Profiling Multithreaded RDF: NUMA Aware Parallelism

Ivan Kabadzhov

ROOT Data Analysis Framework https://root.cern

Outline

- 1. Understanding the parallelism of Multithreaded RDF
 - a. Slots, Tasks, Threads, TBB
- 2. Dashboards of Multithreaded RDF
 - a. Per Slot, Per Thread, Task duration distribution
- 3. NUMA Architectures and TBB solutions
- 4. Using TBB API to pin tasks to NUMA domains + Results
- 5. Comparison with the **TNUMAExecutor** implementation

Multithreaded RDF



- Each thread executes multiple of tasks (by default proportional to number of threads) \rightarrow granularity
- Create a slot for each thread each task reads/writes data allocated for a certain slot; different tasks of the same thread can be done in different slots
- A thread might start in core X, run in core Y, end in core Z

Patch <u>RLoopManager::RunTreeProcessorMT()</u>. Answers:

- 1. are there long tails of execution?
- 2. are there gaps between tasks?

Some of the plots suffer interesting problems. But why?





Time Elapsed (s)



AMD EPYC 7302 16-Core Processor



9

AMD EPYC 7302 16-Core Processor

				0.0170	0.3370	0.0	2 /0	0.0270	0.02 /0	0.5070	0.0570	
0.65%	0.65%	0.65%	C	0.65% (.63%	0.62%	0.5	7% 0.60	1%	0.65%	0.62%	0.64%
0.57%	0.54%	0.63%	0.65%		0.65%	0.65%	0.65%		0.65%	0.659	<mark>%</mark> 0.65%	
0.65%	0.65%	0.65	%	0.65%	0.65%	0.65%		0.65%	0.62%	0.629	ó 0.62	%
0.63%	0.63%	0.63%		0.63%	0.62%	0.61%		0.63%	0.58%	0.64%	6 0.59	%
0.58%	0.62%	0.64	%	0.65%	0.65%	0.65	%	0.65%	0.62%	0.62	<mark>%</mark> 0.6:	2%
0.58%	0.61%	0.6	3%	0.65%	0.65%		0.65%	0.64	%	0.63%	0.65%	
0.58%	0.62%	c	0.62%	0.62%	0.62	%	0.62%	6 0.62	%	0.57%	0.65%	0.64%
0.65%	0.64%	0.6	55%	0.64%	0.59%	0.	.62%	0.64%	C	.60%	0.64%	0.63%
0.62%	0.62%	0.579	%	0.59%	0.56%	0.629	%	0.62%	0.63%	0.58%	<mark>%</mark> 0.	64%
0.64%	0.64%	0.649	%	0.64%	0.64%	0.65	%	0.65%	0.6	5% 0	.58%	0.62%
0.61%	0.61%	0.56%	0.	<mark>60%</mark> 0.	56%	0.62%		0.63%	0.63%	0.62%	0.63	%
0.62%	0.63%	0.639	%	0.62%	0.62%	0.59%		0.59%	0.58%	0.60%	0.64	%
0.65%	0.65%	0.64	4%	0.65%	0.62%	0.58%	6	0.61%	0.65%	0.65	<mark>% 0.6</mark>	2%
0.64%	0.64%	0.6	4%	0.64%	0.64%	0	.64%	0.64%	0.6	54%	0.59%	0.64%
0.64%	0.65%	0.	65%	0.65%	0.65%	10	0.63%	0.649	%	0.64%	0.60%	
	0.65% 0.57% 0.65% 0.58% 0.58% 0.58% 0.65% 0.62% 0.61% 0.62% 0.65% 0.65% 0.64%	0.65% 0.65% 0.57% 0.54% 0.65% 0.65% 0.63% 0.63% 0.58% 0.62% 0.58% 0.62% 0.58% 0.62% 0.55% 0.64% 0.62% 0.62% 0.62% 0.62% 0.62% 0.62% 0.62% 0.62% 0.62% 0.64% 0.61% 0.64% 0.65% 0.65% 0.65% 0.65% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64%	0.65% 0.65% 0.63% 0.57% 0.54% 0.63% 0.65% 0.65% 0.63% 0.63% 0.63% 0.63% 0.63% 0.63% 0.63% 0.58% 0.62% 0.64% 0.58% 0.61% 0.6 0.58% 0.62% 0.6 0.58% 0.62% 0.6 0.65% 0.64% 0.6 0.62% 0.57 0.64% 0.64 0.61% 0.61% 0.56% 0.64 0.61% 0.61% 0.56% 0.63% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.6 0.64% 0.6	0.65% 0.65% 0.63% 0 0.57% 0.54% 0.63% 0 0.65% 0.65% 0.63% 0 0.63% 0.63% 0.63% 0 0.58% 0.62% 0.64% 0 0.58% 0.61% 0.63% 0 0.58% 0.61% 0.63% 0 0.58% 0.62% 0.62% 0 0.65% 0.64% 0.65% 0 0.62% 0.62% 0.57% 0 0.64% 0.64% 0.64% 0 0.61% 0.61% 0.56% 0 0.62% 0.63% 0.63% 0 0.61% 0.61% 0.56% 0 0.62% 0.63% 0.63% 0 0.62% 0.63% 0.63% 0 0.65% 0.64% 0.64% 0 0.64% 0.64% 0.64% 0.64% 0.64% 0.65% 0.65% 0 <td>0.65% 0.65% 0.65% 0.65% 0.65% 0.57% 0.54% 0.63% 0.65% 0.65% 0.63% 0.65% 0.65% 0.65% 0.65% 0.63% 0.63% 0.63% 0.63% 0.65% 0.63% 0.63% 0.63% 0.63% 0.63% 0.58% 0.62% 0.64% 0.65% 0.58% 0.61% 0.62% 0.62% 0.55% 0.64% 0.65% 0.64% 0.65% 0.64% 0.64% 0.64% 0.62% 0.57% 0.59% 0.64% 0.64% 0.64% 0.64% 0.64% 0.61% 0.61% 0.56% 0.62% 0.61% 0.61% 0.63% 0.62% 0.62% 0.63% 0.63% 0.62% 0.61% 0.63% 0.64% 0.64% 0.65% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% <t< td=""><td>0.65% 0.65% 0.65% 0.65% 0.65% 0.57% 0.54% 0.63% 0.65% 0.65% 0.65% 0.65% 0.65% 0.65% 0.65% 0.63% 0.63% 0.65% 0.65% 0.65% 0.63% 0.63% 0.63% 0.65% 0.65% 0.58% 0.62% 0.64% 0.65% 0.65% 0.58% 0.61% 0.63% 0.65% 0.65% 0.58% 0.62% 0.62% 0.62% 0.62 0.58% 0.62% 0.62% 0.62% 0.62 0.65% 0.64% 0.64% 0.59% 0.56% 0.62% 0.57% 0.59% 0.56% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.61% 0.64% 0.64% 0.64% 0.64% 0.61% 0.63% 0.62% 0.62% 0.62% 0.62% 0.63% 0.62% 0.62% 0.62% 0.65%</td><td>0.65% 0.65% 0.65% 0.65% 0.63% 0.65% 0.62% <th< td=""><td>0.65% 0.62% <th< td=""><td>0.65% 0.65% 0.65% 0.65% 0.62% 0.57% 0.57% 0.57% 0.57% 0.57% 0.65% 0.66% <th< td=""><td>0.65% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.58% 0.62% 0.62% 0.65% 0.65% 0.65% 0.62% 0.58% 0.61% 0.63% 0.65% 0.65% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.63% 0.65% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64%</td></th<></td></th<></td></th<></td></t<><td>0.65% 0.62% <th< td=""><td>0.85% 0.85% 0.85% 0.83% 0.82% 0.57% 0.60% 0.85% 0.85% 0.82% 0.57% 0.85% <th< td=""></th<></td></th<></td></td>	0.65% 0.65% 0.65% 0.65% 0.65% 0.57% 0.54% 0.63% 0.65% 0.65% 0.63% 0.65% 0.65% 0.65% 0.65% 0.63% 0.63% 0.63% 0.63% 0.65% 0.63% 0.63% 0.63% 0.63% 0.63% 0.58% 0.62% 0.64% 0.65% 0.58% 0.61% 0.62% 0.62% 0.55% 0.64% 0.65% 0.64% 0.65% 0.64% 0.64% 0.64% 0.62% 0.57% 0.59% 0.64% 0.64% 0.64% 0.64% 0.64% 0.61% 0.61% 0.56% 0.62% 0.61% 0.61% 0.63% 0.62% 0.62% 0.63% 0.63% 0.62% 0.61% 0.63% 0.64% 0.64% 0.65% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% <t< td=""><td>0.65% 0.65% 0.65% 0.65% 0.65% 0.57% 0.54% 0.63% 0.65% 0.65% 0.65% 0.65% 0.65% 0.65% 0.65% 0.63% 0.63% 0.65% 0.65% 0.65% 0.63% 0.63% 0.63% 0.65% 0.65% 0.58% 0.62% 0.64% 0.65% 0.65% 0.58% 0.61% 0.63% 0.65% 0.65% 0.58% 0.62% 0.62% 0.62% 0.62 0.58% 0.62% 0.62% 0.62% 0.62 0.65% 0.64% 0.64% 0.59% 0.56% 0.62% 0.57% 0.59% 0.56% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.61% 0.64% 0.64% 0.64% 0.64% 0.61% 0.63% 0.62% 0.62% 0.62% 0.62% 0.63% 0.62% 0.62% 0.62% 0.65%</td><td>0.65% 0.65% 0.65% 0.65% 0.63% 0.65% 0.62% <th< td=""><td>0.65% 0.62% <th< td=""><td>0.65% 0.65% 0.65% 0.65% 0.62% 0.57% 0.57% 0.57% 0.57% 0.57% 0.65% 0.66% <th< td=""><td>0.65% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.58% 0.62% 0.62% 0.65% 0.65% 0.65% 0.62% 0.58% 0.61% 0.63% 0.65% 0.65% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.63% 0.65% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64%</td></th<></td></th<></td></th<></td></t<> <td>0.65% 0.62% <th< td=""><td>0.85% 0.85% 0.85% 0.83% 0.82% 0.57% 0.60% 0.85% 0.85% 0.82% 0.57% 0.85% <th< td=""></th<></td></th<></td>	0.65% 0.65% 0.65% 0.65% 0.65% 0.57% 0.54% 0.63% 0.65% 0.65% 0.65% 0.65% 0.65% 0.65% 0.65% 0.63% 0.63% 0.65% 0.65% 0.65% 0.63% 0.63% 0.63% 0.65% 0.65% 0.58% 0.62% 0.64% 0.65% 0.65% 0.58% 0.61% 0.63% 0.65% 0.65% 0.58% 0.62% 0.62% 0.62% 0.62 0.58% 0.62% 0.62% 0.62% 0.62 0.65% 0.64% 0.64% 0.59% 0.56% 0.62% 0.57% 0.59% 0.56% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.61% 0.64% 0.64% 0.64% 0.64% 0.61% 0.63% 0.62% 0.62% 0.62% 0.62% 0.63% 0.62% 0.62% 0.62% 0.65%	0.65% 0.65% 0.65% 0.65% 0.63% 0.65% 0.62% <th< td=""><td>0.65% 0.62% <th< td=""><td>0.65% 0.65% 0.65% 0.65% 0.62% 0.57% 0.57% 0.57% 0.57% 0.57% 0.65% 0.66% <th< td=""><td>0.65% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.58% 0.62% 0.62% 0.65% 0.65% 0.65% 0.62% 0.58% 0.61% 0.63% 0.65% 0.65% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.63% 0.65% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64%</td></th<></td></th<></td></th<>	0.65% 0.62% 0.62% <th< td=""><td>0.65% 0.65% 0.65% 0.65% 0.62% 0.57% 0.57% 0.57% 0.57% 0.57% 0.65% 0.66% <th< td=""><td>0.65% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.58% 0.62% 0.62% 0.65% 0.65% 0.65% 0.62% 0.58% 0.61% 0.63% 0.65% 0.65% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.63% 0.65% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64%</td></th<></td></th<>	0.65% 0.65% 0.65% 0.65% 0.62% 0.57% 0.57% 0.57% 0.57% 0.57% 0.65% 0.66% <th< td=""><td>0.65% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.58% 0.62% 0.62% 0.65% 0.65% 0.65% 0.62% 0.58% 0.61% 0.63% 0.65% 0.65% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.63% 0.65% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64%</td></th<>	0.65% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.62% 0.63% 0.63% 0.63% 0.65% 0.65% 0.65% 0.62% 0.58% 0.62% 0.62% 0.65% 0.65% 0.65% 0.62% 0.58% 0.61% 0.63% 0.65% 0.65% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.62% 0.58% 0.62% 0.62% 0.62% 0.62% 0.62% 0.63% 0.65% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64% 0.64%	0.65% 0.62% 0.62% <th< td=""><td>0.85% 0.85% 0.85% 0.83% 0.82% 0.57% 0.60% 0.85% 0.85% 0.82% 0.57% 0.85% <th< td=""></th<></td></th<>	0.85% 0.85% 0.85% 0.83% 0.82% 0.57% 0.60% 0.85% 0.85% 0.82% 0.57% 0.85% <th< td=""></th<>

Open Data Benchmarks 7 10x files (170GB)

AMD EPYC 7302 16-Core Processor



Granularity Task duration distribution



TBB Major Points

- By default oneTBB does not pin threads to cores.
- By default, oneTBB uses work-stealing and auto-partitioning to balance the load across cores.
- Dynamic nature \Rightarrow good composability overall (<u>source</u>)





Potential NUMA Effects

- Main Thread allocating a lot of memory, then all threads need to access this memory \rightarrow isolation is of no help!
- Thread migrating between different NUMA domains within the execution of the same task
 - If load distribution requires migrating a thread off of a processor \rightarrow the OS pick an arbitrary new processor with sufficient capacity.
 - The newly selected processor should not have higher access costs to the memory → If no free processor matching that criteria, the OS migrates to a processor where memory access is more expensive



TBB NUMA Support (I)

Since TBB 2020 Initial Release, Source: tbb::info::numa_nodes()

- 1. Need to identify the system topology (fails on some machines!)
- 2. Create a vector of task arenas for each NUMA domain, statically split the input space to each domain \Rightarrow each domain can only work on its own partition

```
std::vector<tbb::numa_node_id> numa_nodes = tbb::info::numa_nodes();
std::vector<tbb::task_arena> arenas(numa_nodes.size());
std::vector<tbb::task_group> task_groups(numa_nodes.size());
for (int i = 0; i < numa_nodes.size(); i++) { arenas[i].initialize(tbb::task_arena::constraints(numa_nodes[i])); }
for (int i = 0; i < numa_nodes.size(); i++) {
    arenas[i].execute([&] {
        task_groups[i].run([&] {
            auto lowerBound = start + i * (end - start + size) / size;
            auto upperBound = std::min(start + (i + 1) * (end - start + size) / size, end);
            tbb::parallel_for(lowerBound, upperBound, step, f); }); }); }
for (int i = 0; i < numa_nodes.size(); i++) { arenas[i].execute([&task_groups, i] { task_groups[i].wait(); }); }</pre>
```



TBB NUMA Support (II)

Already present in TBB 2017, <u>Source</u>: affinity_partitioner

It not only automatically chooses the grain size, but also optimizes for cache affinity and tries to distribute the data uniformly among threads. Using affinity_partitioner can significantly improve performance when:

- 1. The computation does a few operations per data access.
- 2. The data acted upon by the loop fits in cache.
- 3. The loop, or a similar loop, is re-executed over the same data.
- 4. The affinity_partitioner object lives between loop iterations. It remembers where iterations of the loop ran, so that each iteration can be **hinted** to the same thread that executed it before.

```
tbb::task_arena arena;
arena.initialize(max_concurrency);
arena.execute([&] {
    static tbb::affinity_partitioner ap; //this is the idea, but does not come out of the box, WIP!
    tbb::parallel_for(start, end, step, f, ap);
});
```

Run Times

AMD EPYC 7302 16-Core Processor





NUMA Task duration distributions



AMD EPYC 7302 16-Core Processor

TNUMAExecutor (old solution)

- Spawn a process for each NUMA domain
- Use numa.h to pin each process to a different NUMA domain
- All threads spawned by a process will be restricted to the same domain
- Execute in each thread (isolated per process)
- Collect results (first from threads, then from processes)

• Problems:

- MP+MT unsafe?
- hard to benchmark: numa.h overwrites the core mask specified by taskset or numactl