Parallel processing in GAMS
– how it is used in CAPRI

- Wolfgang Britz, September 2010 -

Background and motivation

Since a few years, increases in CPU speed have slowed downed, while computers with several CPUs have become quite common. Using several CPUs for one task is called parallel processing. So far, the GAMS base engine does not support parallel processing. Especially the farm type layer in CAPRI requires a lot of computing power to solve the close to 1.900 independent supply models and calculate indicators from their results. As many users now have access at least 4 core desktops or even to 8 core computing servers, it seems appropriate to analyse and test the feasibility of parallel processes.

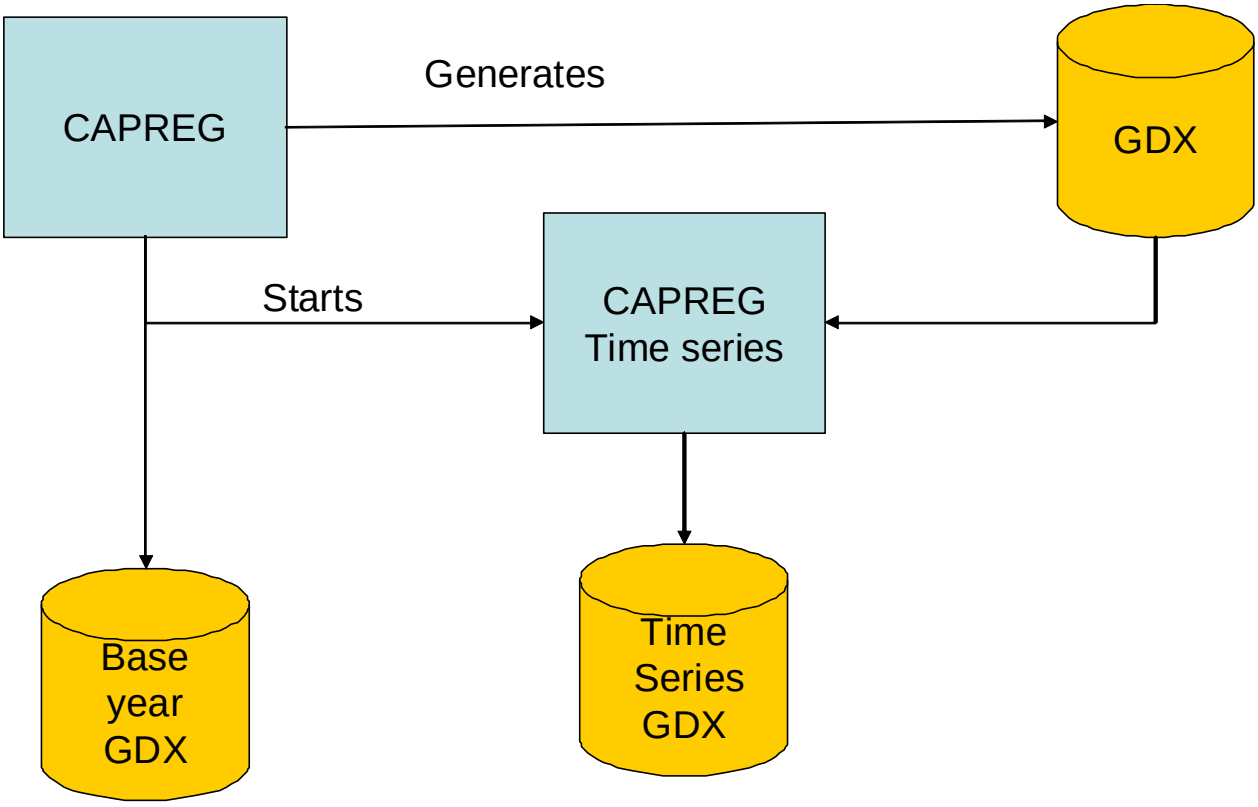
Case 1: two GAMS processes, not output collection

That case is now widely used in CAPMOD, but also in CAPREG. It is e.g. integrated in CAPREG which generates two outputs. Firstly, a GDX file with time series results. There is a huge block of calculations, such as aggregating from single products to activities to groups, where results are not further used in CAPREG itself. Instead, there are only of interest to the user. These calculations are now performed in a separate GAMS program termed CAPREG_TIME_SERIES. It is started by CAPREG by the following sequence:

```
*
*     --- unlaod the yearly results temporarily to a gdx
*
      execute_unload "%SCRDIR%\capreg_temp.gdx" DATA,DATA_RAW,DATA_TRD,META;
*
*     --- generate a flag file which will be deleted if the child process is done
*
      execute "type test > %scrdir%\capreg_time_series.flag"
*
*     --- make sure the test output gdx from capreg_time_series.gms is deleted
*
      execute "del %scrdir%\capreg_time_series.gdx > nul";
*
*     --- start process which calcaluates indicators over time and stores them to GDX
*
      execute 'start /B /HIGH %GAMSEXE% %CURDIR%\capreg_time_series.gms -workdir=%CURDIR%';
```

The first statement (execute_unload) stores all time series results from CAPREG into a GDX files to be read by the second process. Next, it generates a small file on disk which allows checking later if the second process has terminated. The third statement deletes a test output

to be generated by the second process. Should the process not compile without error or terminate at run time, it will not generate that file, and the “mother” process CAPREG can raise an error. Finally, a separate GAMS process is started. The process is also depicts in the following diagram.



The following code piece shows the final sequence to make sure that the second process terminated successfully before CAPREG is started for the next country:

```
*  --- wait until the process calculating and storing the yearly results
*      has finished

$batinclude 'util\title.gms' "'Wait for time series calculation child process to finish'"

    execute "=%curdir%\util\taskSync.bat 1 240 %scrdir%\capreg_time_series.flag"

    scalar p_done;
    execute_load "%scrdir%\capreg_time_series.gdx" p_done;
```

“TaskSunc.bat” is a very simple command program which runs a loop in which it checks if a file or group files exists. If the file(s) do not exist(s), it returns. It will do so for a maximal time (in our case for 4 minutes). The final check consists in try to load a scalar from the test GDX file.

A similar, easier application is applied in CAPMOD for the generation of the iteration log:

```
$iftheni %STEPREPORT% == on
          execute_unload "%SCRDIR%\cur_step.gdx" ITER_CHG,PRODS,UVAN,LEVLS,FEEDS,PREMS,TIMELAP,CURSTEP;
          execute 'start /B %GAMSEXE% %CURDIR%\step.gms --STEP=STEP  -workdir=%CURDIR%';
$endif
```

In that case, the results are only for controlling at run time – it is not necessary to break the master process if an error in the child process occurs.

Case 2: Independent GAMS processes for model generation

The third case combines the two cases above, and adds some more complexity. It is used to solve the many supply models from the farm type layer. It consists of three major elements:

1. A loop which generates a separate GAMS process for a range of region / farm types belonging for one country in the main CAPMOD process (see supply\simu_supply_grid.gms). Each region is called cluster in the following.
2. Cluster specific GAMS processes: they solve the individual model instances for each region / farm types and collect their result. The results are stored in a GDX file.
3. A collection loop in the main CAPMOD process which collects the solutions from these GDX-files.

As a first step, the clusters are defined (see below). The “genModel” (for generateModels) set stores the lists of cluster of regions to generate. To each cluster, a country is attached (genModel_to_MS) as well as a list of regions / farm types (genModel_to_RU). The cluster will comprise up to 20 NUTS2 or up to 30 farm types. The differentiation is introduced as (1) the farm type models comprise typically fewer non-zero activities, and (2) the processes need to load larger data sets in case of the farm types.

```
   if ( card(MS_LARGE) and (card(genModel) eq 0),

       option kill=RUNR;
       option kill=MSACT
       option kill=curGenModel;
       genModelPos = 1;
       curGenModel(allGenModels) $ (allGenModels.pos eq genModelPos) = YES;
       genModel(curGenModel) = YES;

       LOOP(RU_LARGE(RU),
           RUNR(RU) = YES;
           genModel(curGenModel) = YES;

           if ( SUM(RU_MS(RU,MSACT),1) eq 0,
              option kill=MSACT;
              MSACT(MS) $ RU_MS(RU,MS) = YES;
           );
*
*          -- if the next region is in another Member State,
*             the current one is the last one
*
           lstInMemberState(RU)
              $ (sum(RU1 $ ((ru1.pos eq ru.pos+1) and (not SUM(RU_MS(RU1,MSACT),1))),1)) = YES;

           genModel_to_MS(curGenModel,MSACT)  = YES;
           genModel_to_RU(curGenModel,RU)     = YES;
*
*          --- regions in current cluster
*
           if ( (card(RUNR) gt MIN(20 + 30 $ (NTSLVL eq 999),CARD(RU)/%noProc%)) or (lstInMemberState(RU)),
              option kill=RUNR;
              option kill=curGenModel;
              genModelPos = genModelPos + 1;
              curGenModel(allGenModels) $ (allGenModels.pos eq genModelPos) = YES;
           );
      );
   );
```

Next, we generate the process directories and define GDX files comprising the definition set for each model. That step is necessary only once. The MS-specific directories used in the reporting part are also generated here:

```
  if ( card(MS_LARGE),
*
*    --- delete old flag and result files
*
     execute "if exist %scrdir%\supply_model*.flag del %scrdir%\supply_model*.flag";
     execute 'if exist %scrdir%\MSTypes_*.gdx del %scrdir%\MSTypes_*.gdx';
*
*    --- generate a scratch directory for each MS, and if existing, erase content
*        (could be a leftover from an old, unsuccessful or manually deleted run)
*
     if ( first_check_for_dirs eq 0,

        LOOP(MS,
            put_utility 'shell' / "if not exist %scrdir%\"MS.tl:2" mkdir %scrdir%\"MS.tl:2;
        );
*
        LOOP(genModel,
            put_utility 'shell' / "if not exist %scrdir%\"genModel.tl:4" mkdir %scrdir%\"genModel.tl:4;
        );

*
*       --- generate the regional domain set (ALIS for RALL) for the separate model generation
*           processes
*
        loop(genModel_to_MS(genModel,MS),

           option kill=RUNR;
           RUNR(MS) = YES;
           RUNR(RU) $ genModel_to_RU(genModel,RU) = YES;

           put_utilities batch 'gdxout' / '%scrdir%\RCUR_' genModel.tl:4;
           execute_unload RUNR=RCUR;
        );

*
*       --- set the flag to true: we need that piece of code only once
*
        first_check_for_dirs = 1;
     );
```

The next step generates a GDX comprising all symbols used in the supply model:

```
execute_unload '%scrdir%\to_simu_supply.gdx'  RU_MS,RU,data,TECHF,AREQM,DAYS,TrimFeed,MAXSHR,MINSHR,EMILOSS,NAVFAC,
                                              LEVL,FEDNG,NETTRD,YANUSE,FEDUSE,FERTDIST,TotMannNPK,Surplus,v_corfSetr,OBJE,
                                              QUADF,NAVFACC,NUTFAC,NUTFACC,LINEAR,QUADRA,QUADRF,QUADL,landSupCost,v_overShotEntl,v_sumEntl,
                                              v_nonfSlack,SigmSugb,SalesSugb,cdfSugb,pdfSugb,LOSSES,RUNR_OMOBJE,RUNR_OMS_IN_SUPBAL,OBJC,
                                              v_labCap,NFact,MINAN,PMPB,PMPA,PMPD,MXSETA,PSDPAY_T_A,PPENTL,PMPFDG,PMPL,PMPTAIL,MINAN_CORR;
```

As in the case of CAPREG; that GDX will be inputted by the “child” processes (generate_supply_model.gms). In order reduce processing time, the child process does not use the regular “RALL” definition where all regions are comprised, but defines its own “RALL” as an alias to the regions it handles loaded from a cluster specific GDX file:

```
Set RCUR;
$GDXIN %scrdir%\RCUR_%genModel%.gdx
$load RCUR
$GDXIN

 ALIAS(RCUR,RALL);

 SET RU(RCUR),RUNR(RCUR),RUNAGG(RCUR);
 SET R_RAGG(RCUR,RCUR),RU_MS(RCUR,RCUR);

 alias(RM,RALL);

 SET MSACT(RCUR);
 SET MS(RCUR);

 SET CUR_Y_BAS / Y,CUR,BAS /;

 PARAMETER DATA(RCUR,COLS,ROWS,CUR_Y_BAS),INFES;
```

The “execute_load” in the child process statements works then like a filter, loading only the part of the symbol referring to that country. Doing so saves memory in the child processes and helps speeding them up:

```
execute_load "%scrdir%\to_simu_supply.gdx" genModel,genModel_to_MS,genModel_to_RU,RU_MS,RU,DATA,TECHF,AREQM,DAYS,TrimFeed,MAXSHR,MINSHR,EMILOSS,NAVFAC,
                                           LEVL,FEDNG,NETTRD,YANUSE,FEDUSE,FERTDIST,TotMannNPK,Surplus,v_corfSetr,OBJE,
                                           QUADF,NAVFACC,NUTFAC,NUTFACC,LINEAR,QUADRA,QUADRF,QUADL,landSupCost,v_overShotEntl,v_sumEntl,
```

The processing part of the child processes is seen below:

```
 LOOP(genModel_to_RU("%genModel%",RU),

          option kill=R_RAGG;
          R_RAGG(RU,"%MS_LONG%") = YES;
*
*     --- include current regional unit in dynamic set on which
*         equations are defined in supply.gms
*
          option kill=RUNR;
          RUNR(RU) = YES;
*

             SOLVE %CAPMOD% USING NLP MAXIMIZING OBJE;
*
*          --- check if the region was feasible, if not, release some bounds
*
$batinclude 'supply\widen_bounds.gms' RU
*
 );
```

At the end of that loop, each child process generates its own result file and deletes the flag of which the presence indicates that it is still running:

```
execute_unload "%scrdir%\MSTypes_%genModel%.gdx" LEVL,FEDNG,NETTRD,YANUSE,FEDUSE,LOSSES,FERTDIST,TotMannNPK,SURPLUS,OBJE,LINEAR,QUADRA,QUADRF,QUADF,QUADL,
                                    TotMannNPK,SalesSugb,landSupCost,v_overShotEntl,v_sumEntl,MavFacc,sigmSugb,cdfSugb,pdfSugb,sugbRev,v_nonfSlack,v_corfSetr,
                                    landBal_,e_uaar,seta_,setan_,mxseta_,supbal_,FertDistExcr_,Feduse_,nutmin_,
                                    maxshr_,minshr_,reqse_,reqsn_,feduse_,inpani_,nutned_,gwpem;

execute 'del %scrdir%\supply_model_%genModel%.flag'
```

The master process starts these child processes in a loop as seen below. Before doing so, it generates for each child process a flag file which indicates that the process is working:

```
    loop(genModel_to_MS(genModel,MS),
*
*       --- generate flags for the MS (to check if the processes have finished) - no grid use
*
        put_utility batch 'shell' / "echo test > %scrdir%\supply_model_"genModel.tl:4".flag";
*
*       --- generate and start a batch file which comprises the start command the gams process
*           The start number is important: it will avoid conflicts with the ID of the handles
*           if it is zero, the process will collect the model solution itself, solving one model
*           instance at the time and not use the grid
*
        put_utility batch 'shell' / 'start /B /NORMAL %GAMSEXE% %CURDIR%\generate_supply_model.gms -workdir=%CURDIR% -scrdir=%scrdir%\'genModel.tl:4
                    ' %maxProcDir% gdir=%scrdir% --MS=' MS.tl:2 ' --genModel='genModel.tl:4
                    ' -o=%scrdir%\gen_' genModel.tl:4 '.lst --permits=%permits%';
    );
);
```

In practical terms, several dozen GAMS processes will run in parallel on the machine and will generate models. These models will be solved all in parallel by the different CPUs available. The “master process” (CAPMOD) solves for the countries which comprise only a few regions / farm types.

Finally, the master process waits until all flag files are delete (i.e. the child processes are finished), and loads in a loop the results and finally, deletes the results.

```
*
*   --- wait until all the reporting tasks have finished
*
     if ( CARD(genModel),
*
         execute "=%curdir%\util\taskSync.bat 1 240 %scrdir%\supply_model_*.flag";
*
*   --- and collect the solutions (relatively small country and second steps)
*
         loop(genModel,
*
*             --- set dynamically the gdx file name to load
*
              put_utilities batch 'gdxin' / '%scrdir%\MSTypes_' genModel.tl:4;


              execute_loadpoint LEVL,FEDMG,NETTRD,YANUSE,FEDUSE,LOSSES,FERTDIST,TotMannNPK,SURPLUS,OBJE,LINEAR,QUADRA,QUADRF,QUADF,QUADL,
                                TotMannNPK,SalesSugb,landSupCost,v_overShotEntl,v_sumEntl,NavFacc,sigmSugb,cdfSugb,pdfSugb,sugbRev,v_nonfSlack,v_corfSetr,
                                landBal_,e_uaar,seta_,setan_,mxseta_,supbal_,FertDistExcr_,feduse_,nutmin_,
                                maxshr_,minshr_,reqse_,reqsn_,feduse_,inpani_,nutned_,GWPEM;


              nHandles = sleep(0.01);
              putclose batch;
              nHandles = sleep(0.01);
         );

         execute 'if exist %scrdir%\MSTypes_*.gdx del %scrdir%\MSTypes_*.gdx';
     );
*
```