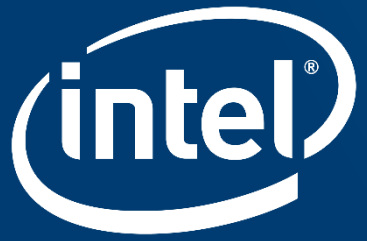


ANALYZING OPENMP* AND THREADING BUILDING BLOCKS (TBB) TASK GRAPHS USING INTEL® ADVISOR - FLOW GRAPH ANALYZER (FGA)

Vishakha Agrawal

Pablo Reble

Michael Voss

Mark Lubin

Vasanth Tovinkere

Overview

- Motivation: Task based Parallel Programing
- Threading Building Blocks (TBB) and OpenMP* Tasking with dependencies
- Introduction to Intel® Advisor -- Flow Graph Analyzer (FGA)
- FGA extension to support OpenMP tasks with dependencies
- Future work
- Summary

Task based parallel programming with dependencies

Advantages of task-based parallelism:

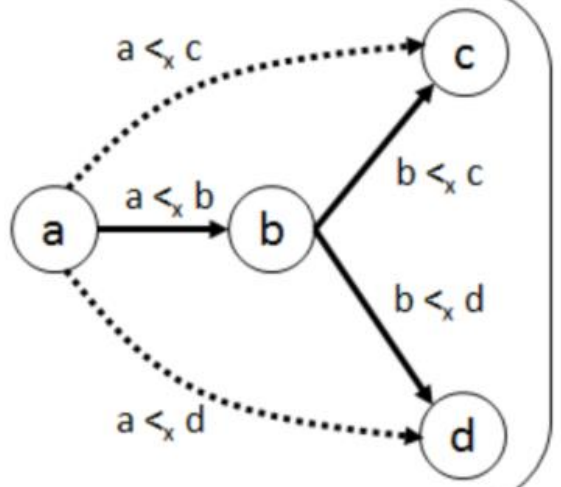
- Makes parallelization efficient for irregular and runtime dependent execution
- Higher level thinking
- Improved load balancing

Additional benefits for tasks with dependencies:

- Extends the expressiveness of task-based parallel programming
- Reduces need for global synchronization mechanisms like task barriers

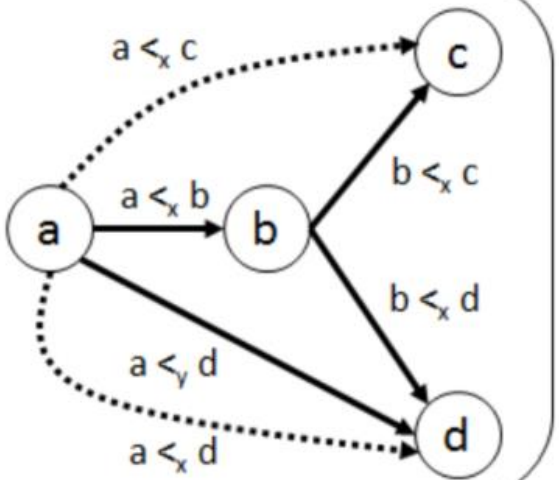
OpenMP* Task dependencies

```
#pragma omp task depend(out: x)
  a(&x)
#pragma omp task depend(inout: x)
  b(&x);
#pragma omp task depend(in: x)
  c(x)
#pragma omp task depend(in: x)
  d(x)
```



(a)

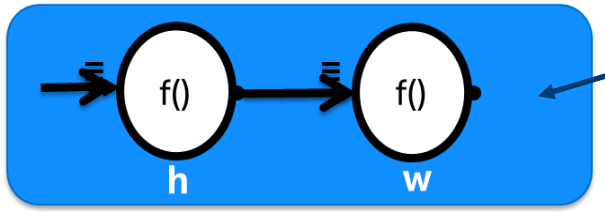
```
#pragma omp task depend(out: x, y)
  a(&x, &y)
#pragma omp task depend(inout: x)
  b(&x);
#pragma omp task depend(in: x)
  c(x)
#pragma omp task depend(in: x, y)
  d(x, y)
```



(b)

Explicit vs implicit graphs:

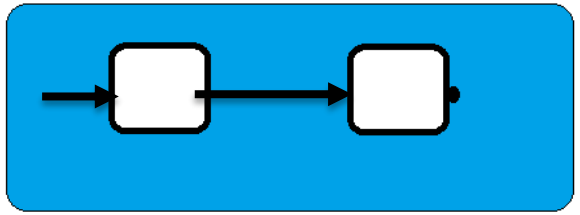
Hello World



Explicit graph, nodes, and edges

```
graph g;
continue_node< continue_msg > h( g,
    []( const continue_msg & ) {
        cout << "Hello ";
    } );
continue_node< continue_msg > w( g,
    []( const continue_msg & ) {
        cout << "World\n";
    } );
make_edge( h, w );
h.try_put(continue_msg());
g.wait_for_all();
```

Threading Building Blocks (TBB) Flow Graph



Implicit graph, nodes, and edge

```
#pragma omp parallel
{
   #pragma omp single
    {
       std::string s ="";
       {
         #pragma omp task depend(out: s)
                  {
                  s = "Hello ";
                  printf("%s", s);
                   }
         #pragma omp task depend(out : s)
                 {
                      s = "World";
                      printf("%s, s);
                  }
        }
}
```

OpenMP* Tasks with dependencies example

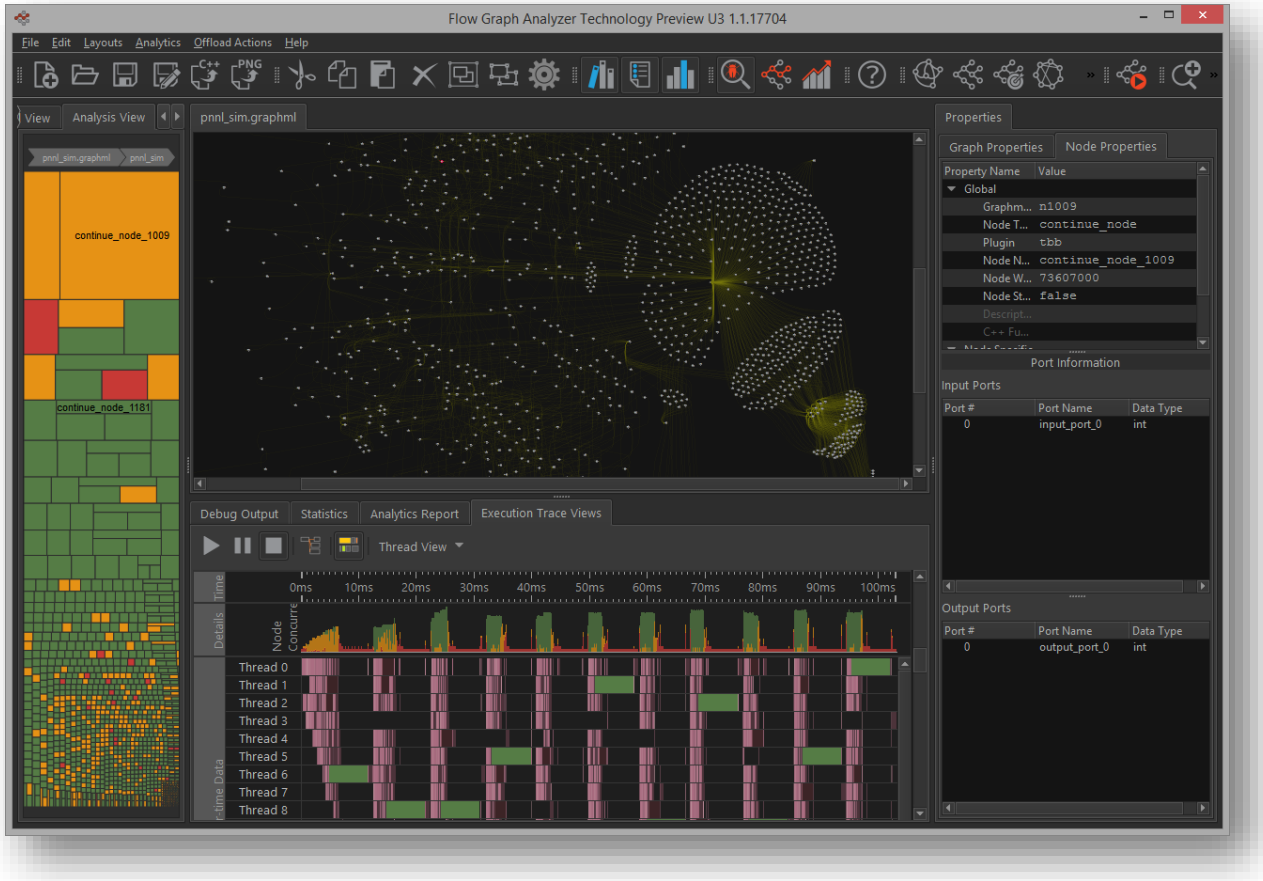
As part of Intel® Parallel Studio XE: Intel® Advisor's Flow Graph Analyzer

Tool supports analysis and design of parallel Applications using the Threading Building Blocks (TBB) Flow Graph API.

Limited support available for OpenMP*

Available for Windows*, Linux* and MacOS*

https://software.intel.com/en-us/articles/getting-started-with-flow-graph-analyzer

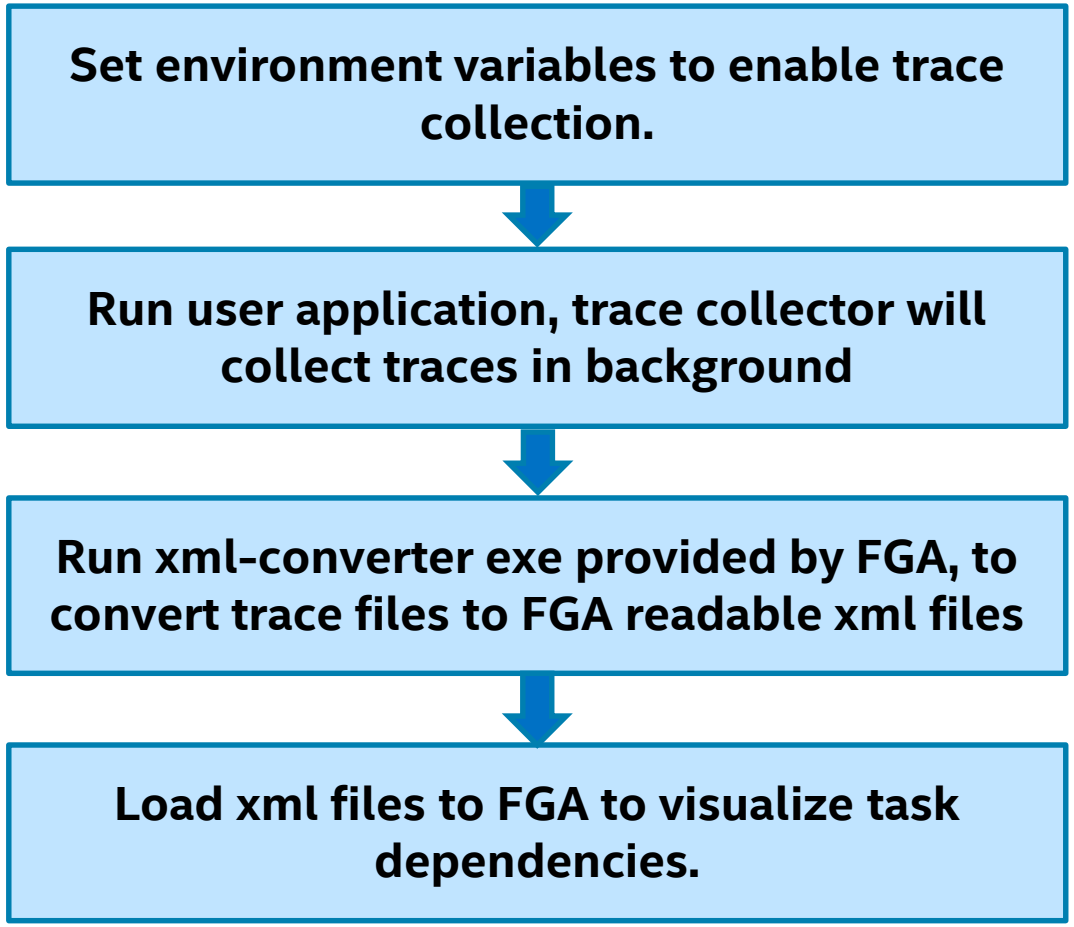


Extensions to FGA to support OpenMP* tasks

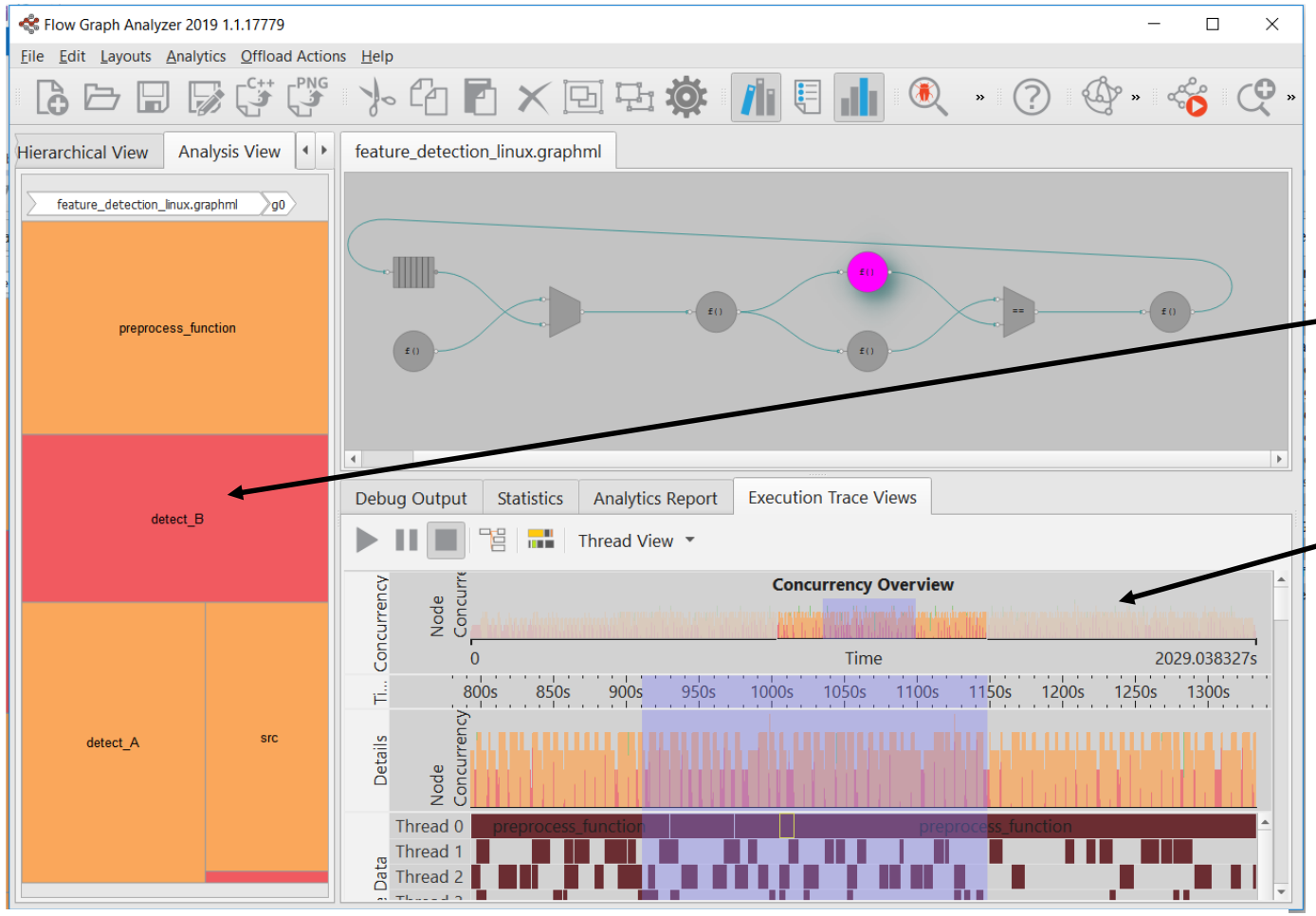
- FGA was primarily developed to support design, debugging, visualization and analysis of graphs built using Threading Building Blocks (TBB) flow graph API.
- Nevertheless, FGA capabilities are generic and can be applied to analyze computational graphs.
- OpenMP* visualization support is recently added in FGA, released in June 2018, which allows user to visualize parallel region execution traces.
- Support for task dependency graph, critical path length, tree map view and display of task execution trace for OpenMP* tasks has **not** been released.

NOTE: OpenMP* task dependence support is not yet released in FGA

FGA Usage:



Flow Graph Analyzer (Analyzer Workflow)



Treemap view gives you the general health of the graph's performance along with the ability to dive to the node level.

The concurrency histogram shows the parallelism achieved by the graph over time. You can interact with this chart by zooming in to a region of time, for example during low concurrency.

The concurrency histogram remains at the initial zoom level, and the zoomed in region is displayed below it.

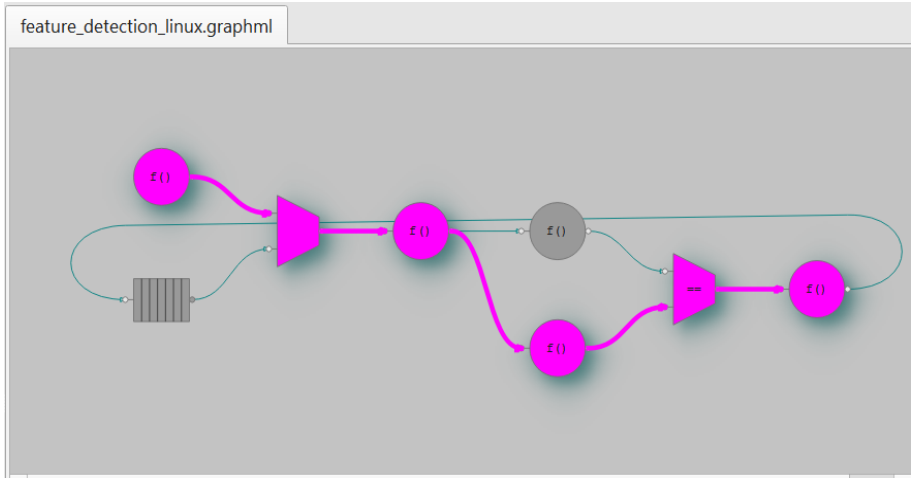
The per-thread task view shows the tasks executed by each thread along with the task durations.

Critical Path Analysis

Identify parts of your Applications on the critical path.

Unique capabilities of mapping trace data to an Applications graph structure

- Flow Graph instrumentation built-in to Threading building Blocks (TBB)
- Visualize Nested TBB algorithm♣



♣ TBB Parallel for instrumentation available since TBB 2018 Update 1

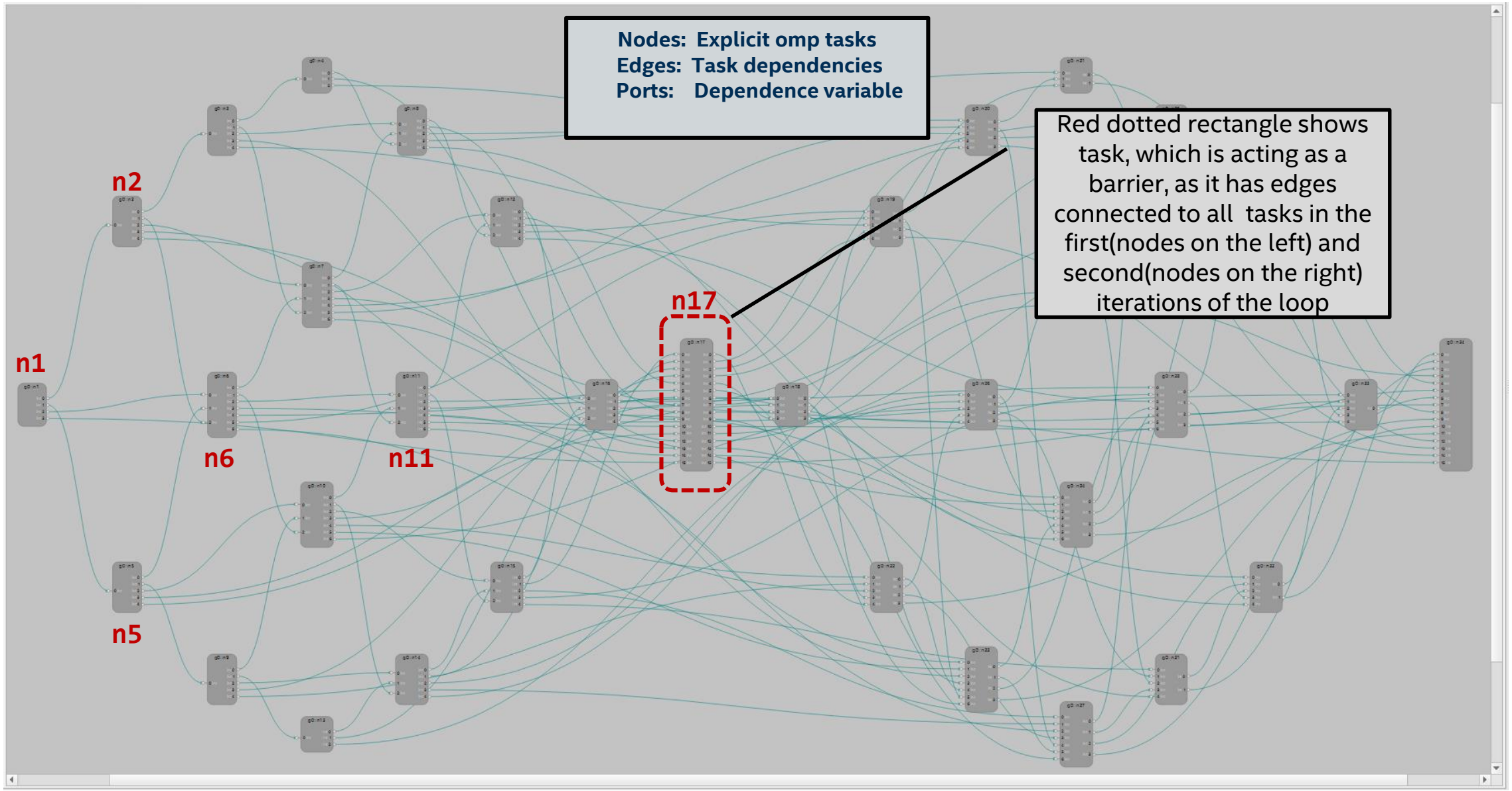
Does FGA provide actionable feedback?

```
1  pipeline_time = prk_wtime();
2  int lic = (m/mc-1) * mc + 1;
3  int ljc = (n/nc-1) * nc + 1;
4  for (int iter = 0; iter<=iterations; iter++) {
5      for (int i=1; i<m; i+=mc) {
6          for (int j=1; j<n; j+=nc) {
7  #pragma omp task depend(in:grid[0],grid[(i-mc)*n+j],\
8                                grid[i*n+(j-nc)],\
9                                grid[(i-mc)*n+(j-nc)]) \
10                       depend(out:grid[i*n+j])
11             sweep_tile(i, MIN(m,i+mc), j, \
12                              MIN(n,j+nc), n, grid);
13         }
14     }
15 #pragma omp task depend(in:grid[(lic-1)*n+(ljc)])\
16                  depend(out:grid[0])
17     grid[0*n+0] = -grid[(m-1)*n+(n-1)];
18 }
19 #pragma omp taskwait
20 pipeline_time = prk_wtime() - pipeline_time;
```

Example: Synch_p2p from the Parallel Research Kernels

Complete code can be accessed at: https://github.com/ParRes/Kernels/blob/master/C1z/p2p-tasks-openmp.c

Synch_p2p: Task dependence graph in FGA:



NOTE: OpenMP* task dependence support is not yet released in FGA

Optimization 1: Replacing task with "taskwait" construct

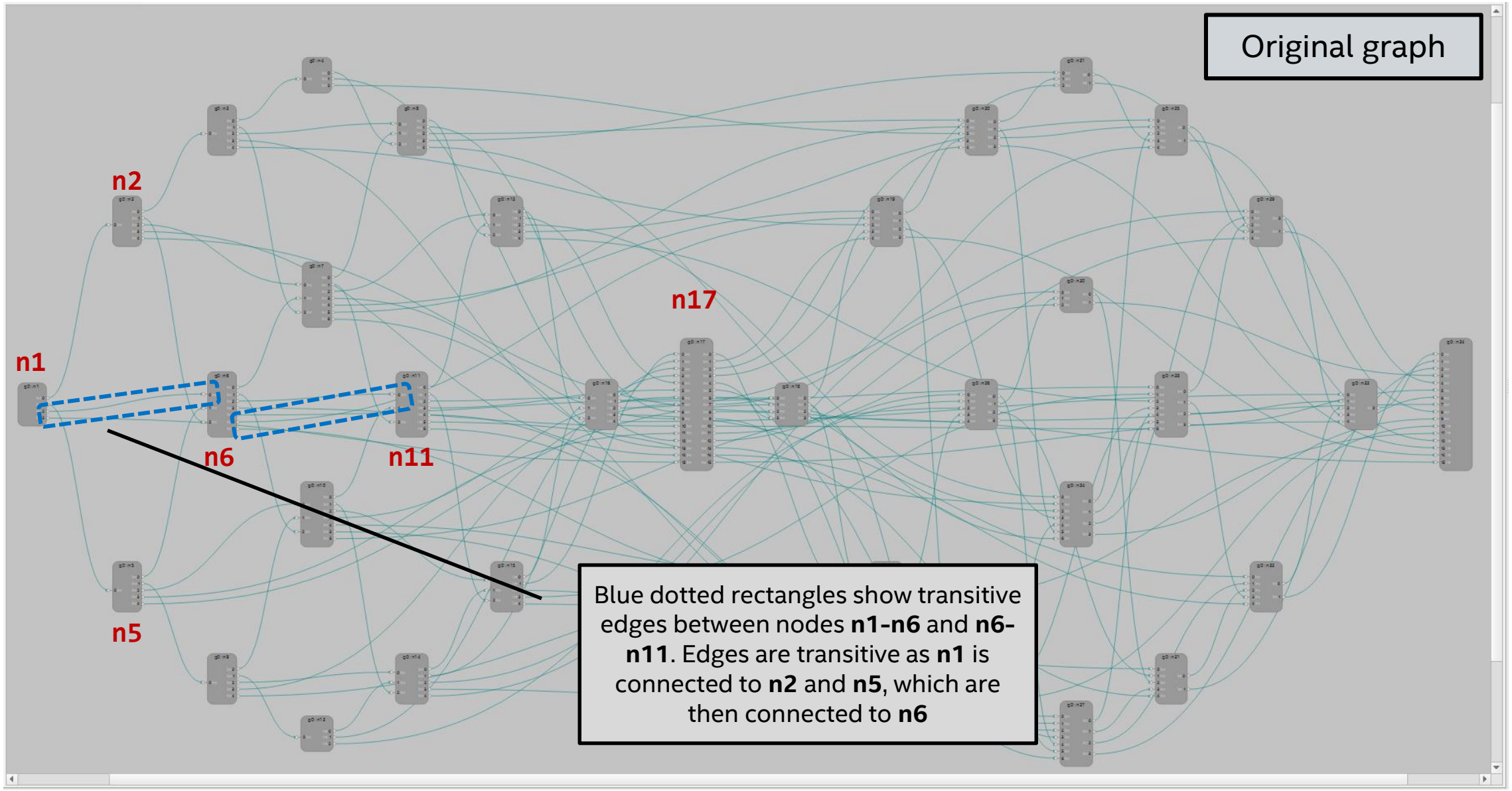
```
1  pipeline_time = prk_wtime();
2  int lic = (m/mc-1) * mc + 1;
3  int ljc = (n/nc-1) * nc + 1;
4  for (int iter = 0; iter<=iterations; iter++) {
5      for (int i=1; i<m; i+=mc) {
6          for (int j=1; j<n; j+=nc) {
7  #pragma omp task depend(in:grid[0],grid[(i-mc)*n+j],\
8                                grid[i*n+(j-nc)],\
9                                grid[(i-mc)*n+(j-nc)]) \
10                       depend(out:grid[i*n+j])
11                 sweep_tile(i, MIN(m,i+mc), j, \
12                                MIN(n,j+nc), n, grid);
13         }
14     }
15 #pragma omp taskwait
16     grid[0*n+0] = -grid[(m-1)*n+(n-1)];
17 }
18 pipeline_time = prk_wtime() - pipeline_time;
```

Task: n17

Example: Synch_p2p from the Parallel Research Kernels

Complete code can be accessed at: https://github.com/ParRes/Kernels/blob/master/C1z/p2p-tasks-openmp.c

Synch_p2p: Task dependence graph in FGA:



NOTE: OpenMP* task dependence support is not yet released in FGA

Optimization 2: Removing redundant dependency

```
1  pipeline_time = prk_wtime();
2  int lic = (m/mc-1) * mc + 1;
3  int ljc = (n/nc-1) * nc + 1;
4  for (int iter = 0; iter<=iterations; iter++) {
5      for (int i=1; i<m; i+=mc) {
6          for (int j=1; j<n; j+=nc) {
7  #pragma omp task depend(in:grid[0],grid[(i-mc)*n+j],\
8                                grid[i*n+(j-nc)],\
9                                grid[(i-mc)*n+(j-nc)]) \
10                       depend(out:grid[i*n+j])
11                 sweep_tile(i, MIN(m,i+mc), j, \
12                                MIN(n,j+nc), n, grid);
13         }
14     }
```

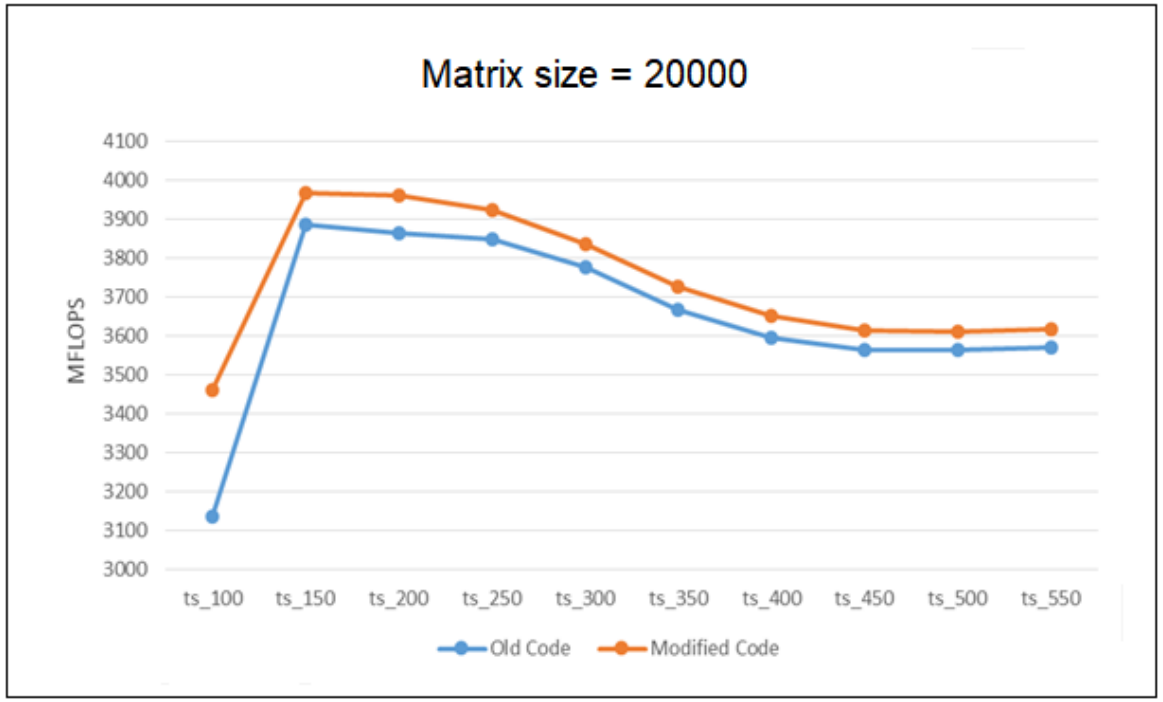
Example: Synch_p2p from the Parallel Research Kernels

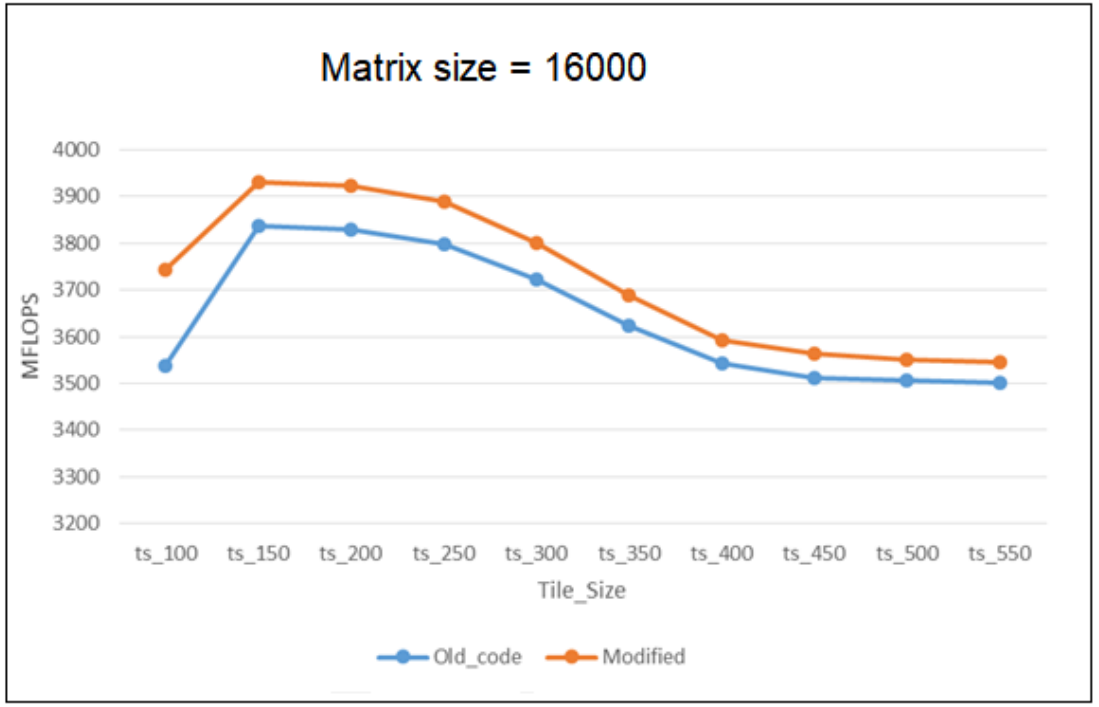
Complete code can be accessed at: https://github.com/ParRes/Kernels/blob/master/C1z/p2p-tasks-openmp.c

Modified code with performance optimization:

```
1  pipeline_time = prk_wtime();
2  for (int iter = 0; iter<=iterations; iter++) {
3      for (int i=1; i<m; i+=mc) {
4          for (int j=1; j<n; j+=nc) {
5  #pragma omp task depend(in:grid[0],grid[(i-mc)*n+j],\
6                                 grid[i*n+(j-nc)]),\
7                  depend(out:grid[i*n+j])
8              sweep_tile(i, MIN(m,i+mc), j, \
9                             MIN(n,j+nc), n, grid);
10         }
11     }
12 #pragma omp taskwait
13     grid[0*n+0] = -grid[(m-1)*n+(n-1)];
14 }
15 pipeline_time = prk_wtime() - pipeline_time;
```

Performance Charts[1]:



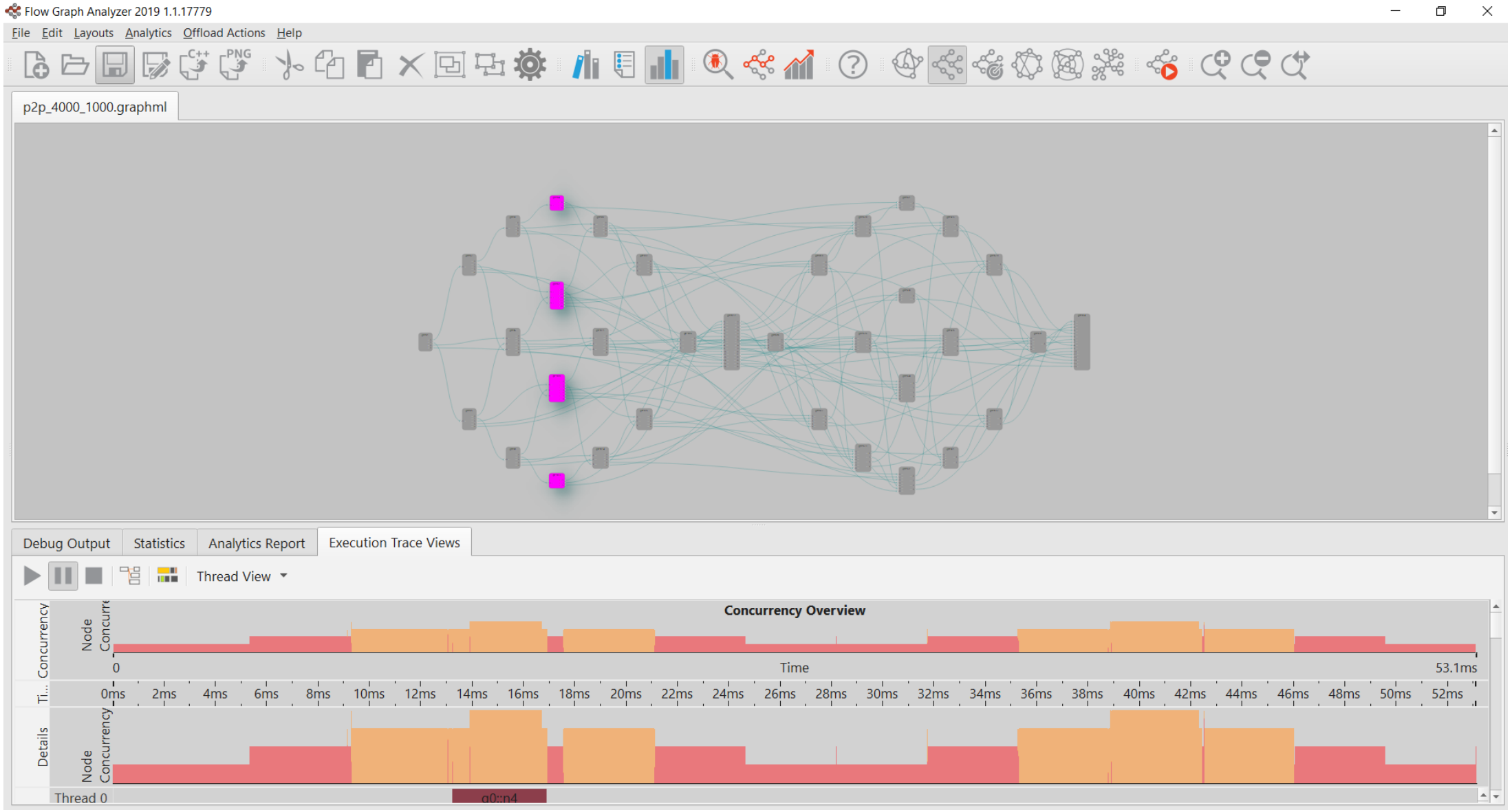


Application was executed on Intel Xeon 6140 processor 18 cores per socket, 2 sockets, with Linux* SLES12 4.4 and Intel composer XE version 18.0.0.128, 64GB RAM.

[1]Performance results are based on testing as of **05/25/2018** and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

NOTE: OpenMP* task dependence support is not yet released in FGA

Play-back of the execution trace



NOTE: OpenMP* task dependence support is not yet released in FGA

Future work

- Intel® Advisor -- Flow Graph Analyzer
  - Adding layers of hierarchy to address visualization of larger graphs.
  - Display line no. from source for its respective tasks and dependencies.
  - Categorize edges through coloring or labels to communicate dependency types and storage locations.
  - Help to identify when a “taskwait” directive may be more efficient than using a barrier task with many dependencies to previously generated tasks.

NOTE: OpenMP* task dependence support is not yet released in FGA

Summary:

- FGA provides powerful features for design and analysis of computational graphs.
- It includes useful correctness and performance capabilities for analyzing task-based applications.
- With ability to visualize graphs, users can correctly express and tune complex constraints.

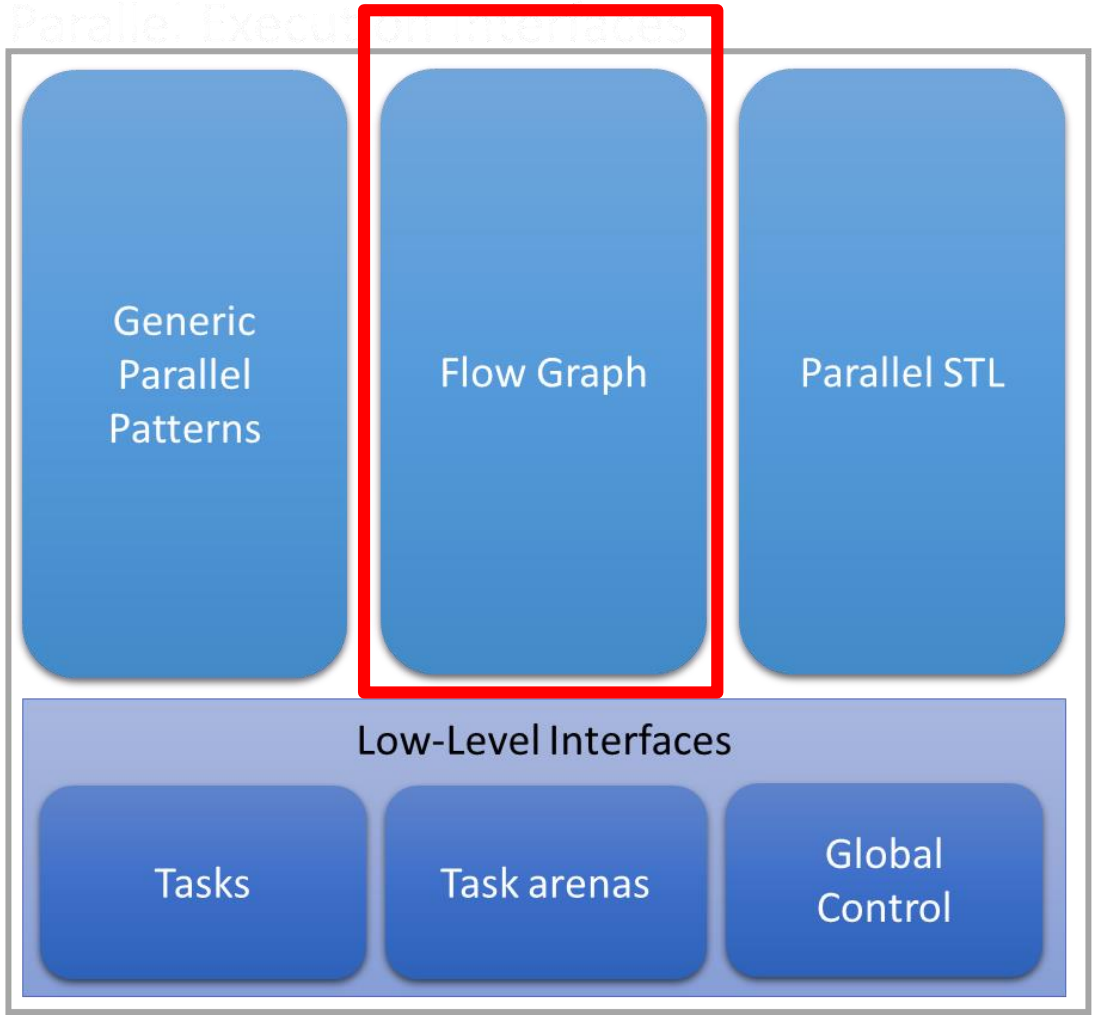
Questions?

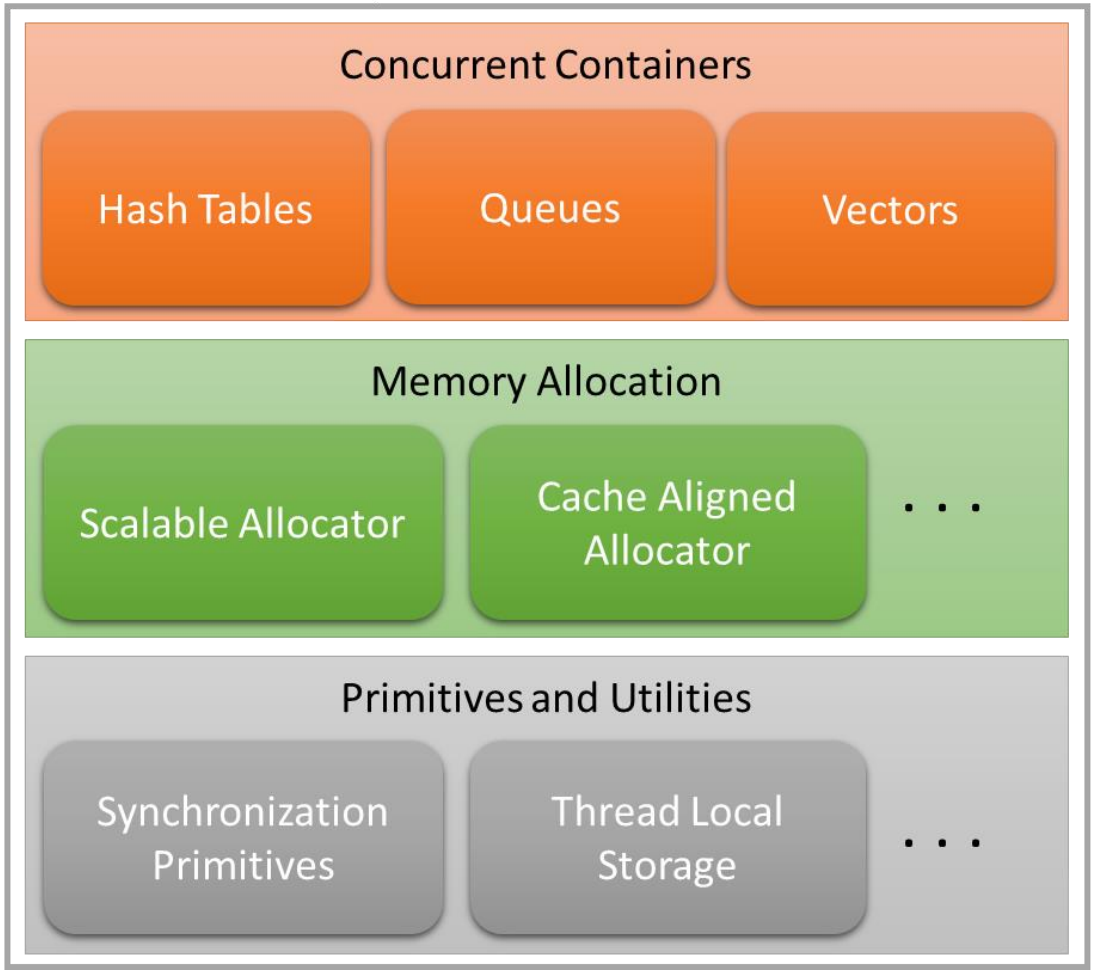
BACKUP SLIDES

Threading Building Blocks (TBB)

Celebrating its 12 year anniversary in 2018!

threadingbuildingblocks.org





OpenMP*

- Standard API for writing multi-threaded application in C, C++ and Fortran
- Introduced support for tasking with OpenMP3.0
- Added support for dependencies with OpenMP4.0
- Recently added runtime interface OMPT that supports gathering performance and correctness trace data.
- **https://www.openmp.org/**

NOTICES AND DISCLAIMERS

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration.

No computer system can be absolutely secure.

Intel, the Intel logo, Xeon, and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. For more complete information about performance and benchmark results, visit http://www.intel.com/benchmarks .

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/benchmarks .

Intel® Advanced Vector Extensions (Intel® AVX)* provides higher throughput to certain processor operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you can learn more at http://www.intel.com/go/turbo.

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.
Intel, the Intel logo, and Intel Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as property of others.