Annotated Task Graph (ATG)

Ananya Muddukrishna

ananya@kth.se

Revision History

Revision	Date	Author(s)	Description
1.0	2014-06-19	ananya	Created
1.1	2014-07-01	ananya	Improved ATG generation text

1 Introduction

The Annotated task Graph (ATG) refers to information obtained by the instruction-level profiling feature of the MIR runtime system library.

The ATG is available in two forms - a raw form and a visual form.

2 Getting the ATG

Instruction-level profiling is performed using a custom Pin call-graph profiling tool which takes two lists as arguments. The first argument list contains names of outline functions which represent tasks. The second argument list contains names of functions which are called within tasks. Instruction count of functions called within a task are added to the task instruction count.

We first identify outline functions and functions called within tasks using a simple script which searches for known outline function name patterns within the object files of the program. The script also lists all function symbols within the object files as potentially callable functions. We should filter the callable function list for those which we are certainly called by tasks defined in the program. Inspecting source files is a quick way to identify certainly

called functions. If in doubt or when sources are not available, we can use the entire callable function list.

```
$ cd $MIR_ROOT/test/fib

$ echo "Examining executable for names of outline and callable functions ..."
$ $MIR_ROOT/scripts/task-graph/profiler_params.py prof-build/*.o
Using "._omp_fn.|ol_" as outline function name pattern
Processing file: prof-build/fib.o
OUTLINE_FUNCTIONS=ol_fib_0,ol_fib_1,ol_fib_2
CALLABLE_FUNCTIONS=fib_seq,fib,get_usecs,main
```

In the case of the fib program, we can exclude main and get_usecs from the called functions list. Now we invoke the Pin call-graph profiler with appropriate arguments. The profiler executes the fib program and provides an instruction-level profile of tasks. We additionally instruct MIR to provide fork-join task graph information.

```
$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PIN_ROOT/intel64/runtime \
    MIR_CONF="-w=1 -g -p" \
    $PIN_ROOT/intel64/bin/pinbin \
    -t ${MIR_ROOT}/scripts/task-graph/obj-intel64/
      mir_outline_function_profiler.so \
    -o fib_test \
    -s ol_fib_0,ol_fib_1,ol_fib_2 \
    -c fib,fib_seq \
    -- ./fib-prof 10 4

$ mv mir-task-graph fib_test-fork_join_task_graph
```

Now, we can get some basic task-based infmration by processing the fork-join task graph produced by MIR.

```
$ echo "Summarizing fork-join task graph ..."
$ Rscript ${MIR_ROOT}/scripts/task-graph/mir-fork-join-graph-info.R
      fib_test-fork_join_task_graph
```

We can also plot the fork-join task graph.

```
$ echo "Plotting fork-join task graph ..."
$ Rscript ${MIR_ROOT}/scripts/task-graph/mir-fork-join-graph-plot.R
      fib_test-fork_join_task_graph color
```

Now, we combine the instruction-level profile produced by the Pin call-graph profiler with the fork-join task graph information produced by MIR. We

call this process as annotating the fork-join task graph with instruction-level information. The annotation process produces the raw format of the ATG.

```
$ echo "Annotating fork-join task graph ..."
$ Rscript ${MIR_ROOT}/scripts/task-graph/mir-annotate-graph.R fib_test-
    fork_join_task_graph fib_test-call_graph fib_test
```

We can plot the raw ATG format to produce the visual form of the ATG.

```
$ echo "Plotting annotated task graph ..."
$ Rscript ${MIR_ROOT}/scripts/task-graph/mir-annotated-graph-plot.R
    fib_test-annotated_task_graph color
```

Let us list the files produced by running the above commands. See Table 2 for a description of files produced.

```
$ echo "Listing ATG files ..."
$ ls fib_test*
```

3 Raw format of the ATG

The ATG raw format is a csv file.

```
$ head fib_test-annotated_task_graph
"task","parent","joins_at","tgpid","ins_count","stack_read","stack_write","
    mem_fp","ccr","clr","mem_read","mem_write","name"
1,0,0,"0.",59,11,15,5,12,15,4,1,"ol_fib_2"
2,1,0,"1.",60,10,15,5,12,15,4,1,"ol_fib_0"
3,1,0,"2.",60,10,15,5,12,15,4,1,"ol_fib_1"
4,3,0,"1.2.",60,10,15,5,12,15,4,1,"ol_fib_0"
5,3,0,"2.2.",60,10,15,5,12,15,4,1,"ol_fib_1"
6,5,0,"1.2.2.",60,10,15,5,12,15,4,1,"ol_fib_0"
7,5,0,"2.2.2.",60,10,15,5,12,15,4,1,"ol_fib_1"
8,7,0,"1.2.2.2.",68,15,15,5,14,17,4,1,"ol_fib_0"
9,7,0,"2.2.2.2.",47,10,10,5,9,12,4,1,"ol_fib_1"
```

Each line shows properties of an explicit task executed by the program. The first line shows names of the properties. Properties are also called annotations. See Table 3.

File name	Description
fib_test-call_graph	Instruction-level information of tasks
fib_test-mem_map	Memory map of program execution
fib_test-fork_join_task_graph	Parent-child task relationship and tgpid information
fib_test-annotated_task_graph	Raw format of the ATG combining instruction-level and parent-child information
fib_test-annotated_task_graph.adjm	Adjacent matrix representation of the visual format of ATG
fib_test-annotated_task_graph.dot	Dot representation of the visual format of ATG
fib_test-annotated_task_graph.edgelist	Edgelist representation of the visual format of ATG
fib_test-annotated_task_graph.graphml	GraphML representation of the visual format of ATG
fib_test-annotated_task_graph.info	Summary information about visual format of the ATG. Includes work, span and critical path from Cilk theory.
fib_test-fork_join_task_graph.adjm	Adjacent matrix representation of the visual format of ATG without instruction-level information
fib_test-fork_join_task_graph.dot	Dot representation of the visual format of ATG without instruction-level information
fib_test-fork_join_task_graph.edgelist	Edgelist representation of the visual format of ATG without instruction-level information
fib_test-fork_join_task_graph.graphml	GraphML representation of the visual format of ATG without instruction-level information
fib_test-fork_join_task_graph.info	Summary information about visual format of ATG without instruction-level information. Includes number of tasks and join degree distribution.

Table 2: ATG files

3.1 Property tgpid

The tgpid uniquely identifies a task irrespective of single-thread or many-thread execution. The format of tgpid is A.B.C.....<implied 0>

- 0. Represents the first task created. This is a special meaning.

Field	Description
task	Identifier of the task
parent	Identifier of the parent task of the task
joins_at	Indicates at which call to taskwait in the parent the task synchronized. Example: 0 indicates the task synchronized with the first call to taskwait in the parent. Several children can synchronize at the same call.
tgpid	Indicates the task graph position identifier. See details below.
ins_count	Indicates total number of instructions executed by the task. Profiling parameters indicate which instructions to count. Typically, instructions part of runtime system calls are excluded and calls to statically-linked functions are included.
stack_read	Indicates number of read accesses to the stack while executing instructions
stack_write	Indicates number of write accesses to the stack while executing instructions
ccr	Computation to Communication ratio. Indicates number of instructions executed per read or write access to memory
clr	Computation to Load ratio. Indicates number of instructions executed per read access to memory
mem_read	Indicates number of read accesses to memory (excluding stack) while executing instructions
mem_write	Indicates number of write accesses to memory (excluding stack) while executing instructions
name	Indicates name of the outline function of the task

Table 3: Raw format fields

- A. means Ath child of the first task
- A.B. means Ath child of task B.
- A.B.C. means Ath child of task B.C.

NOTE: The tgpid is an experimental feature, not fully tested and subject to change.

4 Visual format of the ATG

The visual format gives shape to the raw format of the ATG. It describes task-based execution in an intuitive manner allowing the programmer to spot performance problems. The visual format can be viewed using graph visual-

ization tools such as dot, yEd and cytoscape. See Figure 1 for visualization of the ATG on yEd and Figure 2 for visualization of the ATG on Dot. Details of the visual format are subject of a scientific paper under review and will be made available soon.

```
$ echo "Visualizing annotated task graph ..."
$ dot -Tps fib_test-annotated_task_graph.dot >  fib_test-
    annotated_task_graph.dot.ps
$ yed fib_test-annotated_task_graph.graphml
```

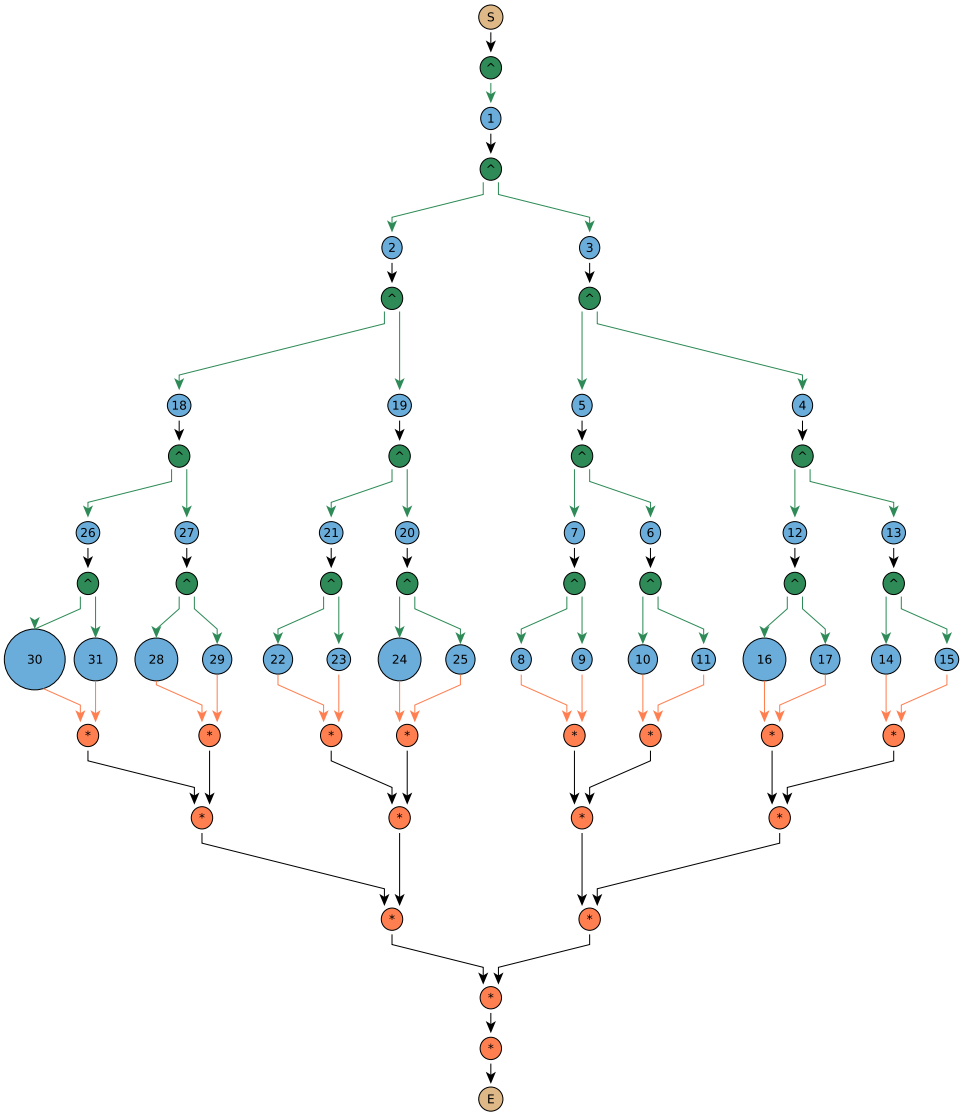

Figure 1: fib_test-annotated_task_graph.graphml viewed on yEd

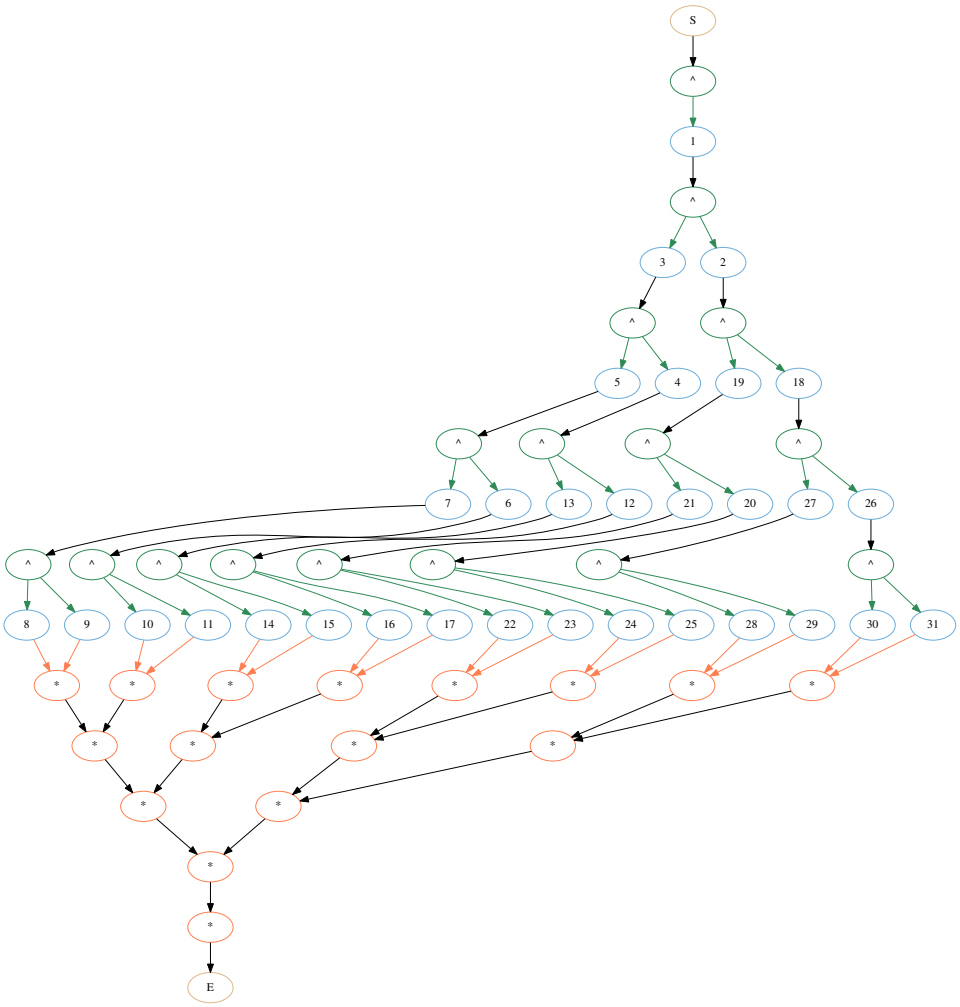

Figure 2: fib_test-annotated_task_graph.dot visualized using Dot