

PERFORMANCE EVALUATION

ABSTRACT. to be completes

1. INTRODUCTION

In order to calculate the movement of a system of N non relativistic bodies, the Newtonian equations of movement have to be solved for every particle i as a result of the interaction with all other particles in the system:

$$(1.1) \quad m_i \frac{dv_i}{dt} \big|_{x(t)} = \sum_{j=1}^N F(x_i(t), x_j(t))$$

$$(1.2) \quad \frac{dx}{dt} = v$$

where m_i is the mass, v_i is the velocity and F the force between two bodies i and j for some given initial conditions $v(0) = v_0$ and $x(0) = x_0$. Basically this means the change of the movement for a given particle i is given by the sum of all forces F between the body i and all other bodies.

2. NUMERICAL INTEGRATION OF NEWTONION TRAJECTORIES

The equation 1.1 can be solved numerically using the Euler method, which means for an given initial condition $v(0) = v_0$ and $x(0) = x_0$ the following iteration is used:

$$(2.1) \quad F_i = \sum_{j=1}^N F(x_i, x_j)$$

$$(2.2) \quad v_i(t + \Delta t) \simeq v_i(t) + \frac{1}{m_i} F_i \Delta t$$

$$(2.3) \quad x_i(t + \Delta t) \simeq x_i(t) + v_i(t) \Delta t + \frac{1}{2m_i} F_i(x_t) \Delta t^2$$

where the last term is a correction term which derived from the Taylor series (see 2.4).

$$(2.4) \quad x(t + \Delta x) = x(t) + \frac{dx}{dt} \Delta t + \frac{1}{2} \cdot \frac{d^2x}{dt^2} \Delta t^2 + Rest$$

Date: 27.8.2012.

Key words and phrases. Task Synchronization.

3. COMPUTATION TIME OF THE INTEGRATION PROCESS

The computation time T for the integration of 1.1 depends on two contributions. First on the number of particles N in a quadratic way since equation 2.1 has to be calculated for each particle and secondly on the calculation time needed to compute the movement (equation 2.3). This directly depends on the number of particles. This means the execution time of a single integration step can be calculated using the following formula.

$$(3.1) \quad T(N) = \alpha N^2 + \beta N$$

By taking measurements with different N values it is possible to derive the α and β values of equation 3.1. The result is shown in figure 3.1.

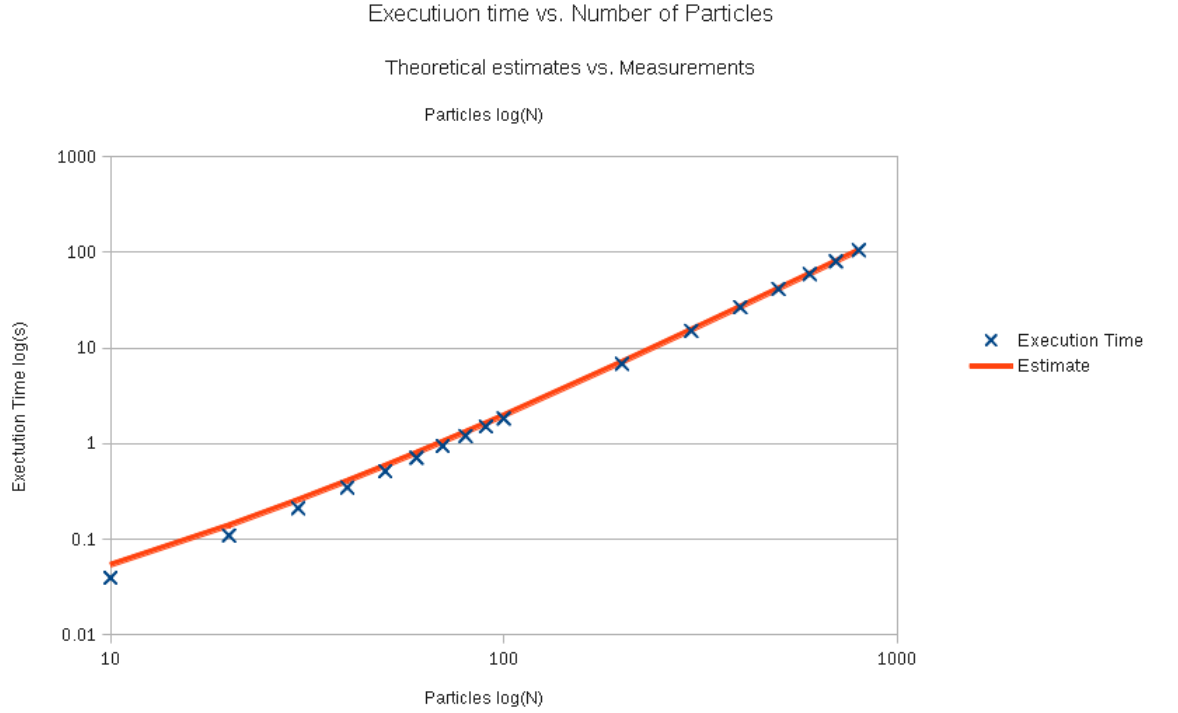


FIGURE 3.1. This figure shows the estimation of the computation time according to 3.1 in comparison with actual measurements taken using an 1.8Ghz Xeon processor running linux. The solid line shows the estimation for $\alpha = 0.0038177886s$ and $\beta = 0.0001589616s$.

4. PARALLISATION OF THE INTEGRATION

Generally speaking the integration process for n particles can be written in the form of an iteration, where $s_i = \langle x_i, p_i \rangle$ is the state ¹ of an individual particle i . All particle states together form a state vector. The new state vector s' can be calculated from the original one s by applying a function I which represents the actual Euler integration of the Newtonian equations 1.1.

$$(4.1) \quad \langle s'_1, \dots, s'_n \rangle = I(\langle s_1, \dots, s_n \rangle)$$

Since s'_i does not depend on any other s'_j it is evident that all components of s' can be calculated in parallel. This will reduce the computation time for 4.1 by a factor of $1/P$ with P for the number processes. On the other hand coordinating the iteration itself requires some processing overhead γ which can be assumed to be proportional to the .

$$(4.2) \quad T(N) = \frac{1}{P}(\alpha N^2 + \beta N) + P\gamma$$

5. THE TEST FRAMEWORK

The basic objective of the framework is to provide an algorithm based on the formula 4.1 and to study the influence of work partitioning and process synchronization on the total execution time. At the same time the test framework serves as an prototype for a new version of the PSim project.

Typically the application developer will have to implement the calculation of an individual element of the state vector by extending the `Abstract_Vector_Processor` type. At a minimum he needs to provide the methods `Process_Element` and `Collect`.

`Process_Element` is expected to contain the calculation according to 4.1 it returns false all vector components s_i have been processed. It is task of the `Process_Element` to select the items to be processed next when called. The framework will call this method until it returns false.

The method `Collect` is called by the framework after all worker tasks have finished processing elements in order to allow the application developer any type of post processing on the result vector before it is feed into the iteration again.

Upon instantiation of the `Abstract_Vector_Processor` instantiation, a given number of worker processes are started. A worker task will register with the synchronisation object and will call the method `Wait_For_Data_Ready`. Since no data has been provided yet, the call will block. If the a task of the application environment is calling the method `Execute` of the `Abstract_Vector_Processor` the waiting worker tasks will wake up start processing the state vector in parallel. Each worker task repeats calling the methods `Process_Element` until the method returns false which indicate that no more elements of the state vector have to be processed. When all work is done the worker task calls the method `Wait_For_Data_Ready` from the `Abstract_Sync` object to indicate that he is ready to process new data.

The synchronisation object is an implementation of the abstract type `Abstract_Sync` and it intended to encapsulate all platform specific of the coordination between main task and worker tasks.

¹Actually this is not the particle state in the sense of the Newtonian physics, but in the sense of the integration algorithm.

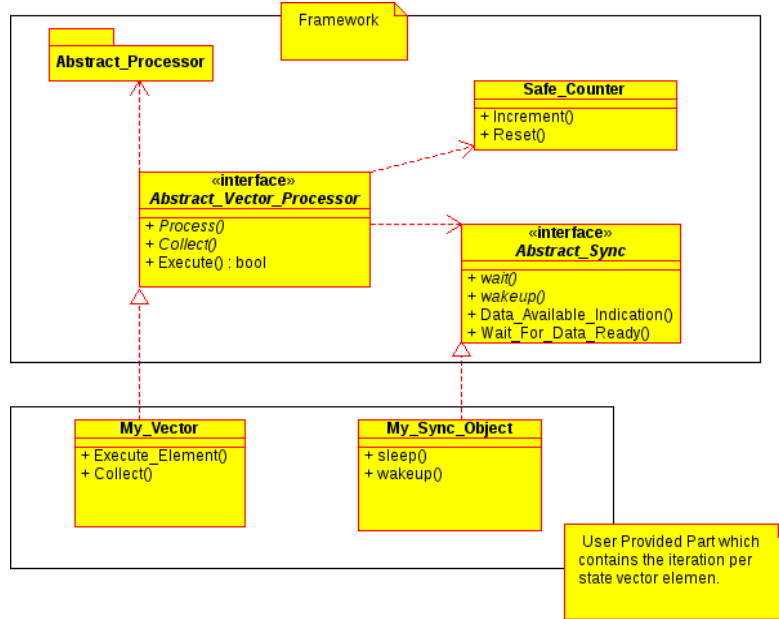


FIGURE 5.1. This Figure a shows the package and type hierarchy of the test framework. The user of the frame work has to provide two implementations. The actual iteration on the state vector and the synchronization object.

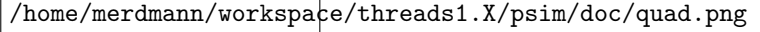
6. WORK PARTITIONING

The work in terms of vector elements has to be split up between the worker tasks. I have tested two different methods of partitioning:

- Partitioned - Each worker task works on a specific range of particles which has to be computed completely to finish the iteration.
- Scattered - Each worker task fetches from a pool the next vector element to be processed. Technically this is done by a counter which is shared among all worker processes. Each time a worker fetches a new particle for processing the worker increments this counter and interpretes the result as the next particle to be selected.

The result is shown in figure 6.1. The figure shows for a given synchronization implementation the execution time depending on the number of particles involved in the simulation for both work split mechanisms. The execution of the partitioned algorithm is systematically higher then the scattered selection if no special precautions are taken. The reason seems to be the scheduling behavior of Linux. In case of the scattered mode the average time until the next item is selected is shorter then the preemption timer. In the case of partitioning the computation of a partition may be preempted before the worker task has finished by the scheduler. This effect delays the completion of the iteration for all worker processes and hence increasing the iteration time. In order to mitigate this issue all processes which could interfere with the worker processes are moved by means of a script to processor 0 and the

worker threads are assigned to the remaining CPUs; which means in turn that the number of worker processes is 1 less then the number of available CPU's.



/home/merdmann/workspace/threads1.X/psim/doc/quad.png

FIGURE 6.1. This diagram shows the execution time vs. the number of particles for different work splits among the worker processes and different CPU assignments. In addition it shows the performance of a non parallel implementation (gray line) and the best possible execution time assuming 7 worker processes. Scattered/-Partitioned denotes the work split with the same meaning as in the text. CPU0 denotes the measurements which have been taken using a fix CPU binding as described above.

7. PROCESS SYNCHRONIZATION

Process synchronization is involved in each iteration of 4.1 because the main iteration loop is running in the main task and the iteration is done by the worker task and the main task waits for the completion of the iteration. The iteration is complete if all worker task have completed there work. As shown in 5.1 the framework provides an abstract class `Abstract_Sync` which implements a barrier based on the abstract methods `wakeup` and `sleep`.

Several different process synchronization methods have been tested as an implementation of of the abstract methods `wakeup` and `sleep`.

- Suspension Objects
- Protected Types

In the following we will discuss shortly the results and with respect to testability and latency time. The code fragments shown in the following are some what simplified to show only the important implementation details. We will use the implementation using suspension objects to explain some of the inner workings of the abstract process synchronization package.

During start up all worker threads are registering with the synchronization object with a so called client id. If the main threads indicates that all data has been provided to perform an iteration the method `Indicate_Input_Available` the synchronization object will wakeup all worker threads using the provided wakeup method and will wait for the completion of all worker threads until it returns to the caller in the main thread.

The details of `Set:True` and `Suspend_Until_True` are hidden in the Ada 95 standard library. Protected types and barriers can be used to implements the same functionality as shown in algorithm 1 is a rather straight forward way.

Figure 7.2 shows the ratio of the execution time for different numbers of particles. For larger numbers the ratio is 1 which means neither algorithm 2 nor algorithm 1 is better to use from the performance perspective. Other selection criteria like maintenance effort and portability are out of scope here.

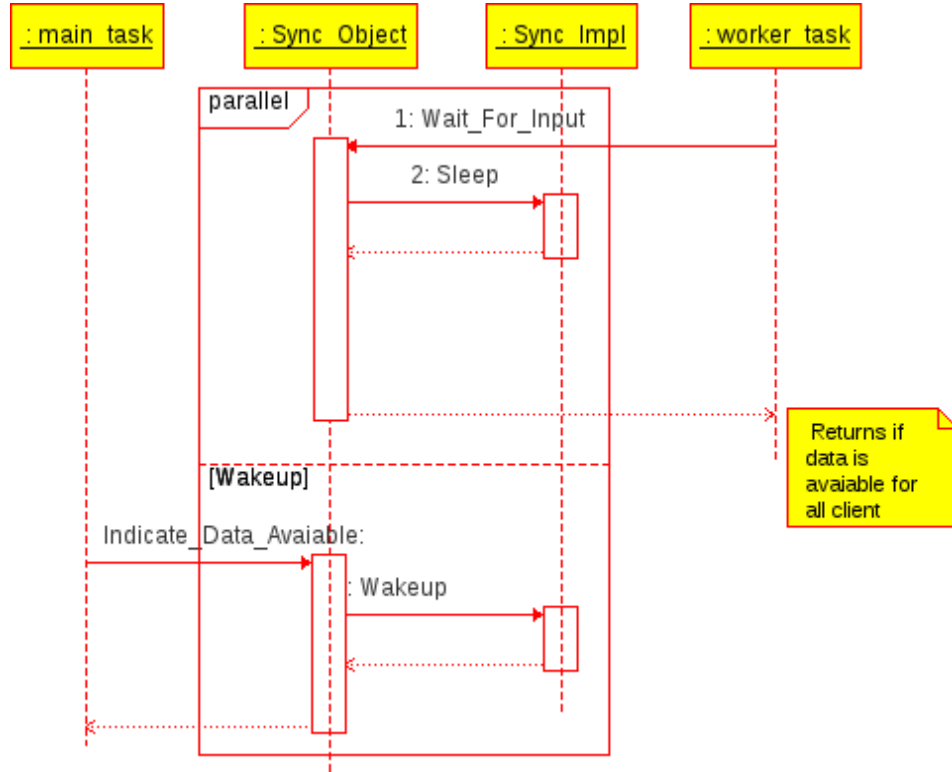


FIGURE 7.1. This figure shows the inter working between the main task which actually calls the calculate procedure to execute the iteration process for n times. The diagram above describes the sequence of interactions for one iteration.

Algorithm 1 Process synchronization using the Suspension Objects from Ada95.

```

procedure Wakeup( This : in out Object_Type; Id : in Client_Id_Type ) is
    Data : Object_Data_Access renames This.Data;
begin
    Set_True( Data.S(Id) );
end Wakeup;

procedure Sleep( This : in out Object_Type; Id : in Client_Id_Type ) is
    Data : Object_Data_Access renames This.Data;
begin
    Suspend_Until_True( Data.S(Id) );
    Set_False( Data.S(Id) );
end Sleep;
  
```

Algorithm 2 Protected

```

protected body Sleep_Type is
    entry Sleep when Ready is
    begin
        Ready := False;
    end Sleep;

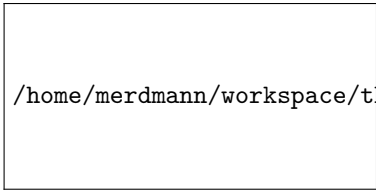
    procedure Wakeup is
    begin
        Ready := True;
    end Wakeup;
end Sleep_Type;

type Object_Data_Type( Max_Worker : Client_Id_Type ) is record
    S : Sleep_Array( 0..Max_Worker ) ;
end record;

procedure Wakeup( This : in out Object_Type; Id : in Client_Id_Type ) is
    Data : Object_Data_Access renames This.Data;
begin
    Data.S(Id).Wakeup;
end Wakeup;

procedure Sleep( This : in out Object_Type; Id : in Client_Id_Type ) is
    Data : Object_Data_Access renames This.Data;
begin
    Data.S(Id).Sleep;
end Sleep;

```



/home/merdmann/workspace/threads1.X/psim/doc/syncratio.png

FIGURE 7.2. This diagram shows the ratio of the execution times using the suspension objects and protected types as synchronization mechanisms for increasing numbers of particles. For larger numbers there is virtually no difference in using either one.

8. OPTIMIZATION TO THE INTEGRATION ALGORITHM

Even though this partitioning of the computational work is technically quite easy to achieve the actual yield in performance depends only linear on the number of parallel processes. The most reasonable target for parallelization is the quadratic term. The force F_i acting on a particle i is the sum of the force caused by all other particles F_{ij} where $i \in [1..N]$ and $i \neq j$.

$$F_1 = 0 + F_{12} + F_{13} + F_{14} + \dots + F_{1N}$$

$$F_2 = F_{21} + 0 + F_{23} + F_{24} + \dots + F_{2N}$$

$$F_3 = F_{31} + F_{32} + 0 + F_{34} + \dots + F_{3N}$$

$$F_4 = F_{41} + F_{42} + F_{43} + 0 + \dots + F_{4N}$$

.....

$$F_N = F_{N1} + F_{N2} + F_{N3} + F_{N4} + \dots + 0$$

Considering $F_{ij} = -F_{ji}$ and $F_{ii} = 0$ it becomes obvious from the schema above that while calculating the value of F_1 the first term of the sum for F_i where $i > 1$ are as well calculated. More generally speaking; while calculating the terms for F_i terms for all F_j where $j > i$ are calculated as well. This observation motivates the following algorithm to calculate the forces F_i for all particles $i \in 1..N$.

Algorithm 3 Using $F_{ij} = -F_{ji}$ the calculation of the forces (see 2.1) where the computational time depends on N^2 can be simplified to an algorithm where the computation time depends on $(N^2 - N)/2$.

```

for i in 1..N loop
  F(i) := 0.0;
  for j in i+1..N loop
    F(i) := F(i) + F(i, j);
    F(j) := F(j) - F(i, j);
  end loop
done

```

The algorithm 3 requires the following number of computation steps:

$$(8.1) \quad M = (N - 1) + (N - 2) + (N - 3) + \dots + N = N^2 - 1 - 2 - 3 \dots$$

using the Gaussian formula this yields:

$$M = \sum_{k=1}^N (N - k) = N^2 - \sum_{k=1}^N k = N^2 - N(N + 1)/2 = N^2 - \frac{N^2}{2} - N/2 = \frac{N^2 - N}{2}$$

to be complete with measurement results

9. COMPARISON WITH OTHER FRAMEWORKS

10. CONCLUSION

Since figure 6.1 shows that the execution time comes for large numbers quite near to the theoretical limit of the algorithm it makes for time being no sense to invest more efforts in tuning the synchronization mechanisms. For small numbers of particles the no parallel implementation should be used since it yields faster results then the parallel implementation of this algorithm.

REFERENCES

- [1] As an example; Reiner Oloff, *Geometrie der Raumzeit*, Vieweg, ISBN 3-528-26917-0
- [2] *Ta-Pei Chend, Relativity, Gravitation and Cosmology; Oxford Master Series, ISBN 0-19-852957-0.*
- [3] American Mathematical Society,
- [4] American Mathematical Society,

NO ADDRESS, SOMEWHERE, SOMEWHERE ZIP NO ADDRESS, SOMEWHERE, SOMEWHERE ZIP

Current address: Home

E-mail address: `erdmann-berlin@t-online.de`

Current address: Home

E-mail address: `erdmann-berlin@t-online.de`

URL: <http://michaelerdmann-berlin.de/>