OpenMP Visualization Tool

Josh Marsico, UNC Charlotte
Dr. Tyler Allen, College of Computing and Informatics

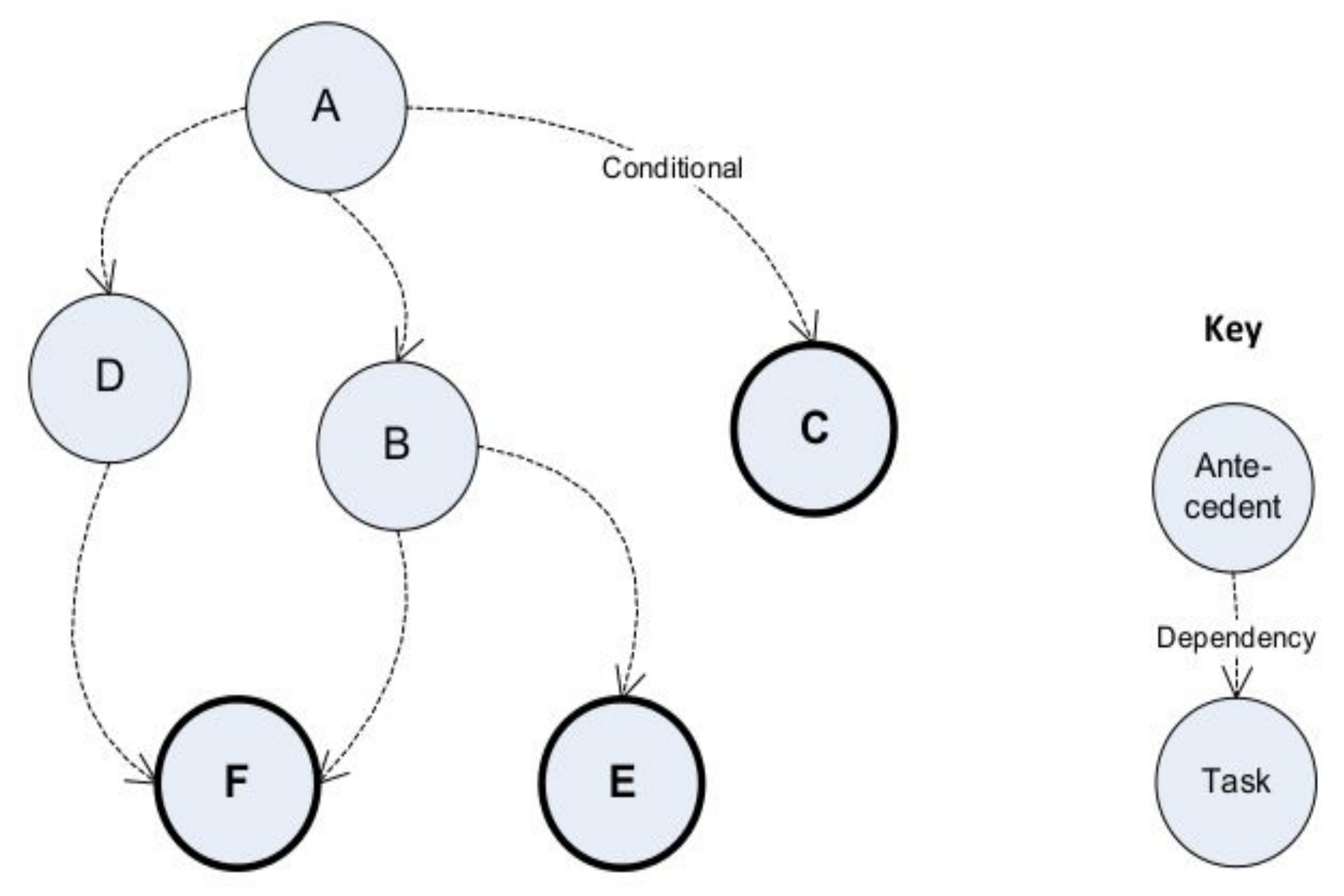
UNIVERSITY OF NORTH CAROLINA CHARLOTTE

Introduction

- Teaching Parallel Computing is challenging.
- Requires good understanding of complex ideas.
  - Sequential computing
  - Abstraction of computational tasks
- Visualizing these concepts will help students.
  - We propose the development of visualizations to enable break through of learning.

Motivation

- Parallel task graphs (PTG) are a modeling tool for understanding how parallel programs execute.
- Students are taught to examine parallel problems by using PTGs.
  - For real world problems, drawing a PTG becomes too complicated.



- This would have millions of nodes for a real world application.
- Build tool to automatically give students instant feedback as they make changes.

Objectives

- Build OpenMP plugin that implements callbacks to collect data.
- Add task data callback to plugin.
- Add dependency logging callback to plugin.
- Get plugin working with various benchmarks.
- Format the output in DOT file.
- Graph output file as visualization.

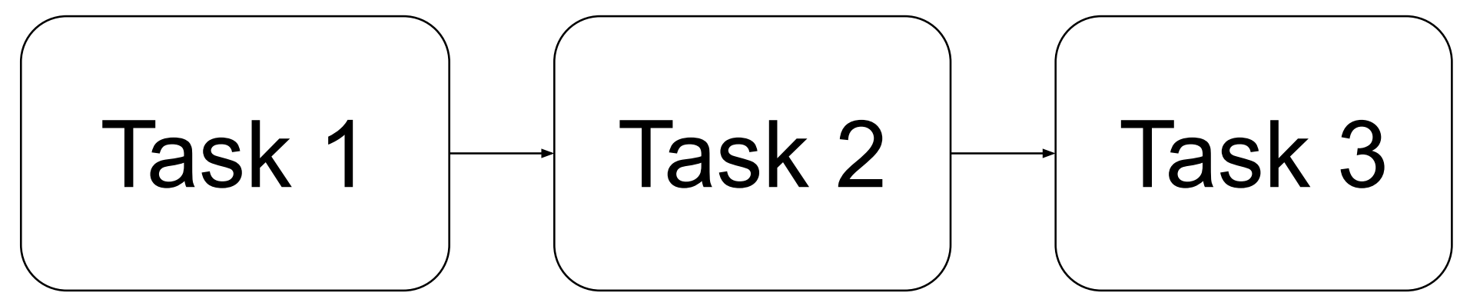
1 Basic OpenMP code

```
#pragma omp for
for (int i = 0; i < 4; i++) {
        int threadNum = omp_get_thread_num();
        int numThreads = omp_get_num_threads();
        printf("Num threads%d\n", numThreads);
        printf("Hello World!%d\n", threadNum);
}
```

2 OpenMP callback tool

```
// Add labels for task types
 if (flags & ompt_task_initial)
        fprintf(output_file, "\\n(initial)");
 if (flags & ompt_task_implicit)
        fprintf(output_file, "\\n(implicit)");
 if (flags & ompt_task_explicit)
        fprintf(output_file, "\\n(explicit)");
```

3 Output visualized as task graph

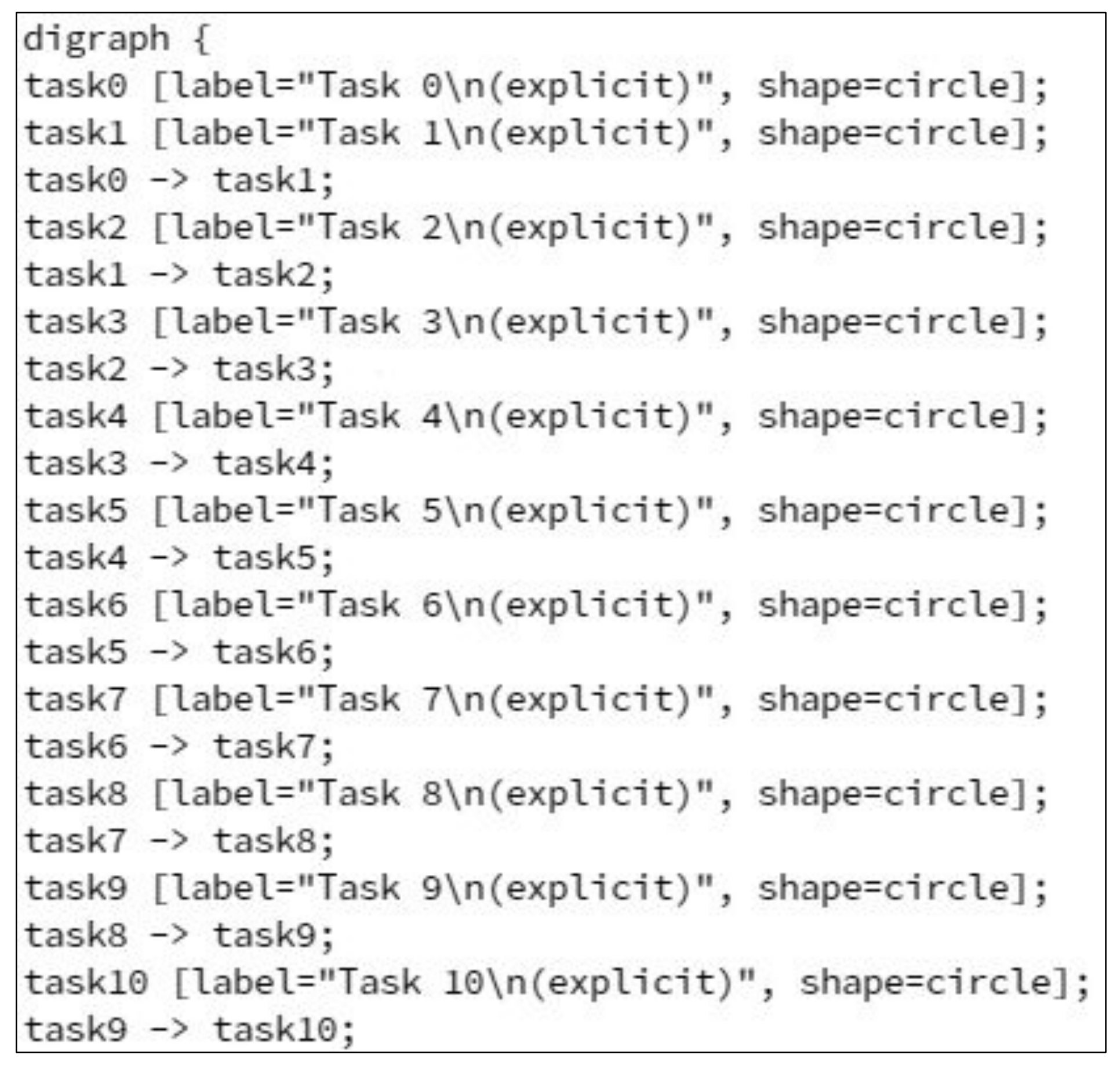


Method

- Start with basic OMP code from student classes / test code
- Develop a tool that visualizes parallel tasks
- Test tool with benchmark code to check dependencies and data logging
- Try tool in classroom to evaluate if it helps with student outcomes

Results

- Current version at: github.com/Origin-code/callbacktest
- Callback that logs task data for parallel code
- Currently generates correct nodes not dependencies
- Working on adding dependencies with the current results and progress
- Current output is DOT format task data with basic linking dependencies.
- Creates a simple task graph when plotted.



```
digraph {
task0 [label="Task 0\n(explicit)", shape=circle];
task1 [label="Task 1\n(explicit)", shape=circle];
task0 -> task1;
task2 [label="Task 2\n(explicit)", shape=circle];
task1 -> task2;
task3 [label="Task 3\n(explicit)", shape=circle];
task2 -> task3;
task4 [label="Task 4\n(explicit)", shape=circle];
task3 -> task4;
task5 [label="Task 5\n(explicit)", shape=circle];
task4 -> task5;
task6 [label="Task 6\n(explicit)", shape=circle];
task5 -> task6;
task7 [label="Task 7\n(explicit)", shape=circle];
task6 -> task7;
task8 [label="Task 8\n(explicit)", shape=circle];
task7 -> task8;
task9 [label="Task 9\n(explicit)", shape=circle];
task8 -> task9;
task10 [label="Task 10\n(explicit)", shape=circle];
task9 -> task10;
```

Conclusions

- Built OMPT plugin that tracks task data and dependencies for parallel code.
- Got plugin working with multiple benchmarks.
- Currently working on improving dependency logging.

Future

- Get dependency logging fully working.
- Begin printing output task graphs.
- Add color and other tools to help emphasize significant tasks.
- Begin testing in classrooms.
  - Have students use in their projects.
  - Evaluate if it helps students be successful in class and learning outcomes.