Raxml repeats benchmarks

April 7, 2017

1 Execution time

We run raxml for each dataset with and without the repeats option, and compare the execution times.

Speedup is calculated as follow :

$$speedup = \frac{time_{tipinner}}{time_{repeats}}$$

When we don't indicate times, the cluster cancelled the jobs before the end (after 24h). We then compare the speedup at the last reached step.

When we write "diverged", the runs diverged. In this case we write the speedup at the last common step.

Raxml commands used (repeats, repeats with jemalloc, tipinner):

```
$ LD_PRELOAD=libjemalloc.so mpirun ./raxml-ng-mpi --seed=42 --msa data.ph
 --simd AVX --threads 1  --search --repeats on
$ LD_PRELOAD=libjemalloc.so mpirun ./raxml-ng-mpi --seed=42 --msa data.ph
 --simd AVX --threads 1  --search
```

dataset	taxas	sites	partitions	type	threads	$time_{reps}$	$time_{ti}$	speedup
404	404	7444	11	DNA	16	457s	755s	1.65
1kite_science	144	371434	50	DNA	16	9351s	13311s	1.42
1kite_hyme	174	2248590	4116	DNA	64	21663s	29971s	1.38
Antl_1_1_nt	40	522173	658	DNA	16	1254s	1668s	1.3
Antl_1_1_aa	40	762438	659	prot	32	4966s	6080s	1.2
para_1_nt	193	1514275	3714	DNA	64	diverged	diverged	1.35

2 Scalability and load balancing

- "speedup" is the speedup between tipinner and repeats.
- "rep" is repeats
- "time lost" is the time lost because of bad load balancing

1kyte_hyme

threads	sites/thread	seq time	seq time rep	time lost	time lost rep	speedup

404

threads	sites/thread	seq time	seq time rep	time lost	time lost rep	speedup
4	1861	7816s	5100s	2.37%	2.83%	1.53
8	930	11608s	6472s	5.14%	4.43%	1.79
16	465	12016s	7392s	5.73%	6.49%	1.62

antl_1_1_nt2

threads	sites/thread	seq time	seq time rep	time lost	time lost rep	speedup
16	32635	26416s	20080s	0.43%	0.90%	1.31
64	8158	25280s	18816s	1.31%	1.65%	1.34
256	2039	31488s	26112s	3.32%	4.89%	1.20

kyte

threads	sites/thread	seq time	seq time rep	time lost	time lost rep	speedup
16	23214	213408s	148688s	1.21%	2.03%	1.43
64	5803	168192s	119616s	2.00%	8.08%	1.40

3 Load balancing

We define

$$busy\ ratio = (1 - \frac{waiting\ time\ in\ reduce\ operations}{elapsed\ time}) * 100$$

It directly gives the speedup we could get from a perfect load balancing.

I will recompute this table later with mpi.