S E S   2 0 1 6
Twelfth Scientific Conference with International Participation
SPACE, ECOLOGY, SAFETY
2 – 4 November 2016, Sofia, Bulgaria

UNIFICATION OF "POOL OF THREADS" CONTROL IN THE FRAMES OF DIFFERENT PARALLEL SOLVERS

Atanas Atanassov

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

Keywords: *parallel calculations, pool of threads model; multi-physic models simulations.*

Abstract: *Simulation of space missions at various stages of their preparation and implementation is essential. Possibilities for application of detailed models for presentation of various aspects of prepared experiments are created with development of computer technologies and the advent of multi-core processors. More versatile and more detailed computer simulations of space missions require the development and implementation of appropriate software for more efficient use of multicore processors.*

The application of computational threads is convenient approach to perform parallel computing. The "pool of threads" is a means for dynamic scheduling of calculations when solving irregular computational problems, related to complex calculation models, which vary in the course of simulation time. This programming model achieves optimal distribution of tasks among the threads, which assure high efficiency of computing systems.

Variants of "pool of threads" model for developed solvers are realized. An unified approach and program realization which are applicable in a variety of solvers are proposed in the present work. A simplification of the program code by development of complex program systems which include different solvers is attainable by this approach. This achieves a simplification of the programming code when develop complex software systems including various solvers.

УНИФИЦИРАНЕ НА УПРАВЛЕНИЕТО НА ПУЛ ОТ ТРЕДОВЕ ЗА РАЗЛИЧНИ ПАРАЛЕЛНИ ИЗЧИСЛИТЕЛНИ СРЕДСТВА

Атанас Атанасов

Институт за космически изследвания и технологии – Българска академия на науките
e-mail: At_M_Atanassov@yahoo.com

Резюме: *Симулацията на космически мисии на различни етапи от тяхната подготовка и провеждане е от съществено значение. С развитието на компютърните технологии и навлизането на много-ядрените процесори се създава възможността за прилагане на детайлни модели за представяне на различни аспекти, системи (подсистеми) протичащите процеси. По-разностранното и по-детайлно описание на реалните елементи на космическите мисии и компютърното симелиране изисква разработката и прилагенето на съответни програмни средства за по-ефективното използване на многоядрените процесори.*

Прилагането на изчислителни тредове е удобен подход за извършване на паралелни изчисления. Моделът "пул от тредове" е средство за динамично планиране на изчисленията при решаване на нерегулярни изчислителни проблеми, свързани с модели сложността на изчисленията по които се менят в симулационното време. С този програмен модел се постига оптимално разределение на задачите по тредовете с което се поддържа висока ефектимност на изчислителните системи.

Разработени са варианти на модела "пул от тредове" за разработените досега солвери. В настоящата работа се предлага единен подход и програмна реализация която е приложима при различни солвери. С това се постига упростяване на програмния код при разработка на сложгни програмни системи включващи различни солвери.

Introduction

Computer simulations are important tool for design at different stages of space mission preparation and implementation [1, 2]. The modern solution of different kinds of problems demands detailed simulations and a lot of computer time.

The present development of computer technologies provides great possibilities for access to power multi-core processors. The use of these processors requires previously developed and appropriately parallelized an algorithms.

Parallelization based on threads is applied to problems having large dimensions. An effective control and dynamic scheduling of parallel calculations is achieved through application of pool of threads program model [3].

Algorithms and program tools for space mission end experiments simulation are under development at branch of SRTI, BAS in Stara Zagora. Parallel ordinary differential equation system integrator for satellite orbits propagation [4] and parallel satellite orbital situation problems solver [5] were developed. The two solvers are included in simulation program system which development is under progress [6]. The two solvers are based on pool of threads program model [3].

In the present article is shown an approach to universalization of the subroutine for pool of threads control.

Problem statement

Multi-satellites space mission simulations demand solving of different types of problems. Some of them are:

– integration of motion of satellites and other objects;
– situation analysis for establishing time intervals when different specific conditions are fulfilled, related to satellite experiments and other activities;
– simulation of the work of scientific instruments and measurements, active experiments;
– charged particles motion;
– thermodynamics and electro-discharging problems.

Numerical integration of satellites motions equations, as well as solving variety of multi-satellite situation problems are related to irregular calculations. The reasons are different:

– selection of different integration methods for different segments for each of satellite orbits depending on local orbital curvature and velocity;
– different force models for each of satellites depending on type of orbit;
– different situation conditions are met on different segments of the satellite orbits.

The efficiency of parallel calculations depends on partitioning of all task to subtask. A problem appears when calculations are irregular, because then subtasks are uneven. A good choice of parallel calculations scheduling is to apply pool of thread program model.

Due to the above reasons, the amount of calculations for every step in simulation time may be different. The analysis discloses the spatial and temporal modality of the irregularity of the calculations.

In addition to optimization, parallel algorithms are applied in case of large amount of calculations. Implementation of threads parallelism and dynamic scheduling of calculations based on pool of threads program model are effective approaches to irregular calculations.

The development of multi-physic applications includes different kind of solvers, based on “pool of threads” program model and their simultaneous execution leads to the presence in the processor cache storages of similar, close in semantics codes. For example, the developed program model “union of pools of threads” aims simultaneous execution of various solvers [7]. A minimization of the size of execution code is as important as its simplification. The application of universal tools is helpful on the stages of development and support.

The two developed solvers, the one for satellites motion integration and other for situation problems solving, use analogous subroutines for control of the threads of each solver/pool. Each solver is based on different code, organized with different subroutines and uses data with different structures and semantics. The list of actual variables for each solver reflects his specifics. Different numbers of subroutine names are given to actual solver, which additional expands his potentialities.

The developed “pool of threads” program model is synchronized with parent thread and works “step by step” in simulation time. Threads are started and work competitively while exhaust the available tasks. Then everything is repeated for the next step of simulation time. Several pools of threads are controlled in the frame of the model “union of pools” [7]. When one solver finishes attached problem “its” threads release processor cores, which are used further from additionally activated threads of other solver, which participates in the union. Subroutines for dynamic parallel calculations scheduling with almost equal codes are in the computer storage, when several solvers are executed simultaneously.This can lead to waste of memory and possibly affect the execution time.

The subroutines for dynamic scheduling and control of the threads are called from buffer subroutines, which transfer to solvers all data necessary for control and processing [4, 5]. Furthermore, these buffer subroutines pass to solvers one or more names of subroutines. One additional name of subroutine, containing the code describing right hand equation system, is passed to parallel integrator of ordinary differential equation systems [4]. So, two actual integrators can work with different motion models. Furthermore, integration methods could be changed too.

New solution

The new solution is achieved through polymorphism - a possibility for using one subroutine name calling different subroutines according to specifics of their arguments. Figure 1a shows definition of the generic name UPC (Universal Pool Control) in the frame of fortran 95 standard. Two subroutine names **Pool_threads_Control_1** and **Pool_threads_Control_2** are used for the two developed solvers respectively - parallel ordinary differential equation systems integrator and parallel situation problems solver.

```
MODULE     RN
    …
 type      task_descriptor_template_integrator              b). user defined type containing parameters necessary
  integer     num_obj,num_equestions                            for integrator
  integer     adr1,adr2
  integer     adr_Grv_model,len_Grv_model
  character    izbor*1
 end type  task_descriptor_template_integrator
!
 type      task_descriptor_template_situations              c). user defined type containing parameters necessary
  integer     num_sat                                           for situation processor
  integer     t_adr,dt_adr,xvn_adr,xvk_adr
  integer     max_num_sit,num_sit_prob
  integer      sci_problem_adr,len_sci_task
  integer     TrajectParam_adr,TrajectParam_len
 end type  task_descriptor_template_situations
    …
  INTERFACE     UPC                                         a). definition of polymorphism
   SUBROUTINE     Pool_threads_Control_1(th_id_num,num_threads,ha_1,adr_glb_counter,thread_par_local,lgranul, &
                                  rkfasd_UPC,task_descriptor_adr,numobj,pertur)
     external                 rkfasd_UPC,pertur
     integer                 th_id_num,num_threads,ha_1,adr_glb_counter,thread_par_local(4),lgranul
     integer                          task_descriptor_adr,numobj
   END SUBROUTINE  Pool_threads_Control_1

  SUBROUTINE     Pool_threads_Control_2(th_id_num,num_threads,ha_1,adr_glb_counter,thread_par_local,lgranul, &
                    Psitanal_UPC,task_descriptor_adr,num_sit_prob)
     external                 Psitanal_UPC
     integer                 th_id_num,num_threads,ha_1,adr_glb_counter,thread_par_local(4),lgranul
     integer                         task_descriptor_adr,num_sit_prob
   END SUBROUTINE  Pool_threads_Control_2
 END INTERFACE  UPC
    …
END MODULE  RN
```

Fig. 1 a). Definition of polymorphic name UPC, which will substitute the names Pool_threads_Control_1 and Pool_threads_Control_2; b) and c) define templates of descriptors of two solvers.

The lists of arguments of subroutines **Pool_threads_Control_1** and **Pool_threads_Control_2** contain two groups of arguments. One of them is fully identical for the two solvers and it is used for pool's control. The other group of arguments is specific for each solver and varies in number and semantic. Instead the second group of arguments, structure containing them or their addresses is passed to subroutine **UPC**. Figure 1b,c shows versions of structures containing arguments for each of the two developed solvers. The name of particular solver is passed as argument to subroutine UPC too. Furthermore, each solver can receive additional subroutine name which must be used in specific current variant of the solver. This name is passed as additional argument to subroutine **UPC**. The possibility for using of additional subroutines is achieved by buffer subroutines [4, 5]. The names of these buffer subroutines are used as first subroutines by threads creation. The buffer subroutines accept values and addresses of all arguments necessary for each solver and contain specific for each solver additional subroutine names. While the application of structures, containing specific for solvers arguments, resolves part of the problem, the necessity to transmit a different number of subroutine names remains.

```
SUBROUTINE     SatelliteIntegratorUPC(th_id_num)
  USE  RN, only: perturb,UPC
    external               pertur,rkfasd_UPC, task_descriptor_template_integrator
    …
    type (task_descriptor_template_integrator) task_descriptor
    …
    task_descriptor%num_obj         = numobj;             task_descriptor%num_equestions= 6
    task_descriptor%adr1            = adr1;               task_descriptor%adr2             = adr2
    task_descriptor%adr_Grv_model= adr_Grv_model;   task_descriptor%len_Grv_model = len_Grv_model
    …
   CALL  UPC(th_id_num,num_threads,ha_1,adr_glb_counter,thread_par_local,lgranul, &
                         rkfasd_UPC,task_descriptor_adr,numobj,pertur)

END SUBROUTINE  SatelliteIntegratorUPC ! Universal Pool of threads Control
```

```
SUBROUTINE     SituationProcessorUPC(th_id_num) _situations
 USE RN, only:UPC, task_descriptor_template_situations
   external      Psitanal_UPC
   …
   type (task_descriptor_template_situations) task_descriptor
   …
 task_descriptor%num_sat           = num_sat
 task_descriptor%t_adr              = t_adr;                 task_descriptor% dt_adr           = dt_adr
 task_descriptor%xvn_adr           = xvn_adr;               task_descriptor%xvk_adr          = xvk_adr
 task_descriptor%max_num_sit      = max_num_sit;          task_descriptor%num_sit_prob    = num_sit_prob
 task_descriptor%sci_problem_adr = sci_problem_adr;     task_descriptor%len_sci_task      = sci_task_len
 task_descriptor%TrajectParam_adr= TrajectParam_adr;   task_descriptor%TrajectParam_len= TrajectParam_len;

 CALL  UPC(th_id_num,num_threads,ha_1,adr_glb_counter,thread_par_local,granule, &
                  Psitanal_UPC,local_task_descriptor_adr,local_num_sit_prob)

END SUBROUTINE  SituationProcessorUPC
```

Fig. 2. Buffer subroutines where the two solvers are called by polymorphic name UPC and using the two structures from figure 1, containing specific for each of the solvers arguments.

Figure 2 illustrates fragments of the buffer subroutines calling the two solvers. The calling of subroutine UPC in the two cases concerning each of the solvers is shown. In our case the names **Pool_threads_Control_1** and **Pool_threads_Control_2** are synonyms which are two entry points of same subroutine. The two entry points of the subroutine allow to be called with different number and semantic parameters. In our case the number of parameters is reduced through application of specific structure, explained above.

Other problem is related to calling of the given solver from pool of thread control subroutine, because every solver has specific list of actual arguments. One part of these arguments is related to pool of thread control and other part to processing data, model parameters or initial conditions. Third part of arguments represents names of subroutine passed to the solver. One of the names is mandatory - this is the name of the solver called from pool of threads control subroutine and the number of other names is according to the specifics of the solver. While the first groups of arguments from the list can be packed in user defined structure, this is not possible for the names of subroutines in the frame of fortran 95. Therefore, they are placed at the end of the list of actual parameters. Some variable names are reserved at the end of the list of arguments of the solver, which can be interpreted eventually from the solver as names of subroutines or functions.

The name “solver” is in the list of formal arguments of the subroutine for pool of threads control, shown on the figure 3. The actual name of the solver (after linker generates the executable) is address, set in the respective buffer subroutine (figure 2).

Conclusion

The proposed unification of the pool of thread control subroutine for application in different solvers simplifies the routines in the frame of multi-physic applications, reducing to only one subroutine for all different solvers.

Assigning a name of a subroutine (function) to the component of structure is possible in the newest versions of fortran (fortran 2003).

```
SUBROUTINE      Pool_threads_Control_1(th_id_num,num_threads,ha_1,adr_glb_counter,thread_par,granule, &
                                       Solver,task_descriptor_adr,num_tasks,RHFun)
 USE  DFmt
 USE  RN
ENTRY                Pool_threads_Control_2(th_id_num,num_threads,ha_1,adr_glb_counter,thread_par,granule, &
                                       Solver,task_descriptor_adr,num_tasks )
  integer                      th_id_num,           ha_1,adr_glb_counter,thread_par(4),granule
  integer                                                                  task_descriptor_adr
  integer, automatic ::  loc_ha_1,loc_counter,loc_counter0
  integer             glb_counter
 POINTER(adr_glb_counter,glb_counter)

          ha_beg= thread_par(3);              ha_end= thread_par(4)
        loc_ha_1= ha_1
 DO WHILE(.true.);

           k=  WaitForSingleObject(ha_beg,WAIT_INFINITE) ! Event for running of the thread

  DO WHILE(glb_counter.LT. num_tasks)
    k= WaitForSingleObject(loc_ha_1,WAIT_INFINITE);
             obj_remain= num_tasks-glb_counter
          IF(obj_remain.GT.granule.AND.granule.GT.1) THEN
                      loc_counte0= glb_counter+1;              ! increment for serial subtask
                      glb_counter= glb_counter + granule
                     k= SetEvent(loc_ha_1)                     ! Allows others threads to get tasks
            DO  loc_counter=loc_counte0,loc_counte0+ granule-1
              IF(loc_counter.GT. num_tasks) EXIT
                              loc_adr= adr + (loc_counter-1)*adr_len;
                 CALL  Solver(loc_counter,task_descriptor_adr,num_tasks,th_id_num,RHFun)
            END DO
                                                        ELSE
                      glb_counter= glb_counter + 1;         ! increment for serial subtask
                      loc_counter= glb_counter;              ! remember in thread's local storage
                                 k= SetEvent(loc_ha_1)   ! Allows others threads to get tasks
             IF(loc_counter.GT. num_tasks) EXIT
                   loc_adr= adr + (loc_counter-1)*adr_len;
                 CALL  Solver(loc_counter,task_descriptor_adr,num_tasks,th_id_num,RHFun)

            ENDIF
  END DO
            k= ResetEvent(ha_beg) ! Preparation of the event for next step in the time
            k=     SetEvent(ha_end) ! This event signals about finish of the thread
 END DO;

END SUBROUTINE  Pool_threads_Control_1
```

Fig. 3. Unified subroutine which is used in the two developed solvers.

Additional investigations are necessary to determine the influence of the presented unification on run-time efficiency.

References:

1. Wertz, J.R., Larson, W.J., 1999. Space Mission Analysis and Design, third ed. Microcosm Press, Kluwer Academic Publishers.
2. Eickhoff, J., 2009. Simulating Spacecraft Systems, vol. 353. Springer-Verlag, Berlin Heidelberg.
3. Rauber, T., and Gudula Rünger. *Parallel programming: For multicore and cluster systems*. Springer Science & Business Media, 2013.
4. Atanassov, A., 2014, Parallel, Adaptive, Multi-Object Trajectory Integrator for Space Simulation Applications, Adv. Sp. Res., v. 54, № 8, p. 1581–1589.
5. Atanassov, A.M., Parallel Satellite Orbital Situational Problems Solver for Space Missions Design and Control. Adv. Sp. Res., v. 58, № 9, p. 1819-1826.
6. Atanassov, A., Program System for Space Missions Simulation – First Stages of Projecting and Realization, In Proceedings of SES 2012, 209-214, 2013.
7. Atanassov, A., Method of Thread Management in a Multi-Pool of Threads Environments, In Proceedings of SES 2014, 241-246, 2015.