S E S 2 0 1 5
E l e v e n t h S c i e n t i f i c C o n f e r e n c e w i t h I n t e r n a t i o n a l P a r t i c i p a t i o n
S P A C E , E C O L O G Y , S A F E T Y
4 – 6 November 2015, Sofia, Bulgaria

APPROACH AND DEVELOPMENT OF TOOLS FOR DIFFERENT VARIANTS OF SPACE MISSIONS SIMULATION DEFINITION AND EXECUTION

Atanas Atanassov

Space Research and Technology Institute – Bulgarian Academy of Sciences
e-mail: At_M_Atanassov@yahoo.com

Keywords: *space missions design; multi-physic models simulation; parallel calculations.*

Abstract: *Flexible approach for synthesis of complex simulation model containing different physical and mathematical models is presented. Different consequences of calculations in field of space mission design are based on some previously developed classes of objects. A simple command language for composition of cyclograms is presented. Execution of these cyclograms is based on developed interpreter.*

Introduction

The development of flexible tools for space mission design and analysis has exclusive importance for decreasing of efforts, price and time [1]. The recent technological advances in the field of multicore processors and other components are challenges and good base for experiments in this direction.

Algorithms and program system for multi-satellite missions and experiments [2] are under development at STIL-BAS, branch in Stara Zagora. Improving flexibility and possibilities for multi-physics space simulations of the program system are shown in the present paper.

Programming system for multi-satellite mission design

The programming system for space mission design is under development. Execution of different types of tasks is possible. These tasks are related to orbital equations integration, geometric and physical quantities along the orbits, situation problems solving, satellite experiments simulation, visualization and etc. [2, 6].

Unfortunate, calculation scenarios, containing simultaneously application of more than one actual integrator [3] and situation problems solver [4] using special developed union of pools model [5] for parallelization, wasn't possible.

Problem statement

Logical mutually connected calculations will be treated as calculation flow. There are different levels of mutual connectivity. The higher level of connectivity demands mutual connection between results in the frame of one calculation flow. The low level of connectivity within the calculation flow is due to applying of unifying/common algorithms. Application of parallelization to one calculation flow depends from availability of appropriate platform and specifics of calculations. Two reasons are possible for calculation flows parallelization - possibilities for optimal use of computer system and development of complex multi-physic models and calculation algorithms.

Each calculation flow could consist from one or more consecutive stages. The parallelization between calculation flows is possible related to particular stages only. The different calculation flows are executed asynchronously until point of synchronization when exchange of results between them is necessary.

For instance, one calculation flow could contains orbital motion integration for set of satellites as first stage, calculation of different quantities heaving geometrical or physical nature as second stage, solving situation problems as third stage and etc. At the same time, a second calculation flow engaged with computation about other set of satellites is possible too. Other calculation flow could be related with space debris. Some exchange of results between separate flows after appropriated stages could be based on situation problems connected with mutual situations between objects evolved in different flows. The realization of such scenarios demands synchronization between respective/relevant calculation stages from the calculation flows.

“Computation flows” is program model for presentation of complex multi-physics applications based on higher abstraction level. This is formal thinking approach which demands appropriate specific program models and tools.

The joining of some calculation flows put the question about effective use of multi-processor system. We will have in mind exclusively shared memory system below in this paper.

Control of calculations

Finite automata approach is applied for realization of flexible scheme of calculations’ control. The algorithm is based on series of commands (cyclogram) execution. The algorithm reads and recognizes a series of commands and transits in the relevant state related to some code execution. Additional tasks are included for initialization of actual integrators and situation processors, in additional to the basic tasks listed above. The compilation of particular variant of calculation scheme consists of ordering appropriated commands. The list of commands developed at the present stage is presented in table 1.

Table 1.

Command name	First parameter	Second parameter	Third parameter
Init_Integ	p_AI_ind	p_IVP_ind	
Init_SitAnal	ini_AI_ind	ini_StPr_ind	
Init_Union			
Integ	s_AI_ind	s_IVP_ind	
'Integ Union'	Union_ind		
'Trajekt param'	t_IVP_ind		
'Sit anal'	run_StPr_ind		
'Display'	dsp_StPr_ind		
'Get_AI_rezult'	u_IVP_ind		
'Cycle'	begin_time	final_time	step_in_time
End_cycle			
END			

All commands are described by user-defined type **command** (fig. 1). This type contains different number of parameters/attributes and semantic specific for each command. The first parameter is common for all commands and contains the name. Other parameters of the commands are presented by integer or real types. Operator UNION is used for description of all variants of command types (fig. 1), because each command has individual format,

The couple of commands **Cycle** and **End_cycle** are important construction for repetition of entire cyclogram or subseries of commands closed between them. All commands which must be repeated are inserted between the **Cycle** and **End_cycle** commands. The commands for initialization are placed at the beginning, before the command **Cycle**.

```
type    command              !
  character*17  name
 UNION
  MAP
   integer num_com      ! The zero element
                        ! contain number of
                        ! commands
  END MAP
  MAP                   ! AI preparation
   integer   p_AI_ind  ! (p)- za pointer
   integer   p_IVP_ind
  END MAP
  MAP                   ! IVP initialization
   integer  s_AI_ind     ! index of integrator
   integer  s_IVP_ind   ! index of IVP
  END MAP
  MAP                       ! Init_SitAnal
    integer   ini_StPr_ind  ! index to solver
    integer   ini_AI_ind     ! index to intrgrator
  END MAP
  MAP     ! Creation and initialization of Union
   integer  ins_AI_ind      ! index of the solver
   integer  com_ind_integ ! multiple integration
  END MAP
  MAP                  ! Multiple IVPs integration
   integer   Union_ind      ! index of Union
   integer   index_IVP(0:10)
   integer   union_atr(2)
  END MAP
  MAP         ! Get data after Union execution
   integer   u_IVP_ind ! index of IVP
  END MAP
  MAP              ! Traject. param. calculation
   integer   t_IVP_ind
  END MAP
  MAP                          ! Situation analysis
   integer   run_StPr_ind ! index of solver
   integer   run_AI
  END MAP
  MAP         ! command 'Disply'
   integer   dsp_StPr_ind
  END MAP
  MAP                     ! Cycle
   real*8     begin_time
   real*8    current_time
   real*8     final_time
   real*8    step_in_time
   real*8     shift_time
   integer   end
  END MAP
  MAP              ! End cycle
   integer   back   ! counter return 'back' steps
  END MAP
 END UNION
end type  command
```

Fig. 1. User-defined type used for cyclogram commands definition

The execution of the commands is based on object-descriptors heaving specific user-defined types [6]. These types appear as templates for description of objects which could be created and used for computational control. For instance, **pool_par** represents actual integrators and situation problem solvers. The type **IVP_par** represents the solving of initial value problems. The type **TrajParam** defines calculation of some parameters along the orbits. The types **SitProblems** and **PoolThUnion** are used for definition of situation problems and union of pools of threads.

Command interpreter

Finite automata approach is used for new version of the program system. Program fragment on figure 3 accepts series of commands and interprets them in sequent mode. The algorithm recognizes the current command and transit to the respective stage and call to corresponding solver or calculation code. Each command leads to calculation of portion of final results. Different primitive of calculations are used like elemental building units for assembling calculation models in one scientific field. A complex model can contains some objects from equal type. Every object could be presented through different characteristics or parameters. So each object is described through user-defined type. Such types are shown in [6]. Objects of the same type are united in class of objects. These classes are created by special polymorphic subroutine [6]. These types contain different specific attributes. Some of them represent geometric or physical quantities which accept values in some interval. Other attributes represent meta-data – addresses and sizes of data structures.

Five object-descriptors are defined through separate user-defined types at the present stage. The access to particular object at random point of the program is possible by global class-descriptor (named common area in fortran) (fig. 2a). This common area contains the number of objects in the class and address of the array in computer storage containing objects of the class (fig. 2a). For example, named common area /**c_AIs** / contains descriptor of the class of parallel actual solvers based on pool of threads program model- integrators of ordinary differential equations and situation processor solvers. The class of "initial values problems" objects is accessible through named common area / **c_IVPs** / (fig. 2b). The class of situation problems is presented trough class-descriptor and common area / **c_StPrs** /. The class of "union of parallel solvers" could be accessed through common area / **c_UPths** /. Figure 2b illustrates the access to classes of objects based on pointers.

The object classes are dynamic objects. The addition of each new object to respective class is connected with changing of the address in the storage.

```
integer            AIs_descriptor_adr
common  /c_AIs/num_AIs,AIs_descriptor_adr
type     (pool_par)    AIs(num_AIs)
!_____________________________________
integer            IVPs_descriptor_adr
common  /c_IVPs/num_IVPs,IVPs_descriptor_adr
type     (IVP_par) IVPs(num_IVPs)
!_____________________________________
integer            TrPas_descriptor_adr
common  /c_TrPas/num_TrPas, TrPas_descriptor_adr
type     (TrajParam) TrPas(num_TrPas)
!_____________________________________
integer            StPrs_descriptor_adr
common  /c_StPrs/num_StPrs,StPrs_descriptor_adr
type     (SitProblems) StPrb(num_StPrs)
!_____________________________________
integer            UPths_descriptor_adr
common  /c_UsPTh/num_UsPth,UPths_descriptor_adr
type     (PoolThUnion) Union_atr(num_UsPth)
                (a)

POINTER(    AIs_descriptor_adr,AIs)
POINTER(  IVPs_descriptor_adr,IVPs)
POINTER(TrPas_descriptor_adr,TrPas)
POINTER( StPrs_descriptor_adr,StPrb)
POINTER(UPths_descriptor_adr,Union_atr)
                (b)
```

Fig. 2. An access to classes of objects is shown. (a) Description of object classes. (b) Allocation of object classes according to addresses.

The addresses of the real data, which are pointed as actual arguments, are contained in attributes of objects-descriptors. Each object-descriptor is used for simulations in the frame of one calculation flow.

Figure 3 represents fragment of the interpreter of command series (cyclograms). Actual arguments of the subroutines don't point directly to transmit data. The access to data addresses is provided. The data transmitting is provided through special developed object-descriptors [6]. Each command has specific parameters for synonymous access to necessary data.

Actual parameters of command **Integ** for multi satellites' orbits integration are presented on figure 3 in some details for illustration. Addresses of data passed as actual arguments to respective subroutines related to other commands are described through analogous approach.

The sophisticated description of actual parameters which is shown on figure 3 is result from application of object-descriptors and command parameters for assigning command to particular calculation flow. The commands initiate usually big portions of calculations and so the pointed sophistication doesn't decrease substantially the speedup.

The command parameters play important semantic role connected with building of calculation algorithm. Except parameters which are defined in the course of cyclogram compilation, there are other hidden parameters which are used for control of cyclogram execution. For instance, the values of the two parameters **End_cycle%back** and **End_cycle%end** are determined from preprocessor depending from their mutual disposition in the frame of cyclogram. The first parameter provides the return of command counter to beginning of the cycle, and the second of them – exiting from the cycle.

A preprocessor for preliminary command cyclogram analysis is under development. It checks the syntax of used commands and determines some of their parameters. The preprocessor executes an automatic adjustment of commands for cycle organization.

```
run:DO WHILE(cyc(counter)%name.NE.'END'.AND.cyc(counter)%name.NE.'end'.and.counter.LE.cyc(0)%num_com)
        SELECT CASE(TRIM(cyc(counter)%name))
                CASE('Init_Integ')
                  CALL  Data_AI(addresses of actual parameters)
                  CALL  Preparation_AI(addresses of actual parameters)
                      counter= counter + 1
                CASE('Init_Union');
                  CALL  InitUnionPools(addresses of actual parameters)
                     counter= counter + 1
                CASE('Init_SitAnal')
                  CALL  Data_Sit_Solver(addresses of actual parameters)
                  CALL  Preparation_Sit_Solver(addresses of actual parameters)
                    counter= counter + 1
                CASE('Integ')
                  CALL  traekt_AI(AIs(cyc(counter)%s_AI_ind) %thread_par_adr, &
                                   AIs(cyc(counter)%s_AI_ind) %num_threads, &
                                   AIs(cyc(counter)%s_AI_ind) %counter_adr, &
                                  IVPs(cyc(counter)%s_AI_ind) %num_objects, &
                                  IVPs(StPrb(cyc(counter)%ini_StPr_ind)%IVPs_index)%t_adr, & !
                                  IVPs(StPrb(cyc(counter)%ini_StPr_ind)%IVPs_index)%dt_adr, &
                                  IVPs(cyc(counter)%s_AI_ind) %xvn_adr, &
                                  IVPs(cyc(counter)%s_AI_ind) %xvk_adr, &
                                  IVPs(cyc(counter)%s_IVP_ind)%transfer_data_adr)
                    counter= counter + 1
                …
                CASE('Cycle')
                    counter= counter + 1
                CASE('End cycle')
                     cyc(cyc(counter)%back)%current_time= cyc(cyc(counter)%back)%current_time + &
                                                          cyc(cyc(counter)%back)%step_in_time
                     cyc(cyc(counter)%back)%    shift_time= cyc(cyc(counter)%back)%shift_time + &
                                                          cyc(cyc(counter)%back)%step_in_time
                    counter= cyc(counter)%back
                CASE DEFAULT
                   !  WRONG  command!!!
  END SELECT
 END DO  run
```

Fig. 3. Fragment of subroutine Drive_M. This fragment realizes interpretation of commands. The command **Integ** is presented in details.

Conclusion

A development of interpreter of commands' cyclogram is presented. The application of such interpreter allows more flexible and adaptable execution of computation scenarios when complex multi-physic model are simulated.

Further development of presented approach demands development of interactive tools (appropriated dialogue) for definition of objects' attributes and cyclogram compilation.

References:

1. Wertz, J.R., Larson, W.J., 1999. Space Mission Analysis and Design, third ed. Microcosm Press, Kluwer Academic Publishers.
2. Atanassov, A.M., 2013Program System for Space Missions Simulation - First Stages of Projecting and Realization. SES 2012, 209-214.
3. Atanassov, A.M., Parallel, adaptive, multi-object trajectory integrator for space simulation applications, Advances in Space Research, Volume 54, Issue 8, 15 October 2014, Pages 1581-1589.
4. Atanassov, A.M., 2014, Parallel Solving of Situational Problems for Space Mission Analysis and Design. SES 2013, 283-288.
5. Atanassov,A.M. Method of Thread Management in a Multi-Pool of Threads Environments, SES 2014, 12 – 14 November 2014, Sofia, Bulgaria.
6. Atanassov, A.M., DEVELOPMENT CLASSES OF OBJECTS' DESCRIPTORS FOR SPACE MISSIONS SIMULATION, proceedings of 9th scientific conference Space Ecology Safety, 2015, 2016, pp.