As Much Resilience As You Want: A Resilient Legion

Karthik Murthy, Mike Bauer, Kyushick Lee, Yongkee Kwon, Todd Warszawski, Wonchan Lee, Elliott Slaughter, Sean Treichler, Alex Aiken, and Mattan Erez

Stanford University, USA,

Abstract. Resilient is an important aspect of scaling applications on large clusters. As we approach parallelism profiles of several millions and long running applications, we need to ensure that ineffect "we reach the end". Defining the policy interaction with the programming model features necessitates that we revisit the memory concistency model. Recoverability, a critical step in resilience, opens the door to optimizations such as speculation. We also evaluate this. . . .

Keywords: resilience

1 Overview of Resilience in Legion

2 Interaction of Resilience Policy with Legion Features

which are the legion features of importance ?

put a figure of interaction.

what is the memory consistency angle here ?

what does it mean to advance the commit wavefront

Resilience is a tangling of the lifetime of a task and a region snapshot

1) when can we advance a commit wavefront ? 2) whats the lifetime of a region-instance snapshot ?

on this front, we see it as a two-step process a) define a consistent cut of tasks problem, b) commit any task strictly behind this wavefront c) garbage collect any snapshot that serves as input to any task that is already committed

consistent cut of tasks that can be part of the commit wavefront: a set of tasks whose inputs are need_preserve'ed, or they are strictly post-dominated by tasks whose inputs are need_preserved.

3) what about copy/index/tasks ? copy local to local follows the above semantics copy local to remote get committed immediately after successful execution. index launch tasks, actually feel need not be in the task graph, unless virtual mapping is used. they can be garbage collected immediately after all child tasks are included in the dependence analysis wavefront - I am thinking we will never have a case where are relaunching the index launch, since hte child tasks either

are part of the committed wavefront or are not (pending a discussion on phase barriers)

What to do about must_epoch tasks with phase_barrier inside them ?

answer: restartable phase_barrier with generation commit callback

3 Implementation of Resilience

3.1 `need_preserve` Implementation

tagging region instances tagging tasks

-the mapper marks an instance as persistent, i.e., vector¡vector¡bool¿¿ persistent -in finalize_map_task, the singleTask sees this and notes it down in the Individual_task's persistent_tasks list -when trigger_complete is called on the task,

1) inside line 5560, invalidate_region_tree_contexts, inside which we have runtime-¿forest-¿invalidate_versions, we do not do on region[idx].region. we also do not do the instance_top_views

2) we retain the mark on the task as allowed_for_gc.

3) we go to the incoming of this task, and just like verified_regions[true], we mark outgoing_edge_dominated[true] if all the outgoing of a task is marked as true, then we change allowed_for_gc = true.

Discussion Points for today ——————————— 1) allow persistence on a subset of mapped instances in map_task ? Pro: flexibility, Con: if a checkpoint is to be considered useful for a restart, not having the full set of inputs checkpointed seems contradictory. 2) verify_regions tracks op dependencies after physical dependence analysis, correct ? 3) a discussion on persistence inside map_task call

discussed design:

1) build a function set_hardened_instnace(instance,task) along the lines of the set_gc_priority(instance, never, task) inside the mapper call 2) inside that mark a task's incoming edges as saying that it leads to a hardened instance(task) 3) the garbage collector will basically collect a task whose outgoing edges are all marked as verified/hardened. 4) if a task is gc'ed, then it marks all its incoming edges as leads to a hardened instance. 5) steps 1-4 will be based on set_garbage_collection_priority and how verified_regions are set. We will be adding a new list to each task, similar to verified_regions, that will represent edges_ending_in_hardened_regions.

Interaction of need_preserve with the commit wavefront

Obtaining dependence graph in the mapper before calling need_preserve

4 Resilience Application: Speculation

there are three different wavefronts in Legion, we can speculate on any of them.

From a speculation perspective, are they different ? Are we novel, since we have these three different wavefronts ? Can we navigate through this, like the blanks

1) execution wavefront what does it mean to speculate here, does the other steps have to be complete before we do this.

2) mapping wavefront

3) dependence analysis wavefront

– see mike's 6 wavefront answer.

There are three different wavefronts, there could

the 6 wavefronts, where does speculation plays a role mike - speculation is more about tracking the resolution wavefront while resilience is more about tracking the commit wavefront, but the when things go bad, then i think the machinery to restart the mapping and execution wavefronts should be the same

5 Experiments

5.1 Index Tasks

Growth of memory as execution proceeds

Performance overhead as execution proceeds

NumIter	No Resl No lg:res	No Res With lg:res	Res No lg:res
100	7.2	6.6	23.6
200	13.8	13.3	46.3
400	26.1	26.5	94.4
800	55	53	186.8
1000	65	66	239.9

NumIter	No Resilience	Resilience
100	1.12	2.43
200	1.56	4.39
400	3.00	9.45
800	5.58	19.09
1000	7.34	23.2

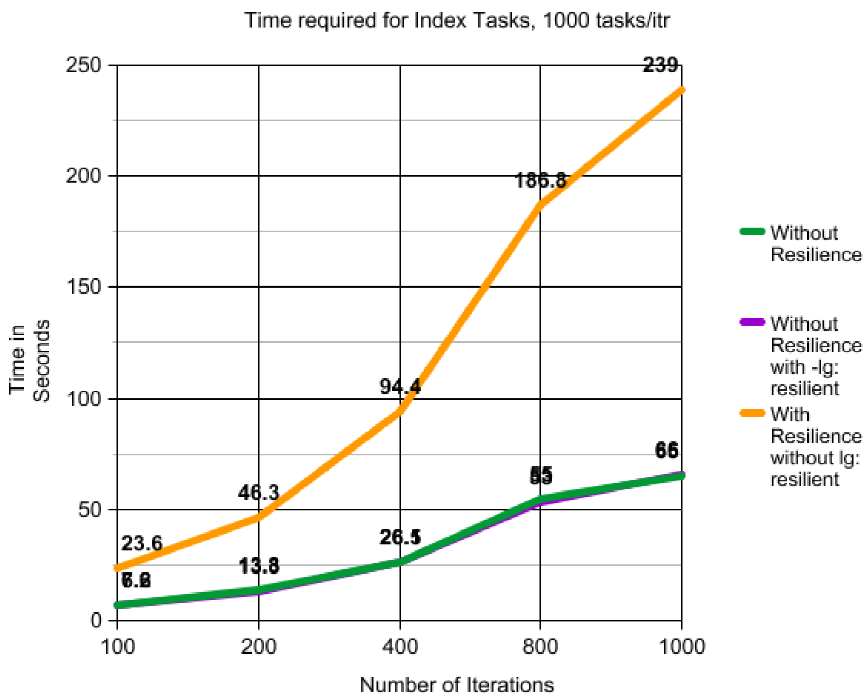

Fig. 1. Total time taken by 02_index_tasks. Time in Seconds, 1000 tasks/index launch

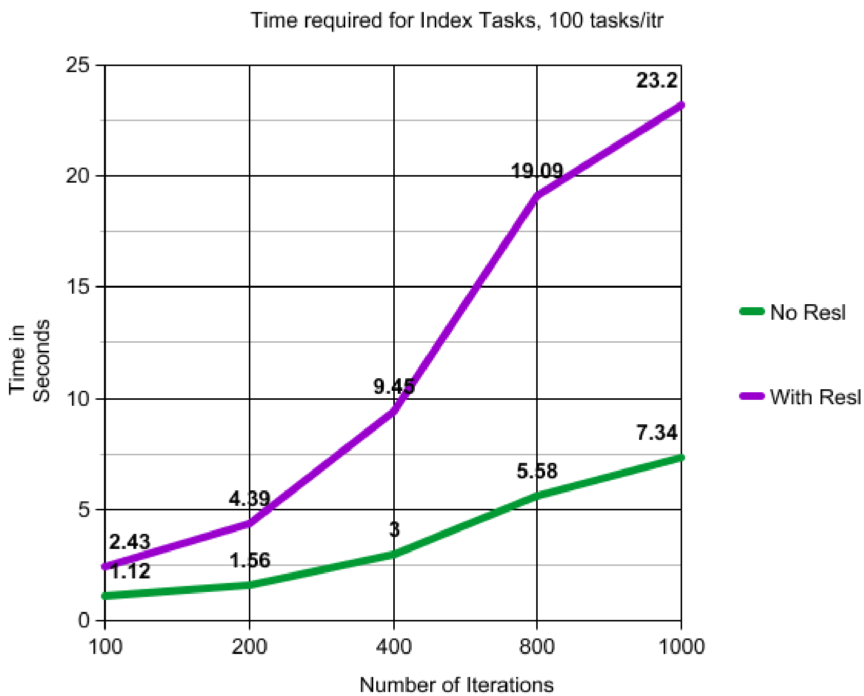

Fig. 2. Total time taken by 02_index_tasks. Time in Seconds, 100 tasks/index launch

NumIter	No Resl No lg:res	No Res With lg:res	Res No lg:res
100	18	140	107
200	22	263	176
400	24	509	245
800	26	1002	428
1000	26	1248	518

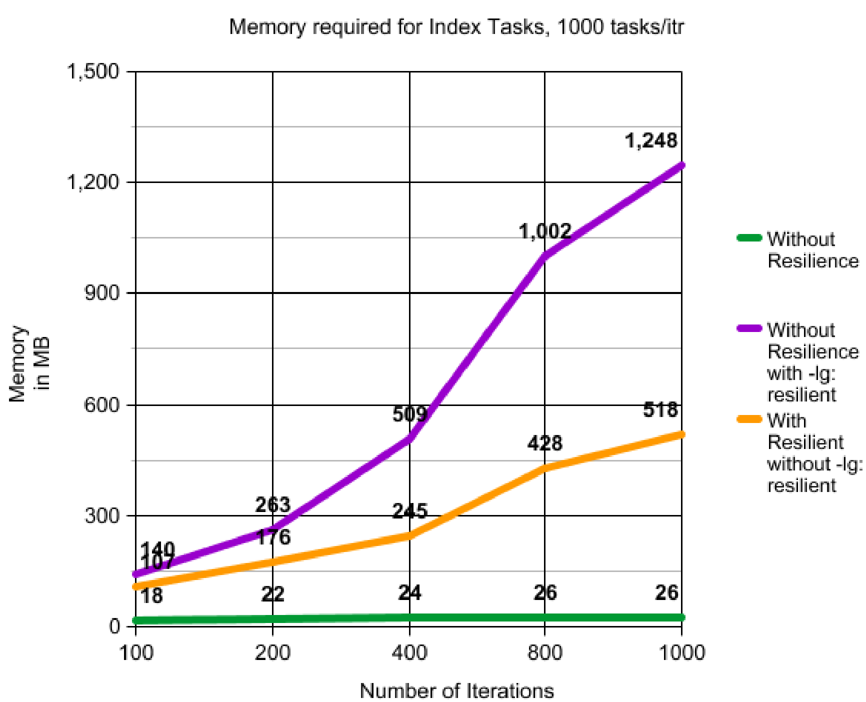

Fig. 3. Total Memory footprint by 02_index_tasks. Memory in MB, 1000 tasks/index launch.

5.2 Stencil

5.3 local recover vs global recovery

5.4 compute/comm vs no-failure/single failure/multi-failure

5.5 S3D, Pennant, Stencil, Circuit

5.6 Some Interesting Task Graphs for Recovery

5.7 bigger examples

pennant, stencil, miniAero (better understood) /Circuit, - limit the failure to the before

6 Adaptive Resilience

7 Resilience Policy Implemented via Mapper: Example 1 Generic

8 Resilience Policy Implemented via Mapper: Example 2 UT Austin

9 Conclusion

References

1. Clarke, F., Ekeland, I.: Nonlinear oscillations and boundary-value problems for Hamiltonian systems. Arch. Rat. Mech. Anal. 78, 315–333 (1982)
2. Rabinowitz, P.: On subharmonic solutions of a Hamiltonian system. Comm. Pure Appl. Math. 33, 609–633 (1980)