Abstract: Compiler-Assisted Scientific Workflow Optimization

Hadia Ahmed[1], Peter Pirkelbauer[2], Purushotham Bangalore[2], Anthony Skjellum[3]

[1]Lawrence Berkeley National Laboratory, Computational Research
[2]University of Alabama at Birmingham, Computer Science
[3]University of Tennessee at Chattanooga, SimCenter
[1]hahmed@lbl.gov, [2]{pirkelbauer, puri}@uab.edu, [3]tony-skjellum@utc.edu

Abstract

With increasing amount of data generated and with stringent power and memory requirements, data analytics will face tremendous challenges on Exascale systems.

This paper presents an early prototype of a compiler-based framework to optimize the workflow for data processing. Our approach is to analyze components across stages of the workflow, and merge some of the analytics code, such as data reductions, into the compute code. By doing so, the transformed workflow reduces data movement over the network, thereby increasing overall system performance.

1. Introduction

In realistic simulations, it is expected that in the scale-out to the level of leadership-class compute systems, there will be a division between compute nodes that generate data and centralized (or specialized/edge) nodes that deal with storage and data analytics. Each compute node will communicate its results to an analytics node for further processing. Several stages of analytics node may exist before the results reach central storage. To achieve a good ratio between compute nodes and analytics nodes, performing in-situ analysis on the compute nodes will be a necessity.

The key insight underlying our work is that scientific code and analytics code form a single cohesive unit. By analyzing the whole application, we can optimize the entire system in ways that are impossible if we process each module independently. For example, an holistic view allows us to identify components of the analytics code that can be better processed on nodes that run the simulation, thereby reducing the amount of inter-node data movement and reducing resource requirements on the analytics side. In some scenarios, it may be suitable to embed parts of the analytics code into the simulation code, thereby exploiting temporal cache locality and reducing intra-node data movement.

This paper presents a first early prototype of a ROSE based tool, that assists in making scientific workflows more efficient. The tool's goal is to jointly analyze and transform simulation and analytics code in ways that make the combined system more efficient.

2. System Design

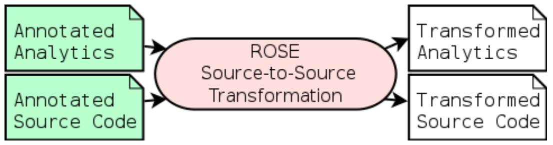

Figure 1. ROSE transformation pass restructures annotated source code and analytics code to reduce communication.

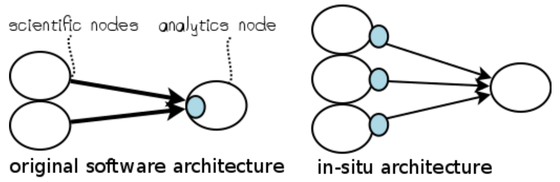

Figure 2. The image shows a case where parts of the data analytics code are merged into the scientific code. The reduced data movement may increase the ratio of compute to analytics nodes.

Fig. 1 gives a graphical view of our ROSE based transformation tool. Our tool analyzes annotated scientific codes and annotated analytics codes. The annotations describe how the whole workflow should be restructured. Following these annotations, our tool moves parts of the data analytics code into the scientific code. For example, reductions can be partially executed immediately after the data has been computed. Fig. 2 shows the transformation graphically. This reduces the amount of data that is communicated from data generating nodes to data processing nodes, and increases the ratio of compute nodes to data processing nodes.

3. Early Results

We have produced a first early prototype and tested it on a Bonds-Csym workflow. Bonds is a LAMMPS backend that computes bonds between atoms based on a distance threshold, and CSym is a central symmetry computation for an atom and adjacent atoms. We tested with the codes on a single node and used ADIOS for intermediate data storage. We annotated both Bonds and CSym, and our tool merged the CSym code into Bonds. The transformation eliminated a data container conversion needed to interact with ADIOS and the need for intermediate storage. Depending on data size, the transformed code runs between 4% and 12% faster.

Merging codes enables further node-level optimizations, such as fusion of data generating loops with analytics loops. For our test case, loop fusion did not yield any tangible speedup.

4. Conclusion and Future work

We have presented an early effort for transforming scientific workflows across node boundaries. Merging processing from across nodes reduces data movement. Thus far, we have experimented with analytics code moved into scientific codes in a 1:1 configuration. Extending this work, will require us to transform codes that operate in n:1 (or even n:m) configurations. To this end, we work on a more expressive annotation language. The ability to transform codes across node boundaries also creates opportunities to utilize heterogeneous hardware (*e.g.*, GPU, FPGAs) at the producer nodes more effectively.