S E S 2 0 1 4
Tenth Anniversary Scientific Conference with International Participation
SPACE, ECOLOGY, SAFETY
12 – 14 November 2014, Sofia, Bulgaria

METHOD OF THREAD MANAGEMENT IN A MULTI-POOL OF THREADS ENVIRONMENTS

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: *Simulations of space missions and experiments are connected sometime with application of irregular algorithms. When it is hard to solve by conventional serial codes, is necessary to apply parallel calculations. Pool of threads program model is very flexible and convenient for cope with parallelization and load-imbalance when adaptive and irregular problem are solved.*

A situation is possible when some problem solvers (pools of threads) are initiated for solving different heterogeneous classes of problems. Every solver is based on "pool of thread" model. Because of the heterogeneity and irregularity of every class of problems the calculation time is different for every one of applied solvers. If the different solvers are started simultaneously one of them can finish before the rest. Then all threads associated with it will transit in waiting state and some processor cores will be free.

An approach for redistribution of active threads (processors) between completed solvers and the rest is realized. "Union of pools" program model is presented.

ДИНАМИЧНО ПЛАНИРАНЕ НА ИЗЧИСЛЕНИЯТА ОСНОВАНИ НА СЪСТАВНИ МОДЕЛИ

Атанас Атанасов

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

Резюме: *Симулациите на космически мисии и експерименти са свързани понякога с прилагане на иерегулярни алгоритми. Прилагането на паралелни изчисления е необходимо когато задачите са трудни за да бъдат решени с конвенционални серийни кодове. Програмният модел „пул от тредове" е гъвкав и удобен за постигане на паралелизация и ефективност зри решаване на иерегулярни изчислителни задачи.*

Възможно е един или повече изчислителни модули основани на модела „пул от тредове" да бъдат инициирени многократно в рамките на една симулация за решаване на разнородни класове от задачи. Поради разнородността на решаваните проблеми изчислителното време за всеки модул ще е различно. Ако различните модули се стартират едновременно един от тях ще свърши отредените изчисления преди останалите. Тогава всички асоциирани с него тредове ще преминат в състояние на чакане и някои процесори ще бъдат свободни.

Реализиран е подход за преразпределяне на активни тредове (процесори) между завършилите модули и останалите. Представен е програмен модел „обединение на пулове от тредове."

Introduction

The satellites mission design, analysis and control demand detailed orbital and payload simulation. Complex simulation models may contain different sub models [1, 2, 3, 4, 5, 6], and their solving is based on application of numerical methods. Some problems are irregular and multi-dimensional. An orbital and payload parameters are determined by repeatedly simulation runs on different stages of mission design. Different heavy calculations are involved on every stage of preparation and control of satellite missions. Parallelization of problem solving algorithms is modern approach for reducing processor time and cost of commercial and scientific projects.

Multi-core processors are already basic component in the modern computers and workstations. Workstations with two or four multiprocessors are on the marked already. Thereby multi-processor shared memory systems are accessible and applicable tools in scientific investigations. Many authors are occupied with development of algorithms for commercial off the shelf computers [7, 8, 9].

A parallel ordinary differential equation systems integrator for satellite motion problems was developed [10]. This integrator is possible to be started multiple times as different simultaneously working actual integrators solving different classes of problems. Besides, parallel situation problems solver was developed too [11]. Other solvers could be developed analogously based on thread parallelization and applying pool of threads.

An approach for redistribution of processor cores among parallel and simultaneously working problem solvers is proposed in the present article. It is based on “union of thread pools” model. This model allows simultaneously execution of different parallel solvers based on thread pool program model and collective using of processor cores in shared memory systems.

Parallelization based on “thread of pool” model

In many cases, one calculation problem can be divided on independent sub-problems. They can be with different difficulties and to demand different processor time for their solution. Adaptive models are possible in field of computer simulation which difficulties vary in the course of simulation time or different mathematical and physical models or calculation methods are applied depending from time varying parameters. The application of used models and methods is adapted to modeled conditions. In other cases different models are applied to different objects which participate in the simulation. An example is explained in [10] when different integration methods are used on different parts of satellite trajectories on elliptic orbits; different perturbation forces models are used for particular objects depending from high of orbits in the same time. Such type’s calculations are named irregular. We have irregular calculations too in the case of solving different situation problem, every of which is connected from different restrictions [11].

Different program models [12, 13] are used for approaching parallel calculations depending from specifics of algorithms. The aims are different sub problems or part of them to be solved simultaneously on different processors or processor cores. One such model is “pool of threads”. The set of threads are mutually synchronized. They could not solve the same sub-problem (race condition) and work to “running out of” all sub problems.

A parallel integrator for solving set of ordinary differential equations systems based on pool of treads is presented in [10]. A multiple parallel starts of this integrator in the frame of one simulation model is possible, as each integrator is initiated for solving different type of problems (satellites motion for one and space debris motion for other). The two integrators executed trajectory calculations with different difficulties for different type of objects in the frame of every integration time step. In general, consecutively execution of calculations for the two integrators is possible. A possibility for simultaneous starting of the two integrators and parallel calculations will be presented in this work.

Ideology of union of thread pools

When some solvers based on “pool of threads” parallelism are started simultaneously they compete for available processors or processor cores. The total number of threads for all solvers should be less than the number of processors. Every solver is created to solve specific kind of problems with different difficulty. By partitioning one problem is possible to be solved collectively by participation of all threads of one solver.

There are two ways for solving problems from different solvers. The first one is “one after another” in sequential mode. If one of problems is not enough difficult for respective solver, pool with less than available processors core threads is initialized. In this case the free processors will be idle. All solvers work in parallel and competitively dividing the available processors. Because of processor time for one of problem can be shorter than others, the respective solver finishes his calculations before the rests and its threads transit to waiting conditions and will be started in the next time step. The redistribution of respective processors between the rests of solvers could be possible. Then will be better released processors to be used from other solvers. It is possible if this solver has inactive threads which could be activated later on.

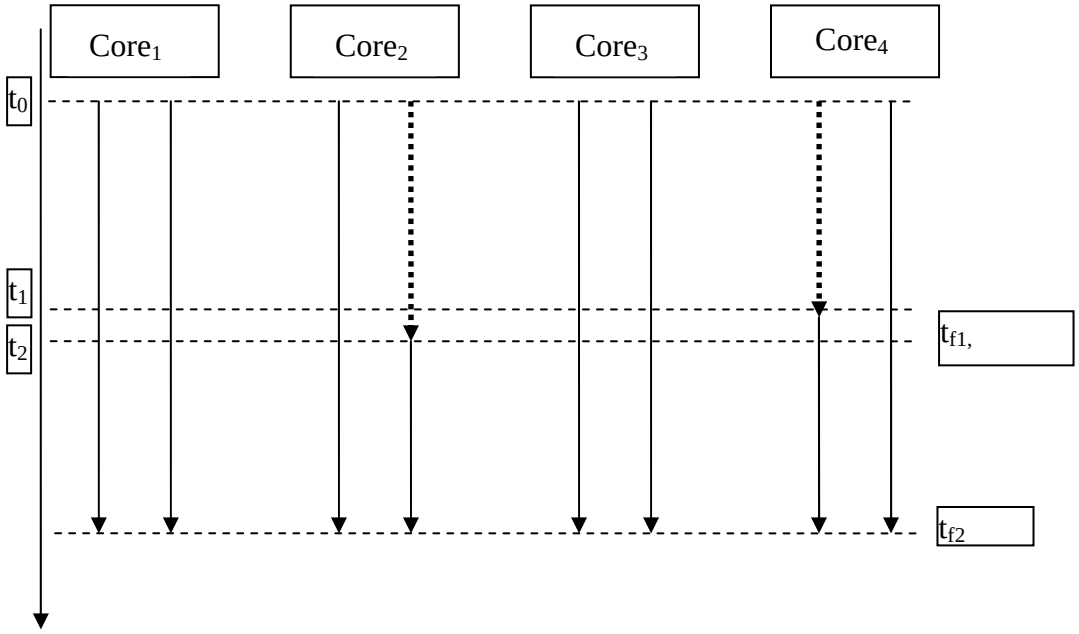

Fig. 1. An illustration of union of two pools. The broken lines present the threads of first finished solver. t_{f1} and t_{f2} are finishing times for the two solvers respectively

"Pool of thread" model is abstraction allowing dynamic scheduling of threads' execution for solving one big problem presented as set of sub-problems. The "**threads pools union**" is a higher level abstraction, which could allow joining the work of set of pools and optimizing of the entire calculation process.

Union of threads pools creation

A model "union of pools" is proposed for realization of above pointed idea, because the solvers are based on "pool of threads" model. "Union of pools" is model where the different pools are created with initial thread's number for every one of them. In the beginning only part of the threads are started, so that total number is optimal in relation to available processors. In the course of calculations, when one of solver finishes the problem its threads transit in waiting condition and could be set in suspended state to release processors (or processor cores). These processors could be redistributed in the frame of pools of the union.

Every "union of pools" is fully determined by relative attributes necessary for his control. These attributes are stored in information array. The creation of "union of pools" is possible after pool's initialization or in our case after all integrators' creation. The all pool's attributes are delivered to special subroutine InitUnionPools which store them in the information arrays of the union.

The "union of pools" creation is made by calling subroutine InitUnionPools:

```
CALL  InitUnionPools(AI_2_thread_par,num_AI_2_threads,6,AI_2_glb_counter,union_atr)
CALL  InitUnionPools(AI_1_thread_par,num_AI_1_threads,2,AI_1_glb_counter,union_atr)
   …
```

Fig. 1. A pools' union creation with two pools is shown

The meaning of actual parameters is the next:

- The first parameter "AI_2_thread_par" represent array containing the pool's parameters.
- The second parameter passes the number of threads in the pool
- The third parameter points the number of threads, which will be active from beginning- in this case 6.
- The next parameter passes the counter address of the pool. This is global variable which serve for pool control [10].
- The last parameter represent one-dimensional array with two elements. The first one is address of two-dimensional (2-D) array containing all attributes of the union of pools and the second one- number of threads for all pools.

Every next turning to union initialization routine adds additional pool. The creation of new union allocates array union_of_pools. Every column of this array contains 8 attributes for every thread of all pools, in the sequence of including the pools in union. The mean of these attributes is:

- Potentially active threads from every pool- for every potentially active thread this parameter is equal to1.
- This parameter has value 1 for all threads which initially will be active, for all the rest the parameter will be zero.
- Determines belonging of the thread to appropriate pool
- Four serial elements from each column contain threads' parameters [10].
- The last attribute contains address of variable representing counter of the pool. This counter is necessary for indexing of sub-problems for each thread and subtask distribution in the frame of the pool.

The optimal number of initial active threads for all pools must be less than number of processors or equal. From other side the number of all created threads is usually bigger than their number. Some of inactive threads of one pool could be eventually activated later on.

Dynamic pools control

The basic idea is for some pools not all created threads to be active when union is started initially. These passive threads will stay active later on when the threads of some pool become passive after finishing of the entire task.

Special function DynamicPoolsControl is developed. It controls the work of the threads of all pools included in the union_of_pools. The number of pools included in the union is determined in the beginning. After that, all threads which can initially to be activated are started and begin to execute calculations. Threads which are marked as initially inactive for some pools could be activated later on

After all initial active threads are started the parent thread transits in waiting condition using the system function:

indeks= **WaitForMultipleObjects**(num_active, ha_end,WaitOne,Wait_infinite)+1

This function waits some events for finishing of thread calculations from all pools' threads to be signaled. The number of these events is contained in variable num_active_threads and theirs handlers in array ha_end. When control variable WaitOne has value .false. then the function WaitForMultipleObjects wait for first signaled event and return the index of the element of ha_end storing this event's handler. The respective thread is marked as inactive until next call to function DynamicPoolsControl, which in our case is next step of simulation time. When all threads of particular pool finish all calculations (for current step of time) and turn into inactive condition then the respective solver completes the solving of problem. Released processors can migrate to inactive treads of other pool occupied with heavier problem. By this way numbers of active threads are no more than available processors.

Conclusion

Experiments with two actual ordinary differential equation systems integrators working simultaneously are fulfilled out on this stage. The formal character of "union of pools" model allows to be applied toward heterogenic pools used as models for different calculation tools.

Experiments with "unions of pools" with larger dimension are foreseen. Additional improvement of the "union of pools" program model is under development for including other possibilities for activation of different stages of calculations in multi-physic model simulations.

The model "union of pools" allows redistribution of released processors from finished solvers toward solvers occupied with more heavy problems in course of calculation processes. After finishing one calculation stage from respective solver other calculation stage is possible on the base of using other solver by starting its threads and engaging released processors. When one solver finishes, the system continue his work with no idle processors due to proposed approach.

The "union of threads pools" is abstraction which is possible to be applied to set of solvers in the frame of multi-physic simulations.

References:

1. Boville, B.A. and P.R., Gent. The NCAR Climate System Model, Version One. *Journal of Climate* 11.6 1998, pp. 1115-1130.

2. Craig, A.P., Jacob, R., Kauffman, B., Bettge, T., Larson, J., Ong, E., & He, Y. CPL6: The new extensible, high performance parallel coupler for the Community Climate System Model. *International Journal of High Performance Computing Applications*, *19*(3), 2005, 309-327.
3. Collins, W.D., P.J. Rasch, B.A. Boville, J.J. Hack, J.R. McCaa, D.L. Williamson, B.P. Briegleb, C.M. Bitz, S.-J. Lin, and M., Zhang. The formulation and atmospheric simulation of the Community Atmosphere Model version 3 (CAM3). *Journal of Climate* 19, no. 11, 2006, pp. 2144-2161.
4. Bernholdt, D.E., B.A., Allan, R., Armstrong, F., Bertrand, K., Chiu, T.L. Dahlgren, K., Damevski et al. A component architecture for high-performance scientific computing. *International Journal of High Performance Computing Applications* 20, no. 2, 2006, 163-202.
5. McInnes, L.C., B.A. Allan, R. Armstrong, S.J. Benson, D.E. Bernholdt, T.L. Dahlgren, L.F., Diachin et al. Parallel PDE-based simulations using the Common Component Architecture. In *Numerical Solution of Partial Differential Equations on Parallel Computers*. Springer Berlin Heidelberg, 2006, pp. 327-38.
6. Portegies, Z.,S., S., McMillan, S., Harfst, D., Groen, M., Fujii, B.Ó., Nualláin, E., Glebbeek et al. A multiphysics and multiscale software environment for modeling astrophysical systems. *New Astronomy* 14(4), 2009, 369-378.
7. Coppola, B.T., S., Dupont, K., Ring, F., Stoner, 2009. Assessing satellite conjunctions for the entire space catalog using COTS multi-core processor hardware. In: AAS 09-374, AAS-AIAA Astrodynamics Specialist Conference, Pittsburgh, August 2009.
8. Bradley, B. K., B.A. Jones, Beylkin, G. & P. Axelrad, A new numerical integration technique in astrodynamics. In *Proceedings of the 22nd Annual AAS/AIAA Spaceflight Mechanics Meeting,* 2012, pp. 1–20.
9. Escobar, D., A., Águeda, L., Martín and F.M., Martínez, Efficient ALL vs. ALL collision risk analyses. In *Advanced Maui Optical and Space Surveillance Technologies Conference*, vol. 1, 2011, p. 32.
10. Atanassov, A.M., Parallel, adaptive, multi-object trajectory integrator for space simulation applications. Advances in Space Research 54, 2014, pp. 1581–1589.
11. Atanassov, A.M., Parallel Solving of Situational Problems for Space Mission Analysis and Design, proceedings of 9th scientific conference Space Ecology Safety 2013, 283–288.
12. Rauber, T., G., Rünger, Parallel Programming: For Multicore and Cluster Systems. 2010, Springer.
13. Rünger, G., Parallel programming models for irregular algorithms. *Parallel Algorithms and Cluster Computing*. Springer Berlin Heidelberg, 2006. 3-23.

Appendix

```
SUBROUTINE     DynamicPoolsControl(union_atr)
 USE DFlib
 USE  DFmt
   integer      union_atr(2),union_of_pools(-2:5,union_atr(2))
   integer       copy_of_union[ALLOCATABLE](:,:)
   integer      ha_end[ALLOCATABLE](:)
   logical  WaitOne/.false./,WaitAll/.true./
   integer   object,ha_ends(union_atr(2))
   integer       address,union_threads_num,            adr_glob_count,glb_counter
    POINTER(address,union_of_pools); POINTER(adr_glob_count,glb_counter)
                 address= union_atr(1)
   union_threads_num= union_atr(2); num_pools= 1
   DO  i=1,union_threads_num-1
      IF(union_of_pools(0,i).NE.union_of_pools(0,i+1)) THEN  ! checking pools number &
                                num_pools= num_pools + 1             ! zeroing glb_counters for every
                           adr_glob_count= union_of_pools(5,i); glb_counter= 0 ! pool
                       ENDIF
   END DO; adr_glob_count= union_of_pools(5,union_threads_num); glb_counter= 0;
    ALLOCATE(copy_of_union(-2:5,union_atr(2))); copy_of_union= union_of_pools

 IF(num_pools.NE.1) THEN
   num_active= 0; ALLOCATE(ha_end(union_threads_num))
  a: DO  i=1,union_threads_num
        IF(union_of_pools(-1,i).EQ.1) THEN !
             k= SetEvent(union_of_pools(3,i)); num_active= num_active + 1
             ha_end(num_active)= union_of_pools(4,i);
          ENDIF
     END DO a;!ha_end(1:nth)= thread_par(1:nth)%ha_end
  p: DO WHILE(num_pools.NE.1);

        indeks= WaitForMultipleObjects(num_active, ha_end,WaitOne,Wait_infinite)+1
                   object= ha_end(indeks)
    q: DO i=1,union_threads_num
          IF(object.EQ.copy_of_union( 4,i)) THEN ! excluding of finished threat from one pool
                             copy_of_union(-2,i)= 0 ! the thread is out
```

```
                                    from_pool= copy_of_union( 0,i) !'0' contain number of pool
            r: DO  j=1,union_threads_num ! finding threat from other pool and starting
                     IF(copy_of_union( 0,j).NE.from_pool.AND.copy_of_union(-1,j).EQ.0) THEN
                                                          copy_of_union(-1,j)= 1
                   k= SetEvent(copy_of_union(3,j)) ! Start thread from other pool
                                                  num_active= 0
                                      DO  l=1,union_threads_num;
              IF(copy_of_union(-1,l).EQ.1.AND.copy_of_union(-2,l).NE.0) THEN !
                                                  num_active= num_active + 1
                     ha_end(num_active)= copy_of_union(4,l);
                     ENDIF
                                       END DO;
                                        EXIT q
                                       ENDIF
                                 END DO r
                                  EXIT q
                                ENDIF
    END DO q
              i0= 1; num_pools= 0              ! Determination the number of the left pools
 q2: DO  i=1,union_threads_num-1
        IF(copy_of_union( 0,i).NE.copy_of_union( 0,i+1).AND.i+1.LT.union_threads_num) THEN
                     num= 0
          DO  j=i0,i-1
            IF(copy_of_union(-1,j).EQ.1.AND.copy_of_union(-2,j).NE.0) THEN; num= num + 1!
          END DO; i0= i + 1; IF(num.GT.0) num_pools= num_pools + 1
       ELSEIF(copy_of_union( 0,i).EQ.copy_of_union( 0,i+1).AND.i+1.EQ.union_threads_num) THEN
                     num= 0
                          DO  j=i0,i-1
                             num= num + 1
                           ENDIF
                          END DO;          IF(num.GT.0) num_pools= num_pools + 1
        ENDIF
    END DO q2
          END DO p
                    object= WaitForMultipleObjects(num_active, ha_end,WaitAll,Wait_infinite)!+1
  s: DO  i=1,num_active !union_threads_num
       k= ResetEvent(ha_end(i)) ! Event for waiting a finish of thread
    END DO s
             DEALLOCATE(copy_of_union)
                                          ELSE ! Only one pool of threads
  c: DO  i=1,union_threads_num
             k= SetEvent(union_of_pools(3,i));
    END DO c; ha_ends= union_of_pools(4,:)
   object= WaitForMultipleObjects(union_threads_num, ha_ends,WaitAll,Wait_infinite)
  b: DO  i=1,union_threads_num
       k= ResetEvent(union_of_pools(4,i)) ! Event for waiting a finish of thread
    END DO b;
   ENDIF
END SUBROUTINE  DynamicPoolsControl
```