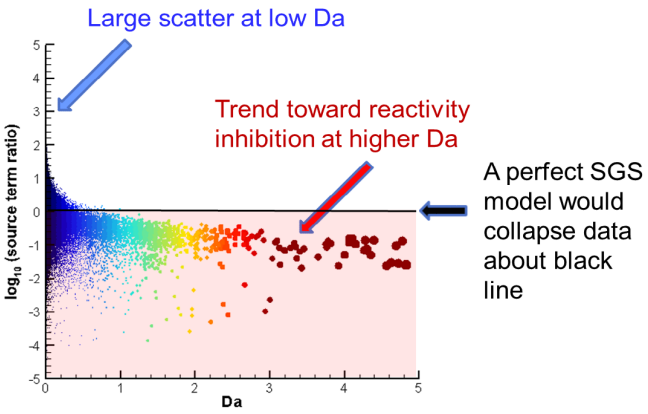

Figure 3: Scatterplot correlation between fine and coarse mesh: reactivity measure

Also, filtered density functions describe the probability of finding a fluid property of a certain value within the sub-filter scales. Such functions have been used to characterize distributions of subgrid quantities, thus enabling certain closure models to be constructed. Most such distributions describe passive scalars — quantities that are invariant during chemical reaction. It is of interest to determine whether FDFs of reactive scalar measures possess characteristic forms and whether they possess any degree of similarity across scales. We have constructed single-point / single-time FDFs of a normalized chemical time scale estimate using fine, medium, and coarse-mesh neighborhood data examples for a few spatio-temporal locations are shown in Figure 4.

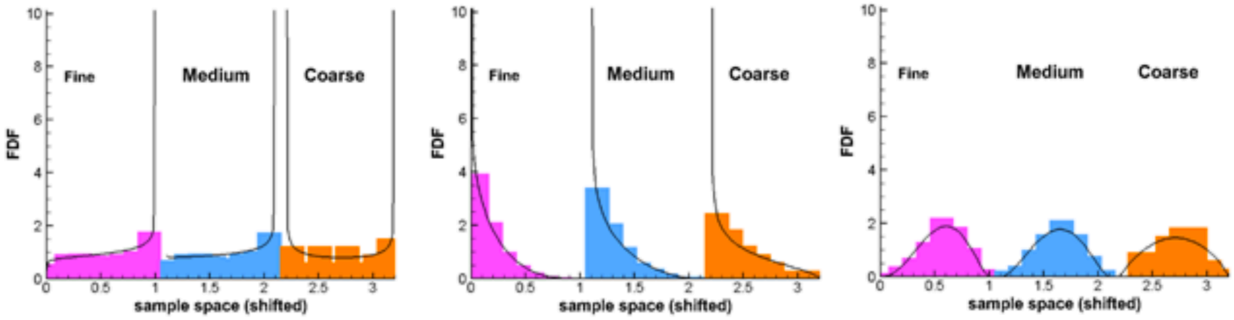

Figure 4: FDFs of normalized chemical time scale (black curve is beta PDF)

Even given the sparsity of the coarse-mesh data neighborhood, strong evidence of scale-similarity can be noted, and a beta PDF form (the solid black lines), constructed using subgrid mean and variance information from the neighborhood, provides a reasonable model for the calculated distributions.

Because of the time spent in obtaining trusted-source data for this effort (the calculation of the bluff-body stabilized flame took two months), we were unable to attempt to map the current data-mining tool to multi-GPU/CPU systems. A promising extension would be to use machine-learning techniques to fit distributions representing the error between model and prediction as a function of non- dimensional ratios of characteristic subgrid time, length, velocity, and chemical rate scales. If this could be done while maintaining sufficient generality, it might be possible to develop a relatively simple closure model for the filtered chemical production rates that would reduce the scatter shown in Figure 4 significantly.

b. Auto-tuning RDGFLO3D: The efficiency of any CFD codes not only depends on the numerical algorithms, but also on the hardware to be used. The renumbering techniques for elements have an effect on the GMRES+LU-SGS method in the RDGFLO3D code. The hyperplane renumbering gives the best performance in computers with GPUs where the vectorization is used. The Reverse Cuthill-McKee method gives the best performance in computers with cache-base CPUs. The machine learning can be used to for automatically exploring the tuning space and achieve the best performance.

c. Auto-tuning GenIDLEST CFD code by using a random forest model to predict the optimal level of granularity in the sub-structure to minimize the time to solution for a range of problem types defined by the following seven input features:

   i. Orthogonality of the fluid grid (binary 0 or 1)
   ii. Total number of grid blocks comprising the global domain (any positive integer)
   iii. Convergence threshold of the linear solvers (typically some value 1e-5 to 1e-8)
   iv. Number of compute nodes working on the problem (integer, typically 1 to 20)
   v. Computational cores per node (integer, typically 2 to 32)
   vi. L3 cache size (typically on the order of 10 to 100 MB)
   vii. Memory bandwidth (typically on the order of 50 to 100 GB/s)

   The initial results are promising and currently being prepared for submission [4].

2. Training

Over the course of this one-year award, ten (10) Ph.D. students, one (1) M.S. student, and one (1) B.S. student were trained in conjunction with this project – six Ph.D. students, one M.S. student, and one B.S. student at Virginia Tech and four Ph.D. students at North Carolina State University.

Skills acquired through the project are in the areas of teamwork, communication, and writing publications. This includes presentation skills (e.g., invited talks, poster presentations at a premier conference, and thesis defenses). Furthermore, a number of independent studies were supervised by the PIs in related areas emphasizing problem solving, design and implementation as well as writing skills.

This project contributes to human resource development by educating students through research, providing new educational materials in the classroom, and giving students the communication and writing skills needed to advance science and engineering for future generations. Hands-on computer systems and new educational materials effectively integrate research and teaching.

3. Publications

[1] T. Wang, D. Beckingsale, N. Kain, T. Gamblin, F. Mueller, "HPCTuner: Auto-tuning Scientific Applications With Per-loop Compilation," PACT'18 (to be submitted).

[2] A. McCall, W. Xue, and C. J. Roy, "Multilevel Parallelism with MPI and OpenACC for Complex CFD Codes," AIAA Paper 2017-3293, 23rd AIAA Computational Fluid Dynamics Conference, Denver, Colorado, 5-9 June, 2017.

[3] W. Xue, C. W. Jackson, and C. J. Roy, "Multi-CPU/GPU Parallelization of Structured Grid CFD Codes using Domain Decomposition," AIAA Paper 2018-0362, AIAA SciTech, January 2018.

[4] A. Moosavi, Z. Cooper, L. He, P. Windes, A. Sandu, and D. Tafti, "Supervised Machine Learning Assisted Preconditioning of Krylov-Based Linear Solvers," Under preparation.

[5] X. Cui and W. Feng, "iML: Iterative Machine Learning for Auto-Tuning Performance on Graphical Processing Units (GPUs)," Under preparation.

4. Changes in Aims of the Project

None.

4.1 Problems/Delays

None.

4.2 Personnel Changes

None.

5. Budget Status

No significant deviation from plan.

References

[Amrit15] Amritkar, A., de Sturler, E., Swirydowicz, K., Tafti, D. K. and Ahuja, K. "Recycling Krylov subspaces for CFD applications," J. Comp. Physics 303, pp. 222-237, December 2015.

[Boehm16] D. Boehme, T. Gamblin, D. Beckingsale, P. T. Bremer, A. Gimenez, M. LeGendre, O. Pearce, and M. Schulz. Caliper: Performance introspection for hpc software stacks. In *Supercomputing*, pages 550–560, Nov 2016.

[Derla13] Derlaga, J. M., Phillips, T. and Roy, C. J. "SENSEI computational fluid dynamics code: a case study in modern Fortran software development," *21st AIAA Computational Fluid Dynamics Conference (AIAA)*, pp. 2013-2450, 2013.

[Falch16] Falch, T. L. and Elster, A. C., "Machine learning-based auto-tuning for enhanced performance portability of OpenCL applications," *Concurrency and Computation: Practice and Experience*, 2016.

[Tafti01] Tafti, D., "GenIDLEST – A Scalable Parallel Computational Tool for Simulating Complex Turbulent Flows, Proceedings of the ASME Fluids Engineering Division, FED -Vol. 256, pp. 347-356, ASME-IMECE, Nov. 2001.

[Tafti10] Tafti, D., "Time-accurate techniques for turbulent heat transfer analysis in complex geometries, Advances in Computational Fluid Dynamics and Heat Transfer; Eds: Ryo Amano & Bengt Sunden, Series: Developments in Heat Transfer, WIT PRESS, Southampton, UK. 2010.