Optimizing Parallel Computations of Gröbner and Janet Bases on SMP-machine

D.A. Yanovich

e-mail: yan@jinr.ru, Laboratory of Information Technologies, JINR, Dubna

In previous papers [1, 2, 3, 4] we presented algorithm for parallel calculation of Gröbner and Janet bases that works in terms of parallel normal forms computations. The realization was quite promising but faced problem of "starvation" (e.g. in some moments of time only few processors was fully loaded).

After several performance tests on eight-core SMP machine we faced low practical scalability. To reveal "bottleneck" we tried to use several third-party thread profilers, but none of them provide sufficient information. Thus, custom thread profiler was written.

Brief specifications of the profiler:

- Size: around 30 lines of C code in main programm, 1.5Kb Python script to parse and plot data
- Speed impact: one fork() syscall, +1% CPU load when profiling eight threads

Profiling revealed a low scalability – "starvation". At some moment only one CPU core is fully loaded and others are idle (Fig. 1).

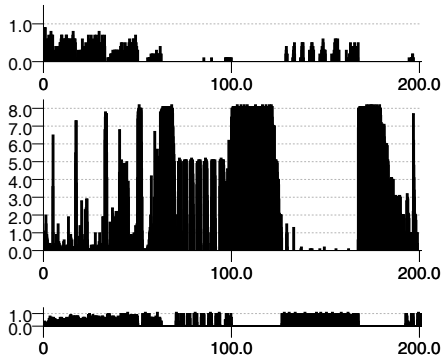

Figure 1: CPU load when computing noon8

After that two ways of lowering effect of starvation tried. In general, we can break most time consuming and lengthy operation – lead normal form computation – to shorter tasks. Results of both approaches shown in table below where the standard benchmarks were used (cf. http:/invo.jinr.ru). In first case reduction of one polynomial by other will be shorter task (NF column in table). In second case we will break reductions of any poly to term reductions level (RD column).

As one can see, a more complicated approach is not always better. It turned out that overhead in the case of term reduction is too large to provide a good scalability.

After optimizations in good cases we get the load graph shown in Fig.2. It shows that machine is

Example	1 thr	NF	up	RD	up
cyclic8	4050.68	1231.01	3.29	1280.64	3.16
discret3	6491.38	1079.31	6.01	1079.24	6.01
eco11	714.53	146.86	4.87	166.68	4.29
eco12	7808.16	1287.37	6.07	1860.08	4.20
extcyc6	194.17	37.88	5.13	40.51	4.79
hcyclic8	2136.40	1230.01	1.74	1233.00	1.73
hf855	544.09	181.16	3.00	188.94	2.88
ilias13	2712.76	564.93	4.80	569.67	4.76
ilias_k_3	157.37	36.10	4.36	37.15	4.24
jcf26	75.65	11.01	6.87	10.93	6.92
katsura10	2119.75	649.08	3.27	681.11	3.11
noon8	336.88	168.94	1.99	206.58	1.63
noon9	23615.50	14590.20	1.62	16594.30	1.42
redcyc7	544.27	115.41	4.72	111.57	4.88
redeco11	110.59	31.08	3.56	33.71	3.28
redeco12	1019.84	261.92	3.89	282.64	3.61
reimer7	879.98	524.12	1.68	518.68	1.70
Overall	54426.13	22460.50	2.42	25216.12	2.16

100% loaded all the time and the scalability coefficient is 6.87, i.e. quite good. But some table examples have poor scalability even after optimization, so we have more work to do.

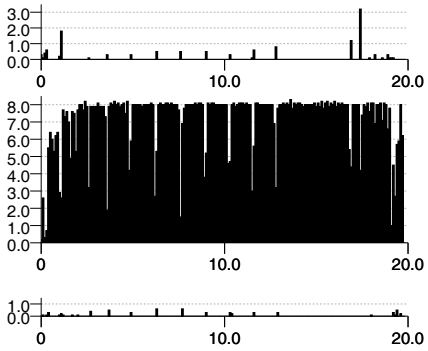

Figure 2: CPU load when computing jcf26

References

[1] V.P. Gerdt, D.A. Yanovich. JINR E5,11-2001-279, Dubna, 2002, pp. 93-103.

[2] D.A. Yanovich. Programming and Computer Software, 2002, 2, pp.66-69.

[3] V.P. Gerdt, D.A. Yanovich. "Computer Algebra in Scientific Computing / CASC 2004", Institute of Informatics, Technical University of Munich, Garching, 2004, pp.185-194.

[4] D.A. Yanovich. Programming and Computer Software, 2008, 4, pp.210-215.