**SAIMI**: Separating Algorithm and Implementation via Programming Model Injection

**Problem**: Algorithms in programs are obfuscated due to implementation details and performance tuning done per machine and by hand.

**Solution**: Use simpler, more restricted programming models to express important sub-computations. Express implementation details as transformations of sub-computations.

**Status**: We are evaluating separation in existing programming models and developing look-up table, grid, and task graph injectable programming models.

---

**Evaluating Existing Separation Mechanisms**

Programming models should target minimal tangling and maximal programmer control.

![Illustration of the SAIMI concept.](image)

- Evaluating Existing Separation Mechanisms
  - Lookup Table Injectable Programming Model
    - Developed a source-to-source code translation tool using ROSE from LLNL to apply the LUT optimization to expressions marked with pragmas. Speeded up a discrete scattering code by 7.1x, a continuous scattering code by 2.6x, and a geospatial code by 4.5x. Below is error versus performance for discrete scattering code.

---

**Full Application**

```
Source to Source Compilation Tool
```

```
Optimized Code for Machine A
Optimized Code for Machine B
```

This won’t hurt a bit!

---

**Early Career Award, DOE grant DE-SC3956**

April 2010 through March 2015

[Visit the website for more information.](http://www.cs.colostate.edu/hpc/SAIMI/)
SAIMI: Separating Algorithm and Implementation via Programming Model Injection

Grid-Based Computation

Many simulation applications become tangled with the underlying discretization grid. We are developing ways to express grid details orthogonally from the computation performed on the grid. We will evaluate our solutions with CGPOP, which is a miniapp that models the conjugate gradient solver in the Parallel Ocean Program. The figure below shows that the 3000 line CGPOP mini app behaves as a performance proxy for the 75K line POP application.

Dynamic Task Graphs

Irregular task graphs occur in sparse computations where dependence structure due to grid topology or sparse matrix structure is not known until runtime. Using task graphs as an execution model enables parallel code from different programming models to be executed by a common scheduler. We are investigating the ability of existing programming models including Concurrent Collections, Open MP Tasks, and Cilk++ to specify and execute dynamic task graphs. We are also developing an injectable programming model for dynamic task graphs that enables the expression of locality between tasks so as to improve performance on multicore machines with non-uniform shared memory access.

Future Directions

We plan to add additional injectable programming models to our existing source-to-source compilation tool and to apply this tool to more DOE applications.

<table>
<thead>
<tr>
<th>Participant</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michelle Strout</td>
<td>Principal Investigator and developer of sparse polyhedral framework.</td>
</tr>
<tr>
<td>Christopher Krieger</td>
<td>Ph.D. student investigating dynamic task graphs.</td>
</tr>
<tr>
<td>Andrew Stone</td>
<td>Ph.D. student investigating orthogonal specification of atmosphere grids.</td>
</tr>
<tr>
<td>Christopher Wilcox</td>
<td>Ph.D. student developing tools to semi-automate look-up table optimizations.</td>
</tr>
<tr>
<td>John Dennis</td>
<td>NCAR collaborator on CGPOP mini app. Provided CGPOP vs. POP graph.</td>
</tr>
<tr>
<td>Stephanie Dinkins</td>
<td>Graduate student doing performance analysis of Sparse Matrix Vector multiply.</td>
</tr>
<tr>
<td>Alan LaMielle</td>
<td>Graduate student developing Sparse Polyhedral Framework software.</td>
</tr>
</tbody>
</table>