Common Subexpression Elimination with Subtree Isomporphisms

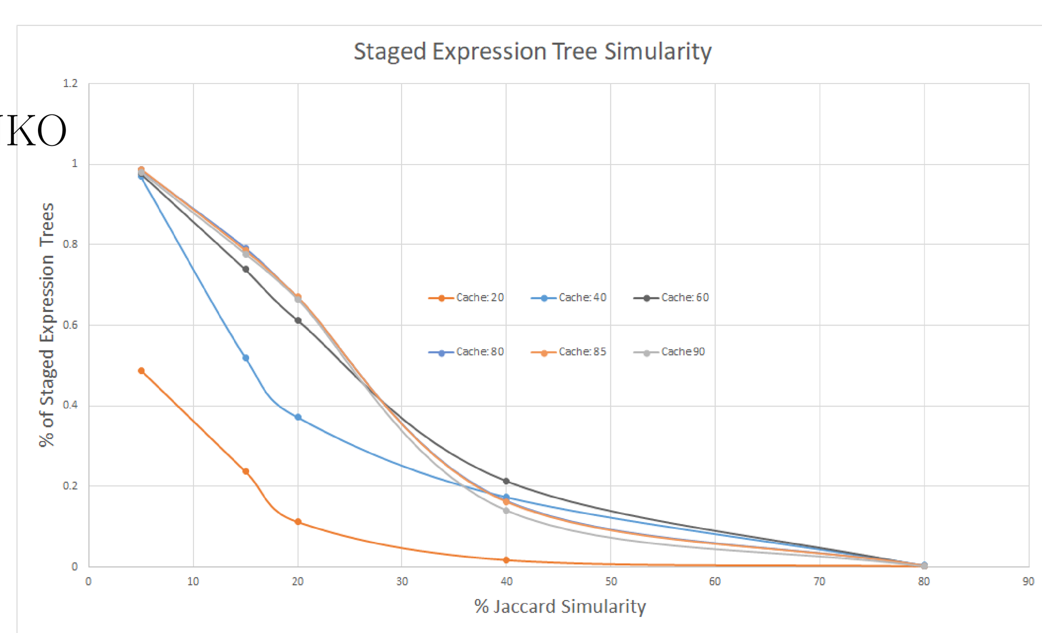
Robert King, Mentors: Hari Sundar, Milinda Fernando
University of Utah

Motivation

The purpose of this work is to improve the run time of the simulation of black hole collisions as seen in the Dendro-Gr framework. The BSSN equations and several other simulation codes consist of several complex partial differential equations and to model these equations the value of each variable is computed once per a time step in the model. The project presented aims to provide a method to increase cache utilization to increase runtime performance.

Contributions

- **Automatic code-generation**. Given the complexity of the Einstein equations, we have developed an automatic code generation framework for GR using `SymPy` that automatically generates architecture-optimized codes which helps to improve code portability.
- **Performance**. Developed a tree isomorphism algorithm for common subexpression elimination. The algorithm can be adapted to different architetures by specifying a cache size. The code is designed to be portability and easily add performance improvements to any code.
- **Visualization**. Code is able to represent expression trees with Graphiz and visualize expression tree staging
- **Proof of Concept** A proof of concept analysis was performed on the BSSNKO equations presented in the Dendro-GR code. The goal is to reduce the overall runtime of the equations through better cache utilization. Expression trees were staged using a greedy algorithm to target a given cache size. The plot illustrates how the overlap between trees increases as the cache size decreases. While the overlap may cause some parts of the expression tree to be duplicated, the overall run time will be faster due to less cache misses.



Symbolic code generation for BSSNKO equations

$$
\begin{aligned}
\partial_t \alpha &= \mathcal{L}_\beta \alpha - 2\alpha K, \\
\partial_t \beta^i &= \lambda_2 \beta^j \partial_j \beta^i + \frac{3}{4} f(\alpha) B^i \\
\partial_t B^i &= \partial_t \tilde{\Gamma}^i - \eta B^i + \lambda_3 \beta^j \partial_j B^i - \lambda_4 \beta^j \partial_j \tilde{\Gamma}^i \\
\partial_t \tilde{\gamma}_{ij} &= \mathcal{L}_\beta \tilde{\gamma}_{ij} - 2\alpha \tilde{A}_{ij}, \\
\partial_t \chi &= \mathcal{L}_\beta \chi + \frac{2}{3}\chi \left(\alpha K - \partial_a \beta^a\right) \\
\partial_t \tilde{A}_{ij} &= \mathcal{L}_\beta \tilde{A}_{ij} + \chi \left(-D_i D_j \alpha + \alpha R_{ij}\right)^{TF} + \\
&\quad \alpha \left(K \tilde{A}_{ij} - 2 \tilde{A}_{ik} \tilde{A}^k_j\right), \\
\partial_t K &= \beta^k \partial_k K - D^i D_i \alpha + \\
&\quad \alpha \left(\tilde{A}_{ij} \tilde{A}^{ij} + \frac{1}{3} K^2\right), \\
\partial_t \tilde{\Gamma}^i &= \tilde{\gamma}^{jk} \partial_j \partial_k \beta^i + \frac{1}{3} \tilde{\gamma}^{ij} \partial_j \partial_k \beta^k + \beta^j \partial_j \tilde{\Gamma}^i - \\
&\quad \tilde{\Gamma}^j \partial_j \beta^i + \frac{2}{3} \tilde{\Gamma}^i \partial_j \beta^j - 2 \tilde{A}^{ij} \partial_j \alpha + \\
&\quad 2\alpha \left(\tilde{\Gamma}^i{}_{jk} \tilde{A}^{jk} - \frac{2}{3\chi} \tilde{A}^{ij} \partial_j \chi - \frac{2}{3} \tilde{\gamma}^{ij} \partial_j K\right)
\end{aligned}
$$

```
from DENDRO_sym import *
a_rhs = Dendro.Lie(b, a) - 2*a*K
b_rhs = [3/4 * f(a) * B[i] +
l2*vec_j_del_j(b, b[i]) for i in e_i]
l2*vec_j_del_j(b, b[i])
for i in e_i]
B_rhs = [Gt_rhs[i] - eta * B[i] +
l3 * vec_j_del_j(b, B[i]) -
l4 * vec_j_del_j(b, Gt[i])
for i in e_i]
gt_rhs =  Dendro.Lie(b, gt) - 2*a*At
chi_rhs = Dendro.Lie(b, chi) +
2/3*chi*(a*K - del_j(b))
At_rhs = Dendro.Lie(b, At) + chi *
Dendro.TF(-DiDj(a) +
a*Dendro.Ricci) +
a*(K*At -2*At_ikAtKj)
K_rhs = vec_k_del_k(K) - DIDi(a) +
a*(1/3*K*K + A_ij_A_IJ(At))
```

The left panel shows the BSSNKO formulation of the Einstein equations. These are tensor equations, with indices $i, j, \ldots$ taking the values $1, 2, 3$. On the right we show the DENDRO_sym code for these equations. DENDRO_sym uses `SymPy` and other tools to generate optimized C++ code to evaluate the equations. Note that $\mathcal{L}_\beta$, D, ∂ denote Lie derivative, covariant derivative and partial derivative respectively, and we have excluded $\partial_t \tilde{\Gamma}^i$ from DENDRO_sym to save space.

For additional details please refer to **Massively Parallel Simulations of Binary Black Hole Intermediate-Mass-Ratio Inspirals**, Milinda Fernando, Hari Sundar `https://arxiv.org/abs/1807.06128` and the Dendro project.

Methods

This research for consists of two main projects. The first is the subtree isomorphism problem that will focus on the common subexpression elimination and the second is a lower bound analysis for the number of temporary variables needed to solve the partial differential equations. The goal is to create an algorithm that will be able to analyze the different partial differential equations and reorder the temporary variable calculations to maximize cache effectiveness and variable reuse.

Staging

Staging is focused on finding variable reuse within the partial differential equations. The first problem is to create an expression tree from the partial differential equations generated from the SymPy auto generated code. Once the expression tree is created, subtree isomorphism analysis can begin. This will be a bottom up approach that considers the values used in each leaf node in addition to finding similar tree structure. Each node within the tree will keep track of all the leaf values that it depends on. Set similarity will be used as a precondition before calculating the more expensive tree isomorphism. By the end of the Staging Process the most common subexpressions will be identified. Each expression will be valued depending on the number of leaf node dependents, to mitigate the total number of temporary variables, and the number of times each expression appears, to maximize data reuse.

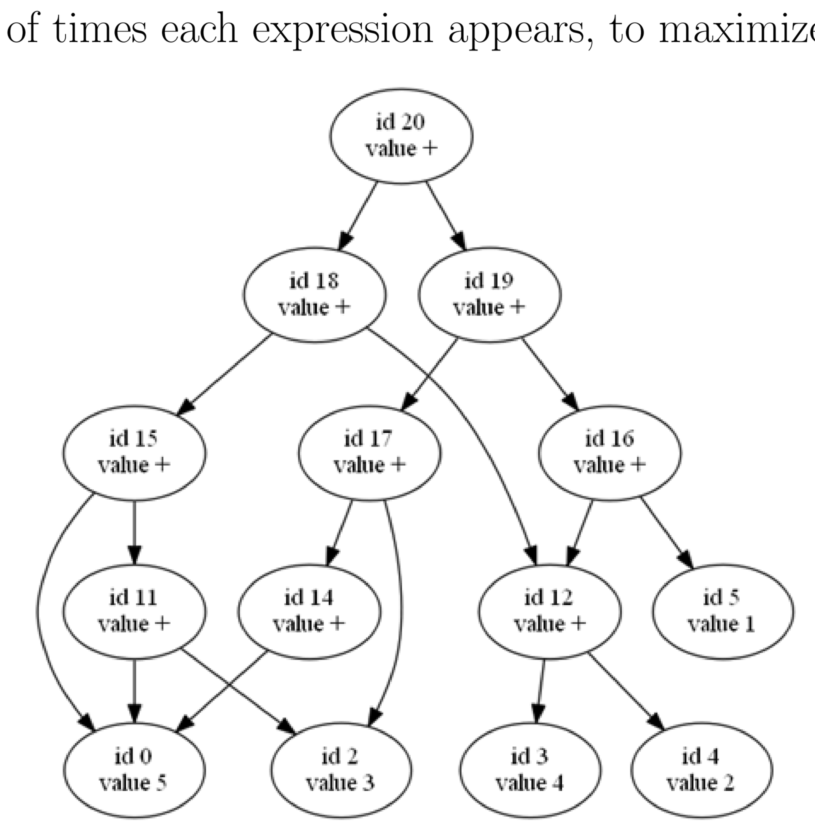


Figure 3: Staged Expression Tree

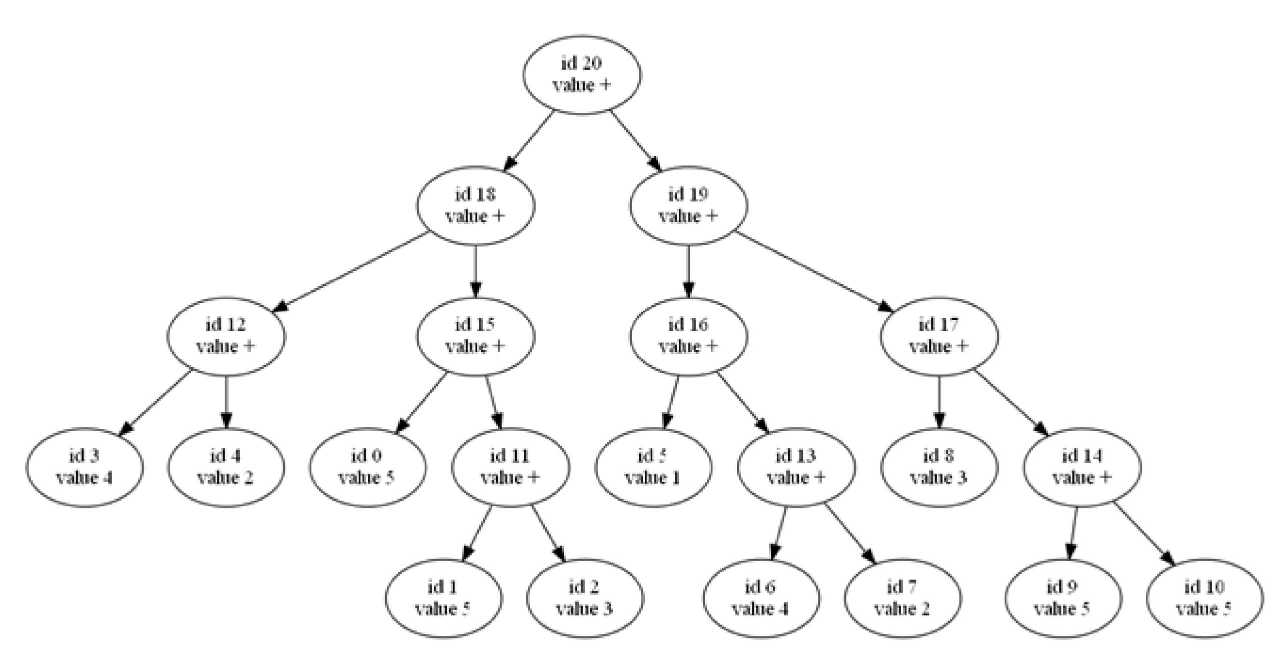


Figure 1: Initial Expression Tree

Rebuilding

The rebuilding phase takes the results from the staging process and rebuilds the expression tree. In order of importance the rebuilding phase must preserve the correct final answers, maximize cache effectiveness, and minimize the total number of temporary variables used. The current strategy is to use a dynamic programming approach. Dynamic programming will also allow for the rebuilding process to scale effectively. Once complete, these results will be measured experimentally. Once a proof of concept has been completed the rebuilding process will take in parameters such as memory architecture, and L1 cache size to optimize the calculations for each machine. Once the rebuilding process is completed a parser will be created to transform the original SymPy autogenerated code to reflect the updated expression tree.

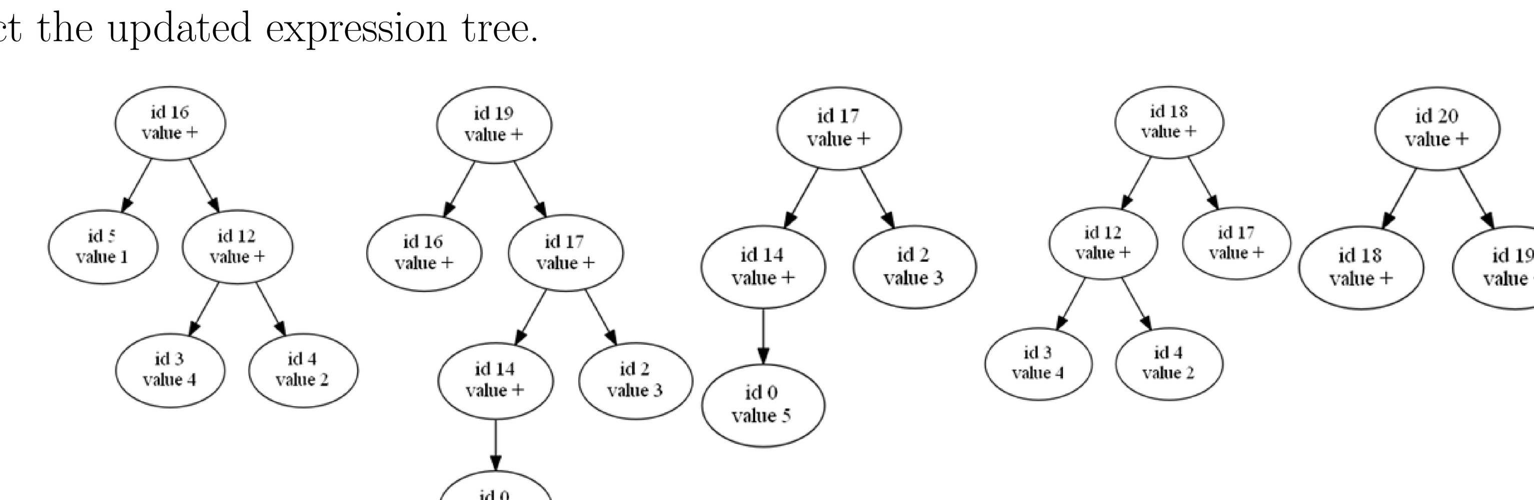


Figure 2: Rebuilt Expression Tree