S E S   2 0 1 2
Eighth Scientific Conference with International Participation
SPACE, ECOLOGY, SAFETY
4 – 6 December 2012, Sofia, Bulgaria

AN ADAPTIVE PARALLEL INTEGRATOR OF ORDINARY DIFFERENTIAL EQUATIONS SYSTEM FOR SPACE EXPERIMENT SIMULATION

Atanas Atanassov

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

Keywords and phrases: *ordinary differential equation system integration, parallel algorithms*

Abstract: *A version of parallel ordinary equations system integrator for multi-satellites missions' simulation is presented. The integrator is based on different order Runge-Kutta-Fehlberg schemes. An optimal scheme is selected for every integration step for achieving necessary local error which improves calculation effectiveness. An integration of differential equation systems is executed simultaneous in the time with constant step. An integration with variable step in the frame of one system time step is applied if the maximum order calculation scheme is not enough for achieving the necessary local accuracy. A possibility for determination of the motion of every object (satellite, space debris, charged particles) based on individual propagation model of forces is realized. That approach allows simultaneously solving of different classes mechanical and electro-dynamical problems by proposed integrator.*

Introduction

Satellites with different orbits are used in some scientific experiments. In these cases propagation models are different on different parts of the orbits. Atmosphere drag and high harmonics of gravity field is significant on low parts of the orbits and decreas on large distance. The basic components of Earth gravity field remain significant on larger distance and influence from the Moon and the Sun could be added. The application of the same perturbation model for all objects on different part of orbits is inefficient when motion equations are integrated simultaneously. This is especially significant by using some calculation time consuming atmosphere models.

Satellite experiments simulation can demand solving of motion equation for different kind of objects. Such objects beside satellite are charged particles (electrons, protons, ions), charged dust particles, space debris. The right hand of SODE which describe its motions can contain different functions describing forces with different nature.

Different aspects of parallel integration of SODE about satellite motion are explained in [1-8].

В настоящата работа се представя вариант на интегратор на СОДУ основно за нуждите на имитационното моделиране.

Multi body ODES integrator

Specialized SODE integrator for Earth satellite propagation was proposed [9]. Computer code was developed for requirements of simulations. A motion of N objects is integrated with constant step Δt in the time and a choice of optimal calculation scheme. The next step of development of the integrator was a parallel version based on calculation threads [10].The integration of the motion equations for different satellites was based on equal perturbation forces model.

New improved version of the integrator is presented in this work. The integrator has possibilities for simultaneous solving of different classes of problems with different dimension and perturbation forces.

Figure 1 shows the possibilities integrator to be applied for solving different classes of problem in the frame of one simulation model.

Integrator initialization

The initialization of the integrator is made by subroutine InitIntegrator (fig. 2). Initialization of the integrator consist in creation of a number of threads (according the value of formal parameter **num_threads** which is determined in parent routine), events for beginning and ending of calculations

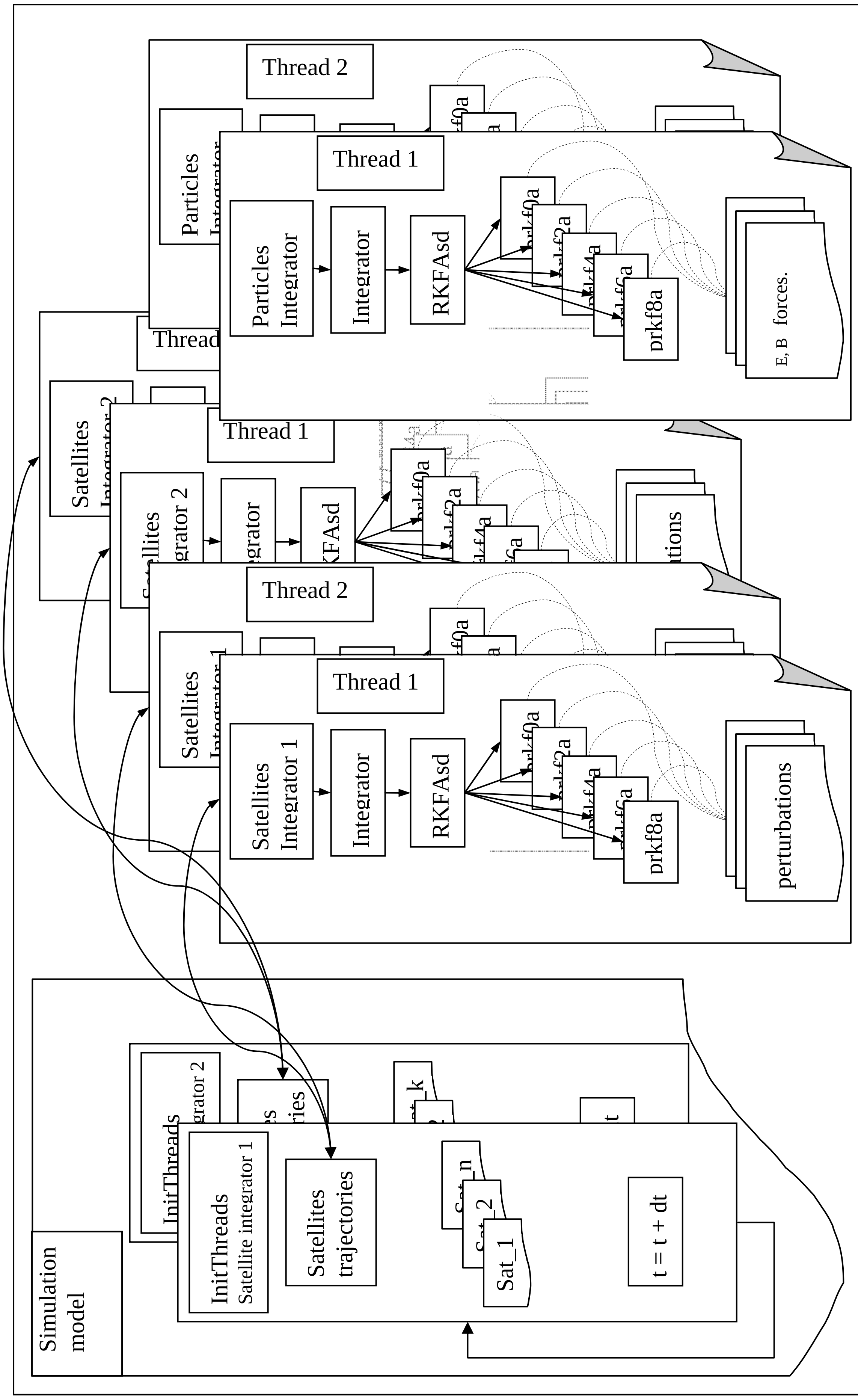

Figure 1. Logical scheme of the integrator and relations in the frame of entire simulation model.

(ha_beg and ha_end) for every thread in interval Δt and one additional event with handler ha_1**,** which serve for synchronization between threads. The threads remain in suspended state after it's creation to the moment when information necessary for SODE integration is prepared.

```
SUBROUTINE     InitIntegrator(IntegratorName, num_threads, thread_par, ha_1)
USE  DFmt
 external                          IntegratorName
  integer                                              num_threads,          ha_1
  integer                                                      thread_par(4,num_threads)
  integer         security/0/,stack/0/,thread_id,security1/0/, kkod/0/
                   ha_1 = CreateEvent(security1,.false.,.true.,0)
       DO    i=1,num_threads
        thread_par(2,i)        = i
        thread_par(1,i)        = CreateThread(security,stack,IntegratorName,LOC(thread_par(2,i)),&
                                                          CREATE_SUSPENDED,thread_id)
        thread_par(3,i)        = CreateEvent (security1,.false.,.true.,0)
        thread_par(4,i)        = CreateEvent (security1,.true.,.false.,0)
       END DO
RETURN
ENTRY         StopIntegrator(num_threads,thread_par)
       DO    i=1,num_threads !numth
          kod= TerminateThread(thread_par(1,i),kkod)
          kod=    CloseHandle(thread_par(1,i))          ! ha
          kod=    CloseHandle(thread_par(3,i))          ! ha_beg
          kod=    CloseHandle(thread_par(4,i))          ! ha_end
       END DO;
END SUBROUTINE  InitIntegrator
```

Fig. 2. Code of integrator initialization subroutine

The threads accept the name (address) of the basic integrator subroutine by variable IntegratorName, described in operator external during of their creation.

The integrator is stopped by subroutine StopIntegrator after completing integration of SODE. This subroutine terminates the existence of threads and events for their synchronization. The integrator may be started again with other number of threads, objects and perturbation models which determine their motion. The subroutine InitIntegrator can be called more than one time with equal or different IntegratorName. Different simultaneous existing integrators could be initialized. The name of integrator reflects specifics of solved SODE.

Adjustment of the integrator to specific problem

The SatelliteIntegrator is buffer subroutine with the purpose of adjusting the real SODE integrator to features of solved problem. The SatelliteIntegrator name is contained in formal parameter IntegratorName. It reflects the specifics of corresponding class problems which the integrator solves. SatelliteIntegrator **(**or any other one on its place) is started as first subroutine from every initiated thread when its state is changed to non-suspended in some moment from parent process. It transfers different data addresses to integrator such as the name of subroutine where the right hands of the solved systems are described. Different version of the integrator, every one for different classes SODE, could be started.

The full information with general system character (handlers of events for threads synchronization and threads parameters), as well as the special information about solving SODE (addresses of data structures for initial conditions, perturbation forces) are transferred to integrators by global data (common blocks) by means of short interface subroutines from type of SatelliteIntegrator. If electro-dynamical problem is solving for example, instead previous SatelliteIntegrator other subroutine with the same template, containing other subroutine describing right hand of differential equation (pointed in the operator external) will be applied. The names of the common blocks will be different but actual parameters of subroutine Integrator must be analogously as by SatelliteIntegrator.

Common block /chth/ transfer number of threads and handlers of events for synchronization. Common block /cAdr_traekt/ transfer address of structure containing initial value for integration, results, integration step, tolerances and global counter for objects. Common block /cMainExpDef/ transfer number of satellites (objects) and parameters describing models of perturbation forces.

```
SUBROUTINE     SatelliteIntegrator(th_id_num)
 USE DFmt
 external              pertur
 integer                      th_id_num
 integer       thread_par(4,num_thrd),  thread_par_local[AUTOMATIC](4)
!
  type      general_data
   integer   adr1, adr2 !
   real*8   time,dt
  end type  general_data
  type     (general_data)            gen_data                      ! global data
  type     (general_data), AUTOMATIC :: gen_data_loc   ! local data
  character,  AUTOMATIC  ::      izbor*1
!............................................................
  integer               thrd_par_adr,ha_1,ha_1l !thrd_hand_adr,
  common   /chth/num_thrd,thrd_par_adr,ha_1       ! threads number
!...............................................................
  integer,     AUTOMATIC :: adr_glb_count,adr1l,adr2l
  integer             adr1,adr2,glb_counter ! handlers_address
  common   /cAdr_traekt/adr1,adr2,glb_counter,numsa_
!_____________________________________________________________________________
 integer                                                  adr_Grv_model
 integer                                   Main_adr_orb_data, Main_adr_Grv_model,len_Grv_model
 common   /cMainExpDef/numsat,Main_adr_orb_data, Main_adr_Grv_model,len_Grv_model !
!_____________________________________________________________________________
  POINTER(thrd_par_adr, thread_par);          POINTER(gen_data_adr,gen_data)

         adr_glb_count  = LOC(glb_counter); adr1l= adr1; adr2l= adr2; ha_1l= ha_1; izbor= 'y'
         adr_Grv_model= Main_adr_Grv_model
      thread_par_local(:)= thread_par(:,th_id_num)

    CALL  Integrator(th_id_num,num_thrd,pertur,adr_Grv_model,len_Grv_model,numsat, &
                              thread_par_local,adr1l,adr2l,ha_1l,adr_glb_count,izbor)
END SUBROUTINE  SatelliteIntegrator
```

Fig. 3. Example of a code of buffer subroutine for adapting integrator to concrete problem

Each buffer subroutine finally calls the real SODE integrator (Fig.3)**.**

The number of threads for every initiated integrator must be less than number of objects from particular class. When the number of objects is higher than number of processor cores, then number of the threads is normal to be less than objects for approaching effective of integration. Every additional thread leads to increasing of overhead for synchronization.

Each free thread takes a next non-solved SODE. While one particular thread takes SODE, other free threads do not participate in the choosing process even if they are finished their calculations. This is guaranteed from signal event with handler **ha_1,** which is unique for every integrator and is created by process of initialization (fig. 4).

Synchronization between parent program and integrator

For each step of system time on specific place in parent program (Fig. 5), the events for beginning of calculations for each thread are set in signal state, which starts their work. The parent program begins to wait treads to solve SODE for every object. The threads send signal for 'end' when all of the SODE are solved. The parent program takes the coordinates and velocities of objects determines for $t=t_0+dt$ and it is possible to start other calculations concerning interval Δt. After that it is passing to next step from system time.

Conclusion and future work

Development of flexible parallel system ordinary differential equation integrator is presented. The flexibility is concerned to optimal order of integration schemes selection as possibility for changing differential equations systems (depending of type of the problem solved), number of threads, perturbation forces models, number of objects etc. Simultaneous running of more of one integrator is possible. An approach for realization of SODE integrator based on threads and program codes are shown.

SODE integrator is developed applying program language Fortran 95, Compaq Fortran compiler and API function for multi-trading.

```
SUBROUTINE     Integrator(th_id_num, num_threads, RHFun, adr, adr_len,numsat, thread_par,   &
                                                   adr1, adr2, ha_1, adr_glb_count, izbor)
USE DFmt
  external                                              RHFun
  character                                                         izbor*1
  integer             th_id_num,adr,adr_len,glb_counter,numsat
  integer, AUTOMATIC :: loc_counter,loc_adr

  integer            thread_par(4)
  type     general_data
   integer   adr1, adr2 ! transfer_data address work_data address
   real*8   time,dt
  end type  general_data
  type     (general_data) gen_data
  type  data
   real*8  t,dt
   real*8  xvn(6),xvk(6),eps(6)
  end type
  type (data)              transfer_data(numsat)
  type     work
   integer  redmet,in !(6)!/6*0/
   real*8   gr(6,12),dt1
  end type work
  type    (work) work_data(numsat),loc_work_data(numsat)
!
  integer           adr1,adr2,m*2/6/   !   real*8     time
  integer   address,       ha_1, adr_glb_count
  integer, AUTOMATIC :: loc_ha_1, ha_beg,ha_end, kk
 POINTER(adr1,transfer_data); POINTER(adr2,work_data); POINTER(adr_glb_count,glb_counter)
          ha_beg= thread_par(3);              ha_end= thread_par(4)
       loc_ha_1  = ha_1; loc_numsat= numsat;
a: DO WHILE(.true.)
           k=  WaitForSingleObject(ha_beg,WAIT_INFINITE) ! Sabitie za puskane na thread-a
                    kk= 0;
     DO WHILE(glb_counter.LT. numsat)
    k= WaitForSingleObject(loc_ha_1,WAIT_INFINITE)
                         glb_counter= glb_counter + 1;        ! za porednija spatnik-obekt
                         loc_counter= glb_counter;            ! i se zapomni v localna za thread-a pamet
                         k= SetEvent(loc_ha_1)
             IF(loc_counter.GT. numsat) EXIT
                              kk= kk + 1; loc_adr= adr + (loc_counter-1)*adr_len
      CALL  rkfasd(loc_counter,m,transfer_data(loc_counter)%t ,transfer_data(loc_counter)%xvn, &
                                 transfer_data(loc_counter)%xvk,transfer_data(loc_counter)%dt,       &
                                transfer_data(loc_counter)%eps,RHFun,                                &
                                 work_data(loc_counter)%redmet ,work_data(loc_counter)%dt1,          &
                                 work_data(loc_counter)%in,        work_data(loc_counter)%gr,        &
                                 izbor,th_id_num,loc_adr,adr_len)
     END DO b
    k= ResetEvent(ha_beg) ! Sabitie za spirane na thread-a
    k=     SetEvent(ha_end) ! Sabitie za krai na stapkata dt v kojato thread-a uchastva
   END DO a
END SUBROUTINE  Integrator
```

Fig. 4. Code of the government subroutine of the integrator

The present integrator is part of program system for space experiments simulation which is under development [10]. The effectiveness of the SODE integrator according to numbers of processor cores, initialized threads and object will be evaluated.

```
SUBROUTINE      traekt(num_sat,t,dt,xvn,xvk,eps)
 USE  DFmt
  real*8                    t,dt,xvn(6,num_sat),xvk(6,num_sat),eps(6,num_sat)
  common   /cadr_traekt/adr1,adr2,glb_counter,numsat
… … …
              glb_counter= 0;
a: DO  i=1,num_thrd
       k= SetEvent    (ha_beg(i)) ! Events for start of threads
   END DO a
       k= WaitForMultipleObjects(num_thrd, ha_end,WaitAll,Wait_infinite)
b: DO  i=1,num_thrd
       k= ResetEvent(ha_end(i)) ! Events for waiting ending of threads
   END DO b
… … …
END SUBROUTINE  traekt
```

Fig. 5. Fragment of parent thread subroutine code, which serve for synchronization with threads of the integrator

References:

1. G e a r, C. W., The Potential for Parallelism in Ordinary Differential Equations, in Computational Mathematics, ed. Simeon Fatunla, Boole Press, Dublin, pp33-48, 1987
2. J a c k s o n, K. R., A Survey of Parallel Numerical Methods for Initial Value Problems for Ordinary Differential Equations, IEEE Transactions on Magnetics, 27 (1991), pp. 3792-3797.
3. S t o n e, L. C., S. B. Shukla, B. Neta, Parallel Satellite Orbit Prediction Using a Workstation Cluster, *International J. Computer and Mathematics with Applications*, **28**, (1994), 1–8.
4. N e t a, B., Parallel Solution of Initial Value Problems, *Proc. Fourth International Colloquium on Differential Equations*, D. Bainov, V. Covachev, A. Dishliev (eds), Plovdiv, Bulgaria, 18-23 August 1993, **2**, (1993), 19–42.
5. P h i p p s, W. E., B. N e t a, D. A. D a n i e l s o n, Parallelization of the Naval Space Surveillance Satellite Motion Model, *J. Astronautical Sciences*, **41**, (1993), 207–216.
6. N e t a, B., D. A. D a n i e l s o n, S. O s t r o m, S. K. B r e w e r, Performance of Analytic Orbit Propagators on a Hypercube and a Workstation Cluster.
7. A r o r a, N., R u s s e l l, R. P., Fast, High-Fidelity, Multi-Spacecraft Trajectory Simulation for Space Catalogue Applications, Paper S1.3, US-China Space Surveillance Technical Interchange, Beijing, China, Oct. 17-21, 2011.
8. B u r r a g e, K., Parallel methods for systems of ordinary differential equations, SIAM News, v. 26, N 5, 1993.
9. A t a n a s s o v, A., Integration of the Equation of the Artificial Earth's Satellites Motion with Selection of Runge-Kutta-Fehlberg schemes of Optimum Precision Order., Aerospace Research in Bulgaria, 21, 24-34, 2007.
10. A t a n a s s o v, A., An Adaptive Parallel Integrator of Ordinary Differential Equationa System for Space Experiment Simulation, Aerospace Research in Bulgaria, 22, pp. 59-67., 2008.
11. A t a n a s s o v, A ., Program System for Space Missions Simulation - First Stages of Projecting and Realization, Proceedings of SES 2012, xxx-xxx.