ACCELERATING COMPUTATIONAL FLUID DYNAMICS CODES ON MULTI-/MANY-CORE INTEL PLATFORMS

Gaurav Bansal[1], Anand Deshpande[2], Paul Edwards[1], Alexander Heinecke[2], Michael Klemm[1], Dheevatsa Mudigere[2], Elmoustapha Ould-ahmed-vall[1], Mikhail Smelyanskiy[2], Michael Steyer[1], Nishant Agrawal[3], Ravi Ojha[3], Ambuj Pandey[3], Rihab Abdul Razak[3], Juan J. Alonso[4], Thomas D. Economon[4], Francisco Palacios[4], and David Keyes[5]

[1] Software and Services Group, Intel Corporation
[2]Parallel Computing Lab, Intel Corporation
{gaurav2.bansal, anand.m.deshpande, paul.m.edwards, alexander.heinecke, michael.klemm, dheevatsa.mudigere, elmoustapha.ould-ahmed-vall, mikhail.smelyanskiy, michael.steyer}@intel.com
[3]Innovation Labs, Tata Consultancy Services Ltd.
{nishant.agrawal, ravi.ojha, ambuj.pandey, rihab.abdulrazak}@tcs.com
[4]Stanford University, Stanford, CA,USA.
{jjalonso, economon, fpalacios}@stanford.edu
[5]King Abdullah University of Science and Technology, Thuwal, Saudi Arabia.
{david.keyes}@kaust.edu.sa

Key words: Many-core architectures, fine-scale parallelism, vectorization, finite-volume unstructured mesh CFD

Abstract. In this paper, we present optimization techniques that are crucial to unlock parallelism and vectorization in modern computational fluid dynamics (CFD) codes thereby significantly improving their performance on emerging Intel multi-/many-core platforms such as the Intel® Xeon®[1] processors and Intel® Xeon Phi™[1] coprocessors. We focus on unstructured-mesh finite-volume codes and restrict the discussion to fine-scale optimizations with the objective to improve the strong-scaling behavior of these classes of algorithms. We present the key architectural features of the Intel Xeon Phi coprocessor and describe strategies to exploit them for improving performance of three widely used CFD codes. Our benchmarking results show substantial performance advantages to speed up time-to-solution.

1 INTRODUCTION

Unstructured mesh CFD codes pose inherent challenges for shared-memory multiprocessors, due to their large irregular working sets, unstructured memory accesses and variable but limited amount of parallelism. Effectively utilizing compute and memory capabilities on such systems requires careful performance tuning and optimization of the most critical kernels. The main computational kernels in most finite-volume unstructured mesh

[1]Intel Xeon and Intel Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.

CFD codes can be broadly classified as follows (1) Edge-based stencil loops such as flux and gradient computations; (2) Sparse, narrow-band recurrences including factorization and matrix solvers; (3) Global collective operations, from the inner products and norms; and (4) Vertex-based loops, for the state updates. We characterize these common compute patterns to identify primary bottlenecks and detail effective optimization strategies on both Xeon processors and the Xeon Phi coprocessor. For this purpose, we use three of the modern and widely used CFD codes: SU2, Petsc-FUN3D, and OpenFOAM.

2 Architectural Features of the Intel Xeon Phi Coprocessor

The Intel Xeon Phi coprocessor has several important features that are different from typical Intel Xeon products. Learning these features and how to make best use of them when programming and tuning HPC applications is critical to getting the best performance from the coprocessor. We provide an overview of its system-level and chip-level topology as well the core architecture. We also cover the Vector Processing Units (VPU) including a general overview of the ISA features and provide tips on how software should issue instructions for best performance. In addition, we cover programming models including the different ways that this platform can be used by developers and practitioners working on CFD codes.

3 CFD codes

In this section we describe the CFD codes on which we apply the various optimizations to improve their performance on modern multi-/many-core architectures.

3.1 SU2

SU2 is an open source code for CFD analysis and design optimization [1]. It solves complex, multi-physics analysis and optimization tasks using arbitrary unstructured meshes. It is designed so that it is easily extensible for the solution of general Partial Differential Equations (PDEs)-based problems. For the implicit solver in SU2, one may use a non-linear multigrid iteration at the outer-level, and the inner linear system (linearized using the Jacobian) may be solved using a preconditioned GMRES solver. In this paper, we build up on our recent work [2] on optimizing SU2 for many-core platforms and present techniques to expose more thread-level parallelism, optimize memory/cache re-use, and modify data layouts to gain speedups from vectorization of key flux residual kernels.

3.2 PETSc-FUN3d

PETSc-FUN3d is a popular CFD benchmark. W.K. Anderson*et al.* [3] won the Gordon-Bell award for this code in 1999 by demonstrating state-of-art performance. We re-examine this unstructured-grid implicit flow solver in light of modern highly parallel architectures and build up on our recent work [4]. PETSc-FUN3D uses an implicit Newton-Krylov-Schwarz (NKS) solver with pseudo-timestepping. The Newton iteration is used as the outer non-linear solver, with an additive Schwarz based ILU preconditioned GMRES

as the inner linear solver.

3.3 OpenFOAM

OpenFOAM [5] is a well-known software package for solving different kinds of partial differential equations and is popular for CFD especially in the automotive segment. The Intel Xeon Phi coprocessor provides an architecture that supports native C/C++ and MPI communication without the need to rewrite OpenFOAM solvers. We will show the optimizations we have applied to the OpenFOAM core parts of the Geometric Algebraic Multigrid solver. Tuning techniques applied include improvements to vectorize the solver code and software prefetching to reduce memory latency. A key contribution is a novel hierarchical decomposition method that takes the heterogeneous hardware layout into account and that strives to reduce communication between the different cluster nodes and their respective coprocessor cards. Our work demonstrates that OpenFOAM can run on heterogeneous clusters with good scalability across 16 nodes each of which contains two Intel Xeon processors and two Intel Xeon Phi coprocessors. The benchmarks show up to a 1.4× better performance of the heterogeneous cluster over the Xeon-only cluster.

4 Conclusions

In summary, we extract a key theme of underlying optimizations for three widely used CFD codes which involves exposing more fine-scale concurrency, vectorization, and efficient use of memory. These three tuning stunts are equally critical to obtain good performance out of modern heterogeneous computing platforms.

REFERENCES

[1] F. Palacios, T. D. Economon, A. C. Aranake, S. R. Copeland, A. K. Lonkar, T. W. Lukaczyk, D. E. Manosalvas, K. R. Naik, A. S. Padrón, B. Tracey, A. Variyar, and J. J. Alonso, Stanford University Unstructured (SU^2): Open-source analysis and design technology for turbulent flows, *AIAA SciTech*, AIAA Paper 2014-0243, 2014.

[2] T. D. Economon, F. Palacios, J. J. Alonso, G. Bansal, D. Mudigere, A. Deshpande, A. Heinecke, and M. Smelyanskiy, Towards High-Performance Optimizations of the Unstructured Open-Source SU2 Suite, *AIAA SciTech*, AIAA Paper 2015-1949, 2015.

[3] W. K. Anderson, W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F.Smith, Achieving high sustained performance in an unstructured mesh CFD application, *Supercomputing*, ACM, 1999, p. 69

[4] D. Mudigere, S. Sridharan, A.M. Deshpande, J. Park, A. Heinecke, M. Smelyanskiy, B. Kaul, P. Dubey, D. Kaushik, and D. Keyes, Exploring shared-memory optimizations for an unstructured mesh cfd application on modern parallel systems, *Parallel & Distributed Processing Symposium (IPDPS)*, IEEE, 2015.

[5] www.openfoam.org