Scheduler Scalability

Subhra Mazumdar

Searching for idle cpus

- select_idle_sibling
  - select_idle_core
  - select_idle_cpu
  - select_idle_smt
- select_idle_core can iterate all cpus in LLC domain even if only one idle core is available
- select_idle_cpu tries to find a idle cpu in LLC domain but iterates only ‘nr’ cpus
  - nr is determined by average idle time of cpu and avg cost of scanning LLC domain
  - Has arbitrary fuzz factor
  - Can end up scanning the entire socket

select_idle_cpu

- Hard to find any formula that is dynamic and yet works well for all cases
  - Better to put a upper bound on the scan (‘nr’)
- What should the bounds be?
  - Experimented with some upper and lower bounds (on SMT2, SMT8)
  - Upper bound of 2 cores and lower bound of 1 core seems to work well
  - Gives a chance to find an idle cpu outside of the current core for all kinds of cpu id enumerations
- How to avoid localization?
  - Have a per cpu variable to track the search limit
  - Will start searching from there next time

select_idle_core

- Has a dynamic switch to disable scanning but still a bottleneck
- Can we have data structures to do it fast?
  - Scheduler fast path is *very* sensitive
  - Just disabling idle core search improves context switch intensive workload (OLTP)
- Have a new sched feature SIS_CORE for disabling idle core search in run time
- Improves most workloads, regresses some on some some architectures (hackbench on SMT8)

Results: 2 socket, 44 cores, 88 cpus Intel x86 (select_idle_cpu)

Hackbench

Groups	Base	New	%gain
1	0.5816	0.5903	-1.5
2	0.6428	0.5843	9.1
4	1.0152	0.9965	1.84
8	1.8128	1.7921	1.14
16	3.1666	3.1345	1.01
32	5.6084	5.5677	0.73

Uperf pingpong with msg_size=8k

Threads	Base	New	%gain
8	45.36	46.28	2.01
16	87.81	89.67	2.12
32	151.19	153.5	1.53
48	190.2	194.79	2.41
64	190.42	202.9	6.55
128	323.86	343.56	6.08

Oracle DB TPC-C

Users	%gain
20	0.68
40	1.03
60	1.78
80	0.92
100	0.9
120	0.48
140	1.16
160	2.64
180	1.94
200	2.8
220	2.29

Results: 2 socket, 44 cores, 88 cpus Intel x86 (select_idle_cpu + NO_SIS_CORE)

Hackbench

Groups	Base	New	%gain
1	0.5816	0.5835	-0.33
2	0.6428	0.5752	10.52
4	1.0152	0.9946	2.03
8	1.8128	1.7619	2.81
16	3.1666	3.1275	1.23
32	5.6084	5.5856	0.41

Uperf pingpong with msg_size=8k

Threads	Base	New	%gain
8	45.36	46.94	3.48
16	87.81	91.75	4.49
32	151.19	167.74	10.95
48	190.2	200.57	5.45
64	190.42	226.74	19.07
128	323.86	348.12	7.49

Oracle DB TPC-C

Users	%gain
20	0.56
40	1.73
60	-0.05
80	1.75
100	1.51
120	2.44
140	3.4
160	3.62
180	4.1
200	2.33
220	1.25

Avoid scheduling overhead altogether

- Profiling context switch intensive workloads will show few common hot stacks calling schedule()
- pipe_read & pipe_write are among them
  - Have busy waiting mechanism for certain amount of time
  - Networking has similar mechanism already
- Can we have dynamic formula for optimal spin time?
  - Different workloads on different architectures will have different optimum
  - Have a tunable that can be set for specific workloads
- Workloads?
  - Obvious: Hackbench pipe, Unixbench pipe
  - To try: OLTP

Results: 2 socket, 36 cores, 72 cpus Intel x86 (spin=10us)

Hackbench pipe

Groups	Base	New	%gain
1	0.6742	0.6842	-1.48
2	0.7794	0.7116	8.7
4	0.9744	0.8247	15.36
8	2.0382	1.572	22.87
16	5.5712	2.7989	49.76
24	7.9314	3.962	50.05

Unixbench pipe

	Base	New	%gain
1 copy	372.2	772.9	107.7
72 copies	11298.2	17089.9	51.3

Takeaways..

- Optimizing the scheduler for performance is hard
  - Always double edged sword
  - Satisfying all workloads on all architectures at all utilizations
  - Scheduler feature may come to rescue
- Will LLC domain continue to get bigger?
  - Intel has 28 cores per socket
  - AMD has 32 cores BUT is MCM (4 NUMA nodes in a socket)
  - ARM?

Questions?