Short-term optimizations of PREP

Tayfun Dalkılıç, Daan Degrauwe

March 11, 2013

1 Introduction

The 2012 SURFEX working week in Brussels resulted in some proposals for short-term and long-term actions to transform PREP into a code that is suitable for operations. In this document we will discuss the ongoing work (4-week stay of Tayfun in Brussels) on the short-term actions:

- OpenMP parallelization of the most expensive parts
- avoidance of NWP-useless computations
- patch averaging

The use of binary LFI files instead of ASCII also resolved the namelist anomaly that was encountered during the WW.

2 Profiling results

2.1 Large domain

We performed a profiling of PREP (cy37t1) on a 500×500 domain on a SGI UV2000 (1TB shared memory, 256 cores) machine:

1	78.59	223.887	223.887	223.887	36	6219.07	6219.08	MODI_AV_PGD:AV_PATCH_PGD_1D_3@1
2	12.27	258.856	34.970	34.970	924	37.85	37.85	MODI_BILIN:BILIN_4@1
3	3.35	268.391	9.535	9.535	66	144.47	144.48	MODI_AV_PGD:AV_PGD_1D_2@1
4	1.81	273.556	5.165	5.166	371	13.92	13.92	MODI_HOR_EXTRAPOL_SURF:HOR_EXTRAPOL_SURF@1
5	1.34	277.388	3.832	3.832	133	28.81	28.81	MODE_GRIDTYPE_CONF_PROJ:XY_CONF_PROJ@1
6	0.22	278.014	0.626	0.626	50	12.52	12.53	MODI_COEF_VER_INTERP_LIN_SURF:COEF_VER_INTERP_
7	0.21	278.615	0.601	5.767	924	0.65	6.24	MODI_BILIN:BILIN_5@1

(to our surprise) **patch averaging remains the most expensive routine, also for moderate to large domains**. We were expecting to see interpolations become relatively more important for such domains.

Apparently, 3 routines take 90% of the runtime: patch averaging, bilinear interpolations, and another type of cover averaging. The good news is that optimizing these routines will significantly improve PREP runtimes. Therefore, we had a look at using OpenMP parallelization of these 3 routines (see infra).

2.2 FP2SX1 versus offline PREP

When converting ISBA to SURFEX, one can either use offline PREP, or run an ee927 configuration with `NFPSURFEX>0`. Applying both of them to the same domain sizes, we get following profiling results:

FP2SX1:

FP2SX1:

1	84.54	109.470	109.470	109.470	15	7297.98	7297.99	MODI_AV_PGD:AV_PATCH_PGD_1D:PART2@1
2	7.32	118.946	9.477	9.477	54	175.49	175.50	MODI_AV_PGD:AV_PGD_1D_2@1
3	0.96	120.189	1.243	5.152	7	177.51	735.99	PREP_HOR_ISBA_FIELD@1
4	0.95	121.425	1.236	1.493	5	247.26	298.52	MODI_READ_SURF:READ_SURFX2COV@1
5	0.83	122.495	1.070	1.851	3	356.64	617.14	SOIL_PROFILE_BUFFER@1

PREP:

```
PREP:
1     63.29       115.211      115.211      115.211              21     5486.23     5486.24    MODI_AV_PGD:AV_PATCH_PGD_1D:PART2@1
2     22.39       155.968       40.758       40.758             924       44.11       44.11    BILIN_2@1
3      5.65       166.253       10.285       10.285              63      163.25      163.26    MODI_AV_PGD:AV_PGD_1D_2@1
4      2.62       171.023        4.769        4.770             371       12.86       12.86    HOR_EXTRAPOL_SURF@1
5      0.88       172.623        1.600        2.497              10      160.04      249.72    MODI_READ_SURF:READ_SURFX2COV@1
6      0.84       174.148        1.525        1.525             133       11.46       11.47    MODE_GRIDTYPE_CONF_PROJ:XY_CONF_PROJ@1
7      0.77       175.551        1.404        1.404             924        1.52        1.52    BILIN_1@1
8      0.47       176.398        0.847        5.617             924        0.92        6.08    BILIN_3@1
```

From these results, we conclude that:

- **interpolations in FullPos are much more efficient** (routine `fpint12` taking 0.075 s) than the PREP interpolations (~ 47 s).
- there is also a small difference (6 s) due to fewer calls to patch averaging (no averaging on departure domain for ISBA scheme). Since the departure domain usually has a relatively low resolution, this doesn't matter too much.

3 Application of OpenMP

We focused on the 3 most expensive routines (`av_pgd_patch_1d`, `bilin` and `av_pgd_1d`), and parallelized the gridpoint loops with OpenMP. This required only minimal code changes on specific hot-spots.

Figure 1 shows the speed-up when using up to 64 processors for different domain sizes.

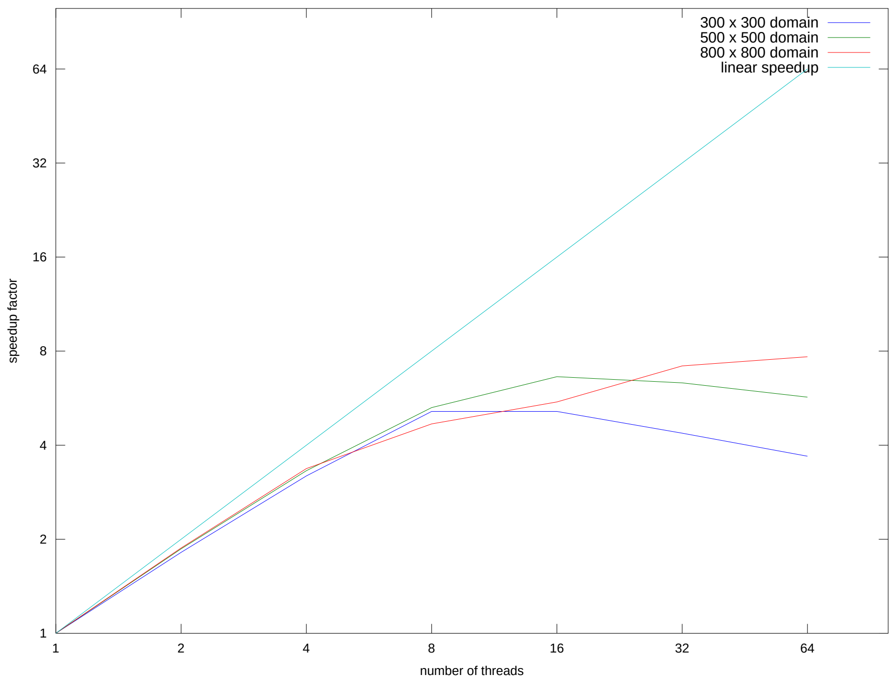

Figure 1: Speed-up of PREP using OpenMP.

Conclusions:

- Up to 8 processors, scaling is quite good
- For the 800×800 domain, which is comparable to the largest domains currently in operational use, the runtime of PREP reduces from 11m (1 thread) to 1m26 (64 threads). **This seems acceptable for operational use.**

- Scaling seems to be better for larger domains

We would like to remark that `bilin` scales nearly perfect (also beyond 64 threads), while the other two lose their scalability beyond 8 threads. However, since we only used the most basic OpenMP directives, we believe that is possible to further increase the scalability of these two routines.

Note that FP2SX1 can also benefit from the introduction of OpenMP in the patch averaging. This was confirmed by profiling.

4 First glance at application of MPI

For large domains, the memory consumption of PREP may become problematic. This cannot be resolved by OpenMP. Therefore, using MPI parallelization should be considered in the longer term.

A first look at the largest arrays points at the `XCOVER` array. For instance, it uses about 3GB of memory for a 800×800 domain. However, since this is a module variable (which also could be used by PGD and OFFLINE), distributing it between several MPI tasks doesn't seem easy.