The N-Body Pipeline

Per Brinch Hansen
Syracuse University, School of Computer and Information Science, pbh@top.cis.syr.edu

SU-CIS-91-09

THE N-BODY PIPELINE

PER BRINCH HANSEN

March 1991

School of Computer and Information Science
Suite 4-116
Center for Science and Technology
Syracuse, New York 13244-4100

(315) 443-2368

THE N-BODY PIPELINE[1]

PER BRINCH HANSEN

School of Computer and Information Science
Syracuse University
Syracuse, New York 13244

March 1991

Abstract. A general pipeline for all-pairs computations is adapted for direct force summation of n bodies which interact through gravitation only. To achieve approximate load balance the pipeline is folded several times across an array of processors. The performance of the pipeline is analyzed and measured on a Computing Surface.

Keywords. Pipeline algorithms, n-body simulation.

1. Introduction

We describe the use of a programming paradigm to solve the n-body problem on a parallel computer. An n-body simulation computes the trajectories of n bodies which interact through gravitational forces only. At discrete time intervals, the algorithm computes the forces on the bodies and adjusts their velocities and positions [5]. Similar algorithms are used in computational fluid dynamics and molecular dynamics [8].

The parallel algorithm presented here uses a pipeline to compute the forces between all pairs of bodies. For $n \leq 10000$ the direct method of force summation allows very accurate simulations of star clusters [13]. For larger systems, the approximate algorithm of Barnes and Hut is much faster [1,11].

The complexity analysis of the n-body pipeline is based on a simple wavefront model of the computation. The pipeline is as fast as the standard ring algorithms [4,6,9]. On a Computing Surface with 45 transputers the pipeline computes the forces on 9000 bodies in 72 s with an efficiency of 79%.

The pipeline algorithm was *not* developed for n-body simulation. It was derived from a general pipeline for all-pairs computations by a trivial substitution of types, variables, and procedure statements. A similar pipeline has been used to solve linear equations by means of Householder reduction [2,3].

[1]Copyright©1991 Per Brinch Hansen

2. Force summation

We consider each body to be a point mass in three-dimensional space. The state of a body is defined by its mass m and three vectors representing its position r, its velocity v, and the total force f by which the other bodies attract the given body.

Figure 1 shows two bodies p_i and p_j with masses m_i and m_j and positions r_i and r_j relative to the origin 0.

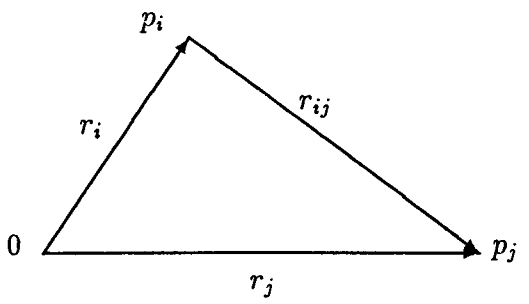

Fig. 1 Two bodies in space.

The body p_j attracts p_i with a force f_{ij}. The magnitude of the force is proportional to the mass of each body and inversely proportional to the square of the distance between the bodies

$$|f_{ij}| = \frac{Gm_i m_j}{|r_{ij}|^2}$$

G is the universal gravitational constant.

The distance $|r_{ij}|$ is the length of the vector

$$r_{ij} = r_j - r_i$$

Newton's law of gravitation can also be stated as a vector equation

$$f_{ij} = |f_{ij}| e_{ij}$$

where e_{ij} is a unit vector in the same direction as r_{ij}

$$e_{ij} = \frac{r_{ij}}{|r_{ij}|}$$

The body p_i attracts p_j with a force f_{ji} of the same magnitude in the opposite direction of f_{ij}

$$f_{ji} = -f_{ij}$$

This is Newton's third law.

The total force f_i acting on the body p_i is the sum of all forces exerted on p_i by the other $n-1$ bodies

$$f_i = \sum_{j \neq i} f_{ij}$$

3. Time integration

Since we are interested in parallel programming rather than astrophysics we will use a simple integration method.

The acceleration a_i of a body p_i is determined by its mass m_i and the total force f_i acting on it according to Newton's second law

$$a_i = \frac{f_i}{m_i}$$

During a small time interval Δt the acceleration a_i is approximately constant. Consequently the velocity v_i of the body increases by

$$\Delta v_i = a_i \Delta t$$

At the same time, the position r_i of the body increases by

$$\Delta r_i = \int_0^{\Delta t} (v_i + a_i t)dt = v_i \Delta t + 0.5 a_i \Delta t^2$$

In other words

$$\Delta r_i = (v_i + 0.5\Delta v_i)\Delta t$$

More accurate integration algorithms are discussed in [12]. The first order discrete mechanics method of Greenspan is particularly effective [10].

In a close encounter between two bodies the accelerations can become extremely large. To prevent numerical instability it may be necessary to make the time step so small that a realistic simulation becomes prohibitively time-consuming. Various techniques are used to deal with near-collisions [8,13]. In our performance measurements we avoid the problem by making the initial mass density of the system sufficiently small.

4. Sequential algorithm

A Pascal program for n-body simulation uses the following data types

```
type axis = (x, y, z);
     vector = array [x..z] of real;
     body = record m: real; r, v, f: vector end;
     system = array [1..n] of body;
```

Each time step dt involves a force summation followed by a time integration of all bodies (Algorithm 1). We have omitted the definition of the zero vector.

```
procedure timestep(var p: system; dt: real);
var i, j: integer;
begin
  for i := 1 to n do p[i].f := zero;
  for i := 1 to n − 1 do
    for j := i + 1 to n do addforces(p[j], p[i]);
  for i := 1 to n do movebody(p[i], dt)
end
```

Algorithm 1

Algorithm 2 defines the forces between two bodies pi and pj. The magnitudes of the distance vector rij and the force vector fij are denoted rm and fm, respectively. The component of fij along an axis k is called fk.

```
procedure addforces(var pi, pj: body);
var k: axis; rij: vector; fk, fm, rm: real;
begin
  for k := x to z do rij[k] := pj.r[k] − pi.r[k];
  rm := sqrt(sqr(rij[x]) + sqr(rij[y]) + sqr(rij[z]));
  fm := G*pi.m*pj.m/sqr(rm);
  for k := x to z do
  begin
    fk := fm*(rij[k]/rm);
    pi.f[k] := pi.f[k] + fk;
    pj.f[k] := pj.f[k] − fk
  end
end
```

Algorithm 2

Algorithm 3 defines the time integration of a body pi. The velocity increment along an axis k is denoted dvk.

```
procedure movebody(var pi: body; dt: real);
var k: axis; dvk: real;
begin
  for k := x to z do
  begin
    dvk := (pi.f[k]/pi.m)*dt;
    pi.r[k] := pi.r[k] + (pi.v[k] + 0.5*dvk)*dt;
    pi.v[k] := pi.v[k] + dvk
  end
end
```

Algorithm 3

5. Pipeline algorithm

Force summation and time integration involve $0(n^2)$ and $0(n)$ operations, respectively. Since summation is the most time-consuming part of the computation, we will use a pipeline to speed it up. The must faster time integration will remain sequential.

Figure 2 shows the pipeline controlled by a master process.

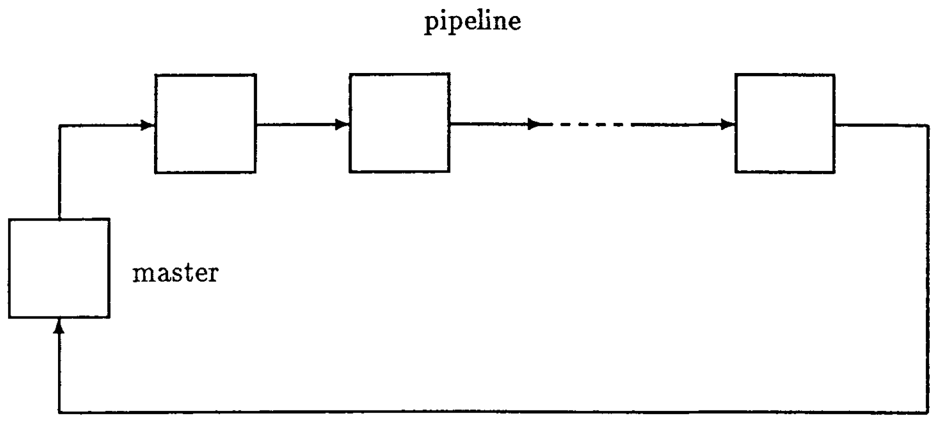

Fig. 2 Master and pipeline.

The parallel processes will be defined in Pascal extended with statements for message communication. Each process has an input channel and an output channel. The input and output of a body pi are denoted

inp?pi   out!pi

For each time step, the master initializes all forces to zero and sends the bodies one at a time through the pipeline. The pipeline sums the forces and returns the bodies to the master which performs the integration (Algorithm 4). (The all-pairs pipeline was designed to output the results in reverse order to facilitate back substitution after matrix reduction [2].) Notice also that it is unnecessary to send velocities through the pipeline.

```
procedure timestep(var p: system; dt: real;
  inp, out: channel);
var i: integer;
begin
  for i := 1 to n do
  begin p[i].f := zero; out!p[i] end;
  for i := n downto 1 do
  begin inp?p[i]; movebody(p[i], dt) end
end
```

Algorithm 4

During a force summation the first $n - 1$ bodies are distributed evenly among the nodes of the pipeline. The last body goes through the pipeline without being stored.

Algorithm 5 defines the force summation performed by a node which holds a block of bodies with indices from s to t, where $1 \leq s \leq t \leq n - 1$. To suppress irrelevant detail we extend Pascal with dynamic array bounds.

```
procedure node(s, t: integer; inp, out: channel);
var p: array [s..t] of body;
  pj: body; i, j: integer;
begin
  for i := s to t do
  begin
    inp?p[i];
    for j := s to i − 1 do addforces(p[i], p[j])
  end;
  for j := t + 1 to n do
  begin
    inp?pj;
    for i := s to t do addforces(pj, p[i]);
    out!pj
  end;
  for i := t downto s do out!p[i];
  for j := s − 1 downto 1 do
  begin inp?pj; out!pj end
end
```

Algorithm 5

First, the node inputs and stores its own bodies. All subsequent bodies then pass through the node. Each body interacts with the previously stored bodies. After the force computation, the node outputs its own bodies. Finally all bodies stored in previous nodes pass through the node.

We emphasize again that the n-body pipeline was derived from a general pipeline for all-pairs computations [2].

6. Load balancing

The number of force calculations performed by each node drops linearly from the first to the last node. If each node runs on a separate processor, the uneven distribution of the computational load will force the first processor to do most of the work. As we will show later, this reduces the speed up of the parallel computation significantly.

To achieve approximate load balancing we fold the pipeline m times across an array of p processors [3]. Figure 3 shows the folded pipeline with $(m+1)p$ nodes. Each processor runs $m+1$ nodes, each holding q bodies. The effect of the folding is to reduce the computing time of the first (and most time-consuming) node by reducing the block length q.

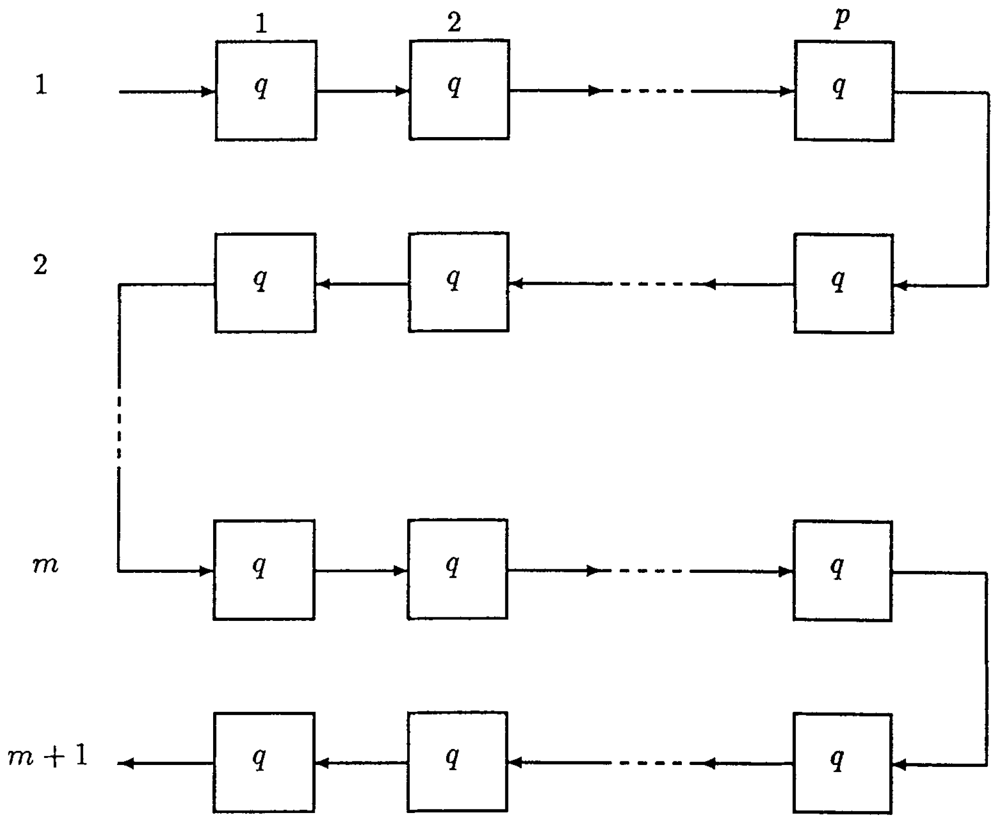

Fig. 3 Folded pipeline.

In the initial analysis of the force computation we ignore process communication and assume that the forces between two bodies are computed in unit time.

The force computation travels along the pipeline like a wave front filling one node at a time with bodies. Since the pipeline is folded, the wave sweeps back and forth across the array of processors $m + 1$ times. Each sweep fills another level of nodes.

Initially all nodes are empty. We will say that a node is full when it holds q local bodies and a single additional body. We assume that the additional body already has interacted with the local bodies and is ready to be output.

First, processor 1 fills node 1. The first body does not interact with any other body. The second body interacts with the first one, and so on. So node 1 accumulates $q + 1$ bodies in time

$$\sum_{k=0}^{q} k = 0.5q(q+1)$$

While processor 2 fills node 2, the bottleneck is processor 1. Initially processor 1 is ready to output the additional body in node 1 in time zero (since we ignore communication time). Each of the following q bodies must interact with the q local bodies in node 1 before they can be output to node 2. Finally the last body input by node 2 must interact with the q local bodies input previously. So node 2 is filled in time $q(q + 1)$.

Processor 3 then fills node 3 in the same amount of time, and so on. Consequently the wave sweeps through the first p nodes in time

$$t_1 = 0.5q(q+1) + (p-1)q(q+1)$$

which can be reduced to

$$t_1 = (p - 0.5)q(q+1)$$

We now assume that the first $k - 1$ levels of nodes are full and observe the wave sweeping through level k. When a given body enters a node at level k, each of the previous nodes must transfer another body to keep these nodes full and propagate the wave.

While level k is being filled with $p(q+1)$ bodies, each processor transfers the same number of bodies through each of the previous $k - 1$ levels. Each processor must therefore spend $p(q + 1)(k - 1)q$ time units to move the wave through the previous levels. In addition the wave must sweep through the new level. So level k is filled in time

$$t_k = p(q+1)(k-1)q + t_1$$

Consequently the k-th sweep takes the time

$$t_k = (kp - 0.5)q(q+1)$$

The total force calculation time is the time required to complete $m + 1$ sweeps

$$T_p = \sum_{k=1}^{m+1} t_k$$

If the block length q is large we have approximately $q(q+1) \approx q^2$ and

$$T_p \approx 0.5(m+1)((m+2)p-1)q^2 \quad \text{for } q \gg 1$$

For a large problem the block length is approximately

$$q \approx \frac{n}{(m+1)p} \quad \text{for } n \gg 1$$

The parallel computing time can now be expressed as

$$T_p = 0.5(1+f)n^2/p$$

where

$$f = \frac{1-1/p}{m+1}$$

is a measure of the computational imbalance.

The complexity analysis exaggerates the computing time a bit. The problem is that we have followed the progression of a wave that always leaves an extra body in each node. When the wave reaches the end of the pipeline, each node still contains an extra (non-existing) body.

In short, the analysis assumes that the pipeline inputs n bodies plus an additional $(m+1)p$ bodies. The effect of the phantom bodies can be ignored if there are very few of them, that is, if

$$n \gg (m+1)p$$

This means that the analysis is accurate, provided $q \gg 1$ (an assumption we have already made).

7. Performance

During a force calculation n messages representing the bodies pass through $m+1$ nodes in each processor. The communication increases the parallel run time of a force calculation to

$$T_p = a(1+f)n^2/p + b(m+1)n$$

where a and b are system-dependent constants for force calculation and communication, respectively. We can ignore the time integration which is a much faster computation running in parallel on a separate master processor.

If the folded pipeline runs on a single processor only (where $p=1$ and $f=0$) the run time is

$$T_1 = an^2 + b(m+1)n$$

The efficiency of the parallel computation is

$$E_p = T_1/(pT_p)$$

We reprogrammed the n-body pipeline in occam and ran it on a Computing Surface with T800 transputers using 64-bit arithmetic. Measurements show that

$$a = 31.8\ \mu s \qquad b = 27.5\ \mu s$$

Table 1 shows measured run times for a pipeline with 45 processors which performs a force calculation for 9000 bodies. The predicted run times are shown in parentheses. If the pipeline is not folded ($m = 0$), the efficiency is 0.50 only. Folding the pipeline three times increases the efficiency to 0.79.

Table 1

p	n	m	$T_p\ (s)$	$E_p\ (est)$
45	9000	0	110 (113)	0.50
45	9000	1	84 (86)	0.67
45	9000	3	72 (72)	0.79

To reduce the effect of communication, we use a large block size $q \gg b/a$. This makes the communication time negligible compared to the computing time. For $m = 3$, $q \gg 1$ and $p \gg 1$ it should be possible to achieve an efficiency $E_p \geq 0.79$.

Table 2 shows measured (and predicted) run times of a force calculation performed by a pipeline which is folded three times. In each experiment the block size $q = 50$. For $m = 3$ this means that $n/p = 200$. By scaling the problem size n in proportion to the computer size p we maintain an almost constant efficiency of 0.79 to 0.81 (see also [7]).

Table 2

p	n	$T_p\ (s)$	$E_p\ (est)$
1	200	1.3 (1.3)	1.00
10	2000	15.8 (15.8)	0.81
20	4000	31.9 (31.9)	0.80
30	6000	48.0 (48.0)	0.79
40	8000	64.2 (64.2)	0.79

8. Final remarks

We have presented and analyzed a parallel algorithm for direct force summation of n bodies which interact through gravitation only. The algorithm automatically avoids self-interactions and takes advantage of the symmetry of the force calculations.

The parallel algorithm illustrates the benefits of developing programming paradigms which can be adapted to different applications. The n-body pipeline was derived from a general pipeline for all-pairs computations. A similar pipeline has been used to solve linear equations by means of Householder reduction.

Acknowledgement

I thank Jonathan Greenfield for helpful comments.

References

[1] J. Barnes and P. Hut, A hierarchical $0(N \log N)$force-calculation, *Nature* **324** (1986) 446–449.

[2] P. Brinch Hansen, The all-pairs pipeline, Technical Report, Syracuse University, 1990.

[3] P. Brinch Hansen, Balancing a pipeline by folding, Technical Report, Syracuse University, 1990.

[4] H. R. P. Ellingworth, Transputers and computational chemistry: an application, *Supercomputing* '88 (1988) 269–274.

[5] R. Feynman, R. B. Leighton and M. L. Sands, *The Feynman Lectures on Physics*, Vol. I, (Addison-Wesley, Reading, Massachusetts, 1989).

[6] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon and D. W. Walker, *Solving Problems on Concurrent Processors*, Vol. I (Prentice-Hall, Englewood Cliffs, New Jersey, 1988).

[7] J. L. Gustafson, Reevaluating Amdahl's law, *Comm. ACM* **31** (1988) 532–533.

[8] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles* (Adam Hilger, New York, 1988).

[9] J. Li, A. Brass, D. J. Ward and B. Robson, A study of parallel molecular dynamics for N-body simulations on a transputer system, *Parallel Computing* **14** (1990) 211–222.

[10] A. Marciniak, *Numerical Solutions of the N-body Problem* (D. Reidel, Dordrecht, Holland, 1985).

[11] J. K. Salmon, Parallel hierarchical N-body methods, Ph.D. Thesis, California Institute of Technology, 1991.

[12] R. Smith and D. E. Harrison, Jr., Algorithms for molecular dynamics simulations of keV particle bombardment, *Computers in Physics* **3** (1989) 68–73.

[13] T. S. van Albada, Models of hot stellar systems, *The Use of Supercomputers in Stellar Dynamics* (Springer-Verlag, New York, 1986) 23–35.