High performance N-body simulation on computational grids

Groen, D.J.

Publication date
2010

Document Version
Final published version

Citation for published version (APA):
High Performance N-body Simulation
on Computational Grids

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam,
op gezag van de Rector Magnificus Prof. dr. D.C. van den Boom,
ten overstaan van een door het college voor promoties ingestelde
commissie, in het openbaar te verdedigen in de Agnietenkapel
op dinsdag 16 november 2010, te 14.00 uur.
door

Derk Jan Groen

geboren te Heemstede.
Promotiecommissie:

Promotor: Prof.dr. S.F. Portegies Zwart
Co-promotores: Prof.dr. P.M.A. Sloot
              Prof.dr. E.P.J. van den Heuvel
Overige leden: Prof.dr. R.H. Bisseling
              Prof.dr. M. Bubak
              Dr. A.G. Hoekstra
              Prof.dr. S.L.W. McMillan
              Prof.dr. R.A.M.J. Wijers
Contents

1 Introduction
   1.1 Simulating $N$-body systems .............................................. 2
      1.1.1 Direct-method integration methods ................................ 2
      1.1.2 Alternative $N$-body integration methods ....................... 4
   1.2 High-performance $N$-body simulations ................................. 4
      1.2.1 Parallelization and speedup ....................................... 5
      1.2.2 Communication in parallel simulations ............................ 7
      1.2.3 $N$-body simulations using special-purpose hardware ........... 7
   1.3 Distributed infrastructures and computational grids ................ 8
   1.4 Thesis overview .......................................................... 9
      1.4.1 Chapter 2 .......................................................... 10
      1.4.2 Chapter 3 .......................................................... 10
      1.4.3 Chapter 4 .......................................................... 10
      1.4.4 Chapter 5 .......................................................... 11
      1.4.5 Chapter 6 .......................................................... 12

2 Distributed Direct $N$-body Simulations on a Global Grid of GRAPEs 13
   2.1 Introduction .............................................................. 13
   2.2 Experiment setup ......................................................... 15
   2.3 Results of grid calculations ............................................ 17
      2.3.1 Timing results of $N$-body calculations ......................... 17
      2.3.2 Profiling of the $N$-body simulations ........................... 18
   2.4 Modelling the performance of the grid ................................ 19
      2.4.1 Single PC .......................................................... 20
      2.4.2 Grid of PCs with copy algorithm ................................ 22
      2.4.3 Grid of PCs with ring algorithm ................................. 23
5.4.2 DAS-3 results ................................................. 65
5.4.3 Gravitational Billion Body Project experiment setup .......... 68
  5.4.3.1 Network configuration .................................. 69
5.4.4 GBBP results ............................................... 70
5.5 Scalability of N-body simulations across supercomputers .......... 77
  5.5.1 Speedup and efficiency predictions for TreePM simulations .... 77
  5.5.2 Speedup and efficiency predictions for tree and direct-method simulations ................................................. 79
    5.5.2.1 Performance model for the tree algorithm .............. 81
    5.5.2.2 Modelling of block time steps ......................... 81
    5.5.2.3 Predictions ......................................... 81
  5.5.3 Bandwidth analysis for cosmological simulations .............. 83
5.6 Conclusion ..................................................... 84

6 A Light-Weight Communication Library for Distributed Computing ... 87
  6.1 Introduction .................................................. 87
  6.2 Related work .................................................. 88
  6.3 Architecture of MPWide ....................................... 88
    6.3.1 Design .................................................. 88
      6.3.1.1 Data transport in the wide area network ............ 89
      6.3.1.2 Functionality and programming interface .......... 89
    6.3.2 Forwarder .............................................. 91
    6.3.3 Implementation ......................................... 92
  6.4 Benchmarking MPWide .......................................... 92
    6.4.1 Results .................................................. 93
      6.4.1.1 Local tests ....................................... 93
      6.4.1.2 National tests .................................. 94
      6.4.1.3 International tests ............................... 95
  6.5 Testing performance in a production environment .................. 95
    6.5.1 Test experiments ........................................ 96
      6.5.1.1 Results on DAS-3 Dutch grid ....................... 97
      6.5.1.2 Results on Amsterdam and Helsinki supercomputers ... 98
      6.5.1.3 Results on Amsterdam and Tokyo supercomputers ..... 98
    6.5.2 Production ............................................ 99
  6.6 Conclusions and future work .................................. 99

7 Conclusions ....................................................... 103
  7.1 Thesis summary ............................................... 103
  7.2 Conclusions and Recommendations ................................ 105
    7.2.1 Simulations on a planet-wide distributed supercomputer .... 105
    7.2.2 Recommendations for planet-wide N-body simulations ........ 107
    7.2.3 Creating a planet-wide distributed supercomputer .......... 107
      7.2.3.1 Configuration of a global supercomputer ............ 108
      7.2.3.2 Scheduling and running on a global supercomputer ... 109
7.2.4 Future perspectives and applications 110

Nederlandse Samenvatting 111
Acknowledgements 115
List of Publications 119
Bibliography 123
In this thesis we study the simulation of gravitational $N$-body systems on infrastructures that span across the globe. We give some background information on the subject matter of this thesis in the first part of this section, and an overview of the contents of this thesis in the second part.

Gravity is the dominant force that shapes the universe. It causes objects with mass to attract each other and has a much longer range than the other three fundamental forces (strong interaction, electromagnetic force and weak force). The movements of planets, stars and galaxies are therefore driven by gravity. By understanding the force of gravity, we learn about the dynamical evolution of the solar system, star clusters, galaxies and the universe as a whole. This challenge in understanding the movements of celestial bodies is commonly referred to as the $N$-body problem.

On July 5th 1687, Isaac Newton proposed laws of motion for classical mechanics, as well as a law of gravity in his “Philosophiae Naturalis Principia Mathematica”. This law defines the gravitational force $F_i$ on a particular body $i$ as a function of the positions and masses of all other bodies, and is given by

$$F_i = m_i a_i = m_i G \sum_{j \neq i}^N \left( m_j \frac{r_{ij}}{|r_j - r_i|^3} \right). \quad (1.1)$$

Here $G$ is the gravitational constant, and $m_j$ and $r_j$ are respectively the mass and position of a given body $j$. Although the motion of planets in the Solar system could be reproduced using Kepler’s laws of planetary motion [71], Newton’s law of gravity introduced a generalized concept of mutual attraction between objects with mass. Using this law it is possible to accurately calculate the movements of celestial bodies in systems with a moderate gravitational field. In the presence of strong gravitational interactions, however, the theory no longer provides correct results.
Between 1907 and 1915, Albert Einstein developed the theory of general relativity, which describes gravity as a geometric property of both space and time. This theory can be applied to accurately calculate the motions of \( N \)-body systems with a strong gravitational field and it provides explanations on various astrophysical phenomena, such as gravitational lensing [137, 135] and the existence of black holes [116]. Einstein's theory of relativity has also been applied to support the Big Bang [75], a widely accepted cosmological model that describes the origin and the expansion of the universe.

The breakthroughs achieved by Newton and Einstein provided tools to tackle the \( N \)-body problem, as scientists were now able to write down the equations of motion for interacting celestial bodies. By integrating these equations over time it became possible to model the evolution of an \( N \)-body system. This integration cannot be performed analytically for \( N > 2 \). However, we can use a numerical integration method to simulate the dynamics of larger \( N \)-body systems. These numerical integrations provide approximate answers and can be applied to systems of any size, if sufficient resources to perform the required computations are available.

1.1 Simulating \( N \)-body systems

The first known attempt to reproduce the movement of stars through numerical integration was a brief simulation of a three body problem performed on paper by Str"omgren [122].

About 30 years later, the first known \( N \)-body simulation consisting of more than a few bodies using a numerical integration scheme was performed by Erik Holmberg in 1941 [63]. He used two sets of 37 light bulbs to simulate the gravitational dynamics of two nebulae (see Fig. 1.1). Such analogue simulations required a vast amount of manual computations and man hours, and were therefore difficult to reproduce.

With the advent of digital computers in the 1950s and 60s, it became considerably easier to numerically integrate \( N \)-body systems. Since computers have the capability to perform arithmetic operations and handle data, it became possible to map celestial bodies to digital point particles and perform \( N \)-body simulations electronically. The first computer simulations were performed in the 1960s by von Hoerner (1960 [130]), Aarseth (1963 [1]) and van Albada (1968 [126]) and were still proportional in size to Holmbergs acoustic simulation. These simulations used direct-method integration, a method where the equations of motion are solved between all particle pairs. Several production-quality direct-method \( N \)-body codes have been developed in the subsequent decades. Examples include the NBODY1 to NBODY7 codes developed by Aarseth [3], the kira integrator in the Starlab environment [110] and phiGRAPE [50].

1.1.1 Direct-method integration methods

Direct method \( N \)-body simulations are highly accurate due to the explicit calculation of the force exchanges between each pair of particles. The high accuracy of direct-method simulations makes them well suited for modelling tightly interacting stellar systems such as dense star clusters (e.g., see [128, 16, 107, 111, 129]) and stars near the Galactic
1.1 Simulating $N$-body systems

Figure 1.1: Snapshots of two light bulb based $N$-body simulations of galaxy encounters, one with the galaxies rotating clockwise (left panel), and one with the galaxies rotating counter-clockwise (right panel) [63].

center (e.g., see [35, 80, 102]). More details on $N$-body problems simulated using direct method can be found in [53, 4].

When simulating an $N$-body system using a direct method, the motions of particles are commonly approximated using a fourth order Hermite predictor-corrector scheme [85] with either an individual or shared time step scheme. A shared time step scheme imposes the same update interval on all particles while an individual time step scheme assigns an update interval that is specific for each particle. When a particle is selected to update its position and velocity in an individual time step simulation, the positions and velocities of other particles are extrapolated to that point in time. Then, the force exchanges are evaluated between the selected particle and the rest of the system, after which the positions, velocities and time step of the selected particles are updated. Future direct $N$-body codes may feature sixth or eighth order integration schemes which operate more efficiently and obtain comparable accuracy using fewer time steps [96].

Using a shared or individual time step scheme for $N$-body integration has disadvantages, especially when the code is run on a parallel or multi-core architecture. Shared time step schemes scale well on parallel architectures, but waste CPU time when tight few body interactions lower the time step length of the whole $N$-body system. Schemes using individual time steps are more calculation efficient, but scale poorly on parallel architectures due to the time dependencies of gravitational dynamics. The block time-step scheme [90] rounds the individual time step sizes down to a range of discrete values (e.g., powers of two), allowing the integration of multiple particles simultaneously while avoiding unnecessary force calculations. Block time step codes can therefore efficiently be used in parallel [50, 105].

The explicit calculation of forces between all particle pairs in direct-method simula-
tions results in a computational complexity that scales with the particle count squared per integration step ($O(N^2)$). In addition, the number of integration steps required for a full-length simulation also scales with $O(N)$ leading to a total computational complexity of $O(N^3)$. Due to this steep scaling, direct-method integration can only be applied nowadays to problems of up to a few million particles, whereas for example the Milky Way contains about 100 billion stars.

1.1.2 Alternative $N$-body integration methods

To reduce the computational complexity of $N$-body simulations, and enable the integration of larger $N$-body systems, several alternative techniques have been developed. One of the most widely used alternatives is the Barnes-Hut tree method [14], which orders particles into cells of a hierarchical three-dimensional tree. The force interactions over long distances are then calculated using averaged values for mass and position from tree cells instead of using individual particles. The force interactions over short distances are still calculated using individual particles. The accuracy of the tree integration method can be adjusted by modifying the opening angle ($\theta$). A larger opening angle results in fewer force evaluations per particle, and a lower accuracy overall. The application of the Barnes-Hut method results in a reduced computational complexity of $O(N \log N)$ per integration step. A graphical example of the computed interactions in a tree method force calculation can be found in Fig. 1.2.

Another method to speed up $N$-body integration is the particle-mesh method [92, 62, 91], where all particles are mapped to a mesh by adding weights to the closest mesh vertices. The method then calculates the gravitational potential by converting the mesh to the frequency domain using a Fast Fourier Transform, solving the Poisson equations and then converting it back to the spatial domain. The particle-mesh method is highly accurate in long-distance force calculations, scales linearly with the number of particles and with the number of mesh cells ($N_c$) as $O(N_c \log N_c)$. The method is less suitable for resolving force interactions over short distances, and is therefore frequently used in combination with an adaptive grid [20] or other integration methods (e.g., PPPM [58]).

One hybrid method that became popular recently in cosmological $N$-body simulations is the tree/particle-mesh (TreePM) method [133]. This method calculates the force interactions over short distances using the tree method and over long distances using the particle-mesh method. Simulations using the TreePM method are commonly performed to investigate the formation of galaxies in the universe (e.g., see [120, 100, 66]).

1.2 High-performance $N$-body simulations

The simulation of large $N$-body systems is computationally expensive, and even approximate integration methods often require more compute power than a single system can provide. We can speed up the $N$-body calculations by parallelizing the simulation or optimizing it for specialized hardware.
1.2 High-performance N-body simulations

Figure 1.2: 2D example of several force calculations using a tree code. The particle for which tree integration is performed is given by the blue dot in the bottom left corner. The tree cells that are used to integrate this particle are given by the colored boxes and the individual particles are indicated by red dots. The centers of mass of tree cells containing one or more particles are indicated by black dots. Particle-particle force interactions are shown by the red lines, particle-cell interactions are shown by the black lines.

1.2.1 Parallelization and speedup

A parallelized simulation uses multiple computing nodes by distributing the computationally intensive tasks such as force calculations among the processes. It therefore takes less time to complete than a single process simulation. However, some knowledge of parallel programming is required, which can be found in e.g., [32].

If both the single process and the parallel simulation are run using the same problem size, the obtained improvement in performance is referred to as the fixed-size speedup ($S_{\text{fixed-size}}$). The maximum obtainable fixed-size speedup is constrained by the ratio between the serial execution time $V$ and the parallel execution time $T$ of an otherwise identical N-body simulation [8]:

$$S_{\text{fixed-size}} \leq \frac{V}{T_s + \frac{T - T_p}{p}}.$$  \hspace{1cm} (1.2)

where $T_s$ is the time spent on work that has not been parallelized and $p$ the number of concurrent processes. In this equation, we find that the fixed-size speedup never exceeds the number of processes and will always be less than $V/T_s$.

If we take the communication overhead required for parallel processing into account
as suggested by [124], we are able to further constrain the maximum speedup to

\[ S_{\text{fixed-size}} \leq \frac{V}{T_s + \frac{T_\text{p}}{p} + LC_c}. \]  (1.3)

Here \( L \) is the average point-to-point network latency and \( C_c \) the number of blocking communications in the simulation. The number of blocking communications in \( N \)-body simulations scales with either \( \log p \) or \( p \), because the summation of forces requires communications over all processes. Such global exchanges require \( \log p \) blocking communications, or \( p \) communications if a ring topology is used. The overhead introduced by inter-process communication results in a decrease in speedup if \( p \) is sufficiently large.

The efficiency of a simulation across sites (\( E \)) is calculated by dividing \( V \) by the product of \( T \) and \( p \). It is therefore equal to \( S_{\text{fixed-size}} \) divided by \( p \):

\[ E = \frac{V}{Tp} = \frac{S_{\text{fixed-size}}}{p}. \]  (1.4)

A second limitation to the obtained fixed problem size speedup is the degree of parallelism of the application, which indicates the maximum number of concurrent operations that can be performed by the simulation. The maximum degree of parallelism for \( N \)-body simulations is typically very large, because a large number of independent force calculations are performed during each integration step.

Modern \( N \)-body simulations require an amount of wall-clock time that is comparable to historical runs but, thanks to the advances in compute technology, perform much more calculations during this period. As a result, the scaling we observe historically is one where the compute time of simulations was kept (roughly) constant, rather than the problem size. Gustafson observed this and proposed the concept of fixed-time scaling and fixed-time speedup [47]. Fixed-time (or scaled) speedup is measured by scaling up the problem size as more processes are added to the simulation, maintaining a constant execution time. We then measure the fixed-time speedup using

\[ S_{\text{fixed-time}} = \frac{T_{s1}}{T_{sp}}. \]  (1.5)

Here, \( T_{s1} \) is the hypothetical execution time of the scaled workload using one process and \( T_{sp} \) the execution time of the scaled workload on \( p \) processes. Gustafson found that scaled speedup has fewer constraints than fixed-size speedup, and that parallel programs are more efficient when the problem is properly scaled. An alternative method of scaling and measuring scaled speedup is the concept of memory-bound speedup [124]. Here, the problem is scaled such that the memory on all nodes is fully used. This metric is useful for integrators with low computational complexity (such as the TreePM method), as these are more likely to be limited in scaling by the memory size.
1.2 High-performance N-body simulations

1.2.2 Communication in parallel simulations

The performance of a parallel simulation depends on the properties of both the network used and the communication characteristics of the application.

The latency of a network is defined as the time required to send a signal from one endpoint to the other. The latency is heavily dependent on distance because a signal cannot travel faster than the speed of light. Reducing the latency is therefore difficult without modifying the topology of the network. The bandwidth of a network is defined as the number of bytes per second that can be transferred through the network. The bandwidth is limited for a given network, but can be increased for example by upgrading the hardware or adding more network cables. In addition to a low latency and a high bandwidth, a high quality of service and a minimal background load on the network are important to efficiently run a parallel simulation.

The performance of parallel simulations also depends on the communication characteristics of the simulation itself. The time spent on network latency is calculated by multiplying the number of blocking communication steps in the simulation with the network latency, while the time spent on transferring data is calculated by dividing the communication volume of each process by the available network bandwidth.

1.2.3 N-body simulations using special-purpose hardware

Executing an N-body simulation on special-purpose hardware can lead to large improvements in performance. While general-purpose CPUs have a broad instruction set applicable for a limited number of pipelines, specialized hardware components typically map a more narrow instruction set to a large number of pipelines. This allows a single PC with special-purpose hardware components to achieve a performance comparable to that of dozens of mundane PCs. Several types of special-purpose hardware are currently applied for N-body simulations.

The GRAvity PipE (GRAPE) has been specifically designed to perform fast force calculations, achieving a performance comparable to a hundred PCs at time of launch. The first version was presented in 1990 [123], and broke all performance records once in operation. The latest GRAPE card, the GRAPE-DR [87], was launched in 2009 and delivers 1 TFLOP/s per chip (a card contains up to 8 chips). For nearly two decades, the GRAPE has been the most powerful device for large-scale direct-method N-body simulations.

A notable alternative for the GRAPE emerged in 2007, when the release of the Compute Unified Device Architecture (CUDA) enabled programmers to efficiently use Graphics Processing Units (GPUs) for general-purpose computing. GPUs are designed to produce realistic 3D visualizations, and modern cards are optimized to process as many as 240 floating point operations in parallel [99]. The force calculations required for N-body integration map particularly well to this architecture as they can be performed independently within a given time step and require many repetitions of a limited number of floating point operations. Several groups obtained a simulation performance of up to 200 GFLOP/s per card [18, 49, 40]. Because GPUs offer performance comparable to
specialized hardware at a fraction of the cost, they are now used for a wide range of scientific applications.

The concept of using a high-performance instruction set with limited functionality is not exclusive to special-purpose hardware. Since the release of the 8087 co-processor for floating point operations in 1980 [101], CPU manufacturers have sought to improve the CPU performance for commonly applied computations by introducing specialized processing units. For example, modern commodity processors sold by AMD and Intel provide the SSE1-4 instruction sets to improve floating-point calculation performance. The performance of an application can be improved if it is tuned to use these instructions, but such enhancements are not straightforward to implement. Many compilers are capable of mapping applications to use special-purpose instructions to some extent, but the best performance is obtained by libraries directly written in x86 assembly language. The Phantom GRAPE is a library written in assembler to specifically speed up $N$-body integration [97], and provides a speedup of $\sim 8$ compared to regular compiler generated code.

1.3 Distributed infrastructures and computational grids

During the last two decades, the quality and performance of wide area networking has improved at a rate that exceeds the increase in computational power. This has made it increasingly convenient to create a distributed infrastructure by linking machines together over a wide area network (WAN). Distributed infrastructures are often heterogeneous and encompass multiple administrative domains. However, if proper measures are taken to combine the different sites, these infrastructures become
a suitable platform for parallelized simulations.

The most convenient way to combine computational sites into a unified infrastructure is by installing grid middleware such as the Globus Toolkit [31, 30]. Grid middleware adds an abstraction layer on top of the distributed infrastructure, and provides a uniform interface to the full system for all its users. This interface includes a single sign-on method for all sites using proxy certificates, which allows users and programs alike to freely switch between sites once they are granted access to the grid. In addition, grid middleware supports schedulers that distribute computational ‘jobs’ across the participating sites [5, 55] and mechanisms that allow for automated data replication [22].

The technical barriers for installing grid middleware on computers are trivial to bypass but the political limitations are more difficult to overcome. Especially the larger and more production-oriented supercomputer centers are reluctant to compromise on their scheduling, access and security policies in order to collaborate in a computational grid. Although grid projects consisting of supercomputers do exist (e.g., DEISA [25]), these infrastructures tend to have a more restrictive set of middleware with limited capabilities for resource reservations and no scheduling mechanism for runs across sites. In addition, such grids of supercomputers require dedicated WAN links, as the data volumes exchanged between these machines are generally too large to be efficiently handled by regular internet.

Computational grids are mainly used for high-throughput computing, a discipline where grid infrastructures are used to execute vast numbers of relatively small jobs. However, several projects have developed tools to enable the parallel execution of programs across multiple grid sites. These tools include MPI [118] implementations that specifically support grid security (such as MPICH-G2 [70] and MPIG [89]) or support execution across sites more generally (such as PACX-MPI [38] and OpenMPI [37]). Running a simulation across sites is a potentially attractive way to improve the performance of large-scale $N$-body simulations, and to enable the simulation of $N$-body problems of unprecedented scales.

1.4 Thesis overview

The $N$-body simulations performed up to now provided answers on many research questions, and a variety of integration techniques have been developed to efficiently simulate the dynamical evolution of stellar and cosmological systems. However, up to this point all production $N$-body simulations have been run on local computing sites only. Using multiple sites in parallel allows the simulation of larger $N$-body systems, and the continual increase in the number of computing sites will provide a structural boost in growth of $N$-body simulation performance. However, the computational capabilities provided by a planet that is already covered with computer hardware and networks are not trivially applied to solve large-scale $N$-body problems. Is it possible to map high performance $N$-body simulations to infrastructures that span up to several thousands of kilometers? If so, how can we efficiently use these widely distributed resources? The
primary aim of this thesis is to examine the feasibility of simulating \( N \)-body systems on wide area computational (grid) infrastructures. We examine a variety of \( N \)-body integrators used for increasingly large \( N \)-body systems. These integrators range from the high-accuracy direct summation methods used to model dense star clusters to the fast Tree/Particle Mesh methods used to reproduce structure formation in the universe.

1.4.1 Chapter 2

In Chapter 2 we present performance measurements of direct gravitational \( N \)-body simulations on the grid, with and without specialized (GRAPE-6) hardware. Our inter-continental virtual organization consists of three sites, one in Tokyo, one in Philadelphia and one in Amsterdam. In many cases, high performance simulations over the entire planet are dominated by network bandwidth rather than latency. With this global grid of GRAPEs our calculation time remains dominated by communication over the entire range of \( N \), which was limited due to the use of three sites. Increasing the number of particles will result in a more efficient execution. Based on these timings we construct and calibrate a model to predict the performance of our simulation on any grid infrastructure with or without GRAPE. We apply this model to predict the simulation performance on the Dutch DAS-3 wide area computer. Equipping the DAS-3 with GRAPE-6Af hardware would achieve break-even between calculation and communication at a few million particles.

1.4.2 Chapter 3

In Chapter 3 we describe the living application, a method to autonomously manage applications on the grid. During its execution on the grid, the living application chooses the resources to use in order to complete its tasks. These choices can be based on the internal state, or on autonomously acquired knowledge from external sensors. By giving limited user capabilities to a living application, the living application is able to port itself from one resource topology to another. The application performs these actions at run-time without depending on users or external workflow tools. We demonstrate this new concept in a special case of a living application: the living simulation. Today, many simulations require a wide range of numerical solvers and run most efficiently if the solvers are matched to specialized nodes. The idea of the living simulation is that it decides itself which grid machines to use based on the numerical solver currently in use. In this chapter we apply the living simulation to model the collision between two galaxies in a test setup with two specialized computers. This simulation switches at run-time between a GPU-enabled computer in the Netherlands and a GRAPE-enabled machine that resides in the United States. It uses a Barnes-Hut tree \( N \)-body code \[39\] when it runs on the GPU and a direct \( N \)-body code \[50\] when it runs on the GRAPE.

1.4.3 Chapter 4

Chapter 4 reports on CosmoGrid, a large-scale cosmological \( N \)-body simulation. Cosmological \( N \)-body simulations are an essential tool for evolving density perturbations in
1.4 Thesis overview

Figure 1.4: Overview of the infrastructures used in this thesis. The left panel shows the systems used in Chapters 2 and 3, the middle panels shows the systems used in Chapters 4 to 6 and the right panel shows the system used additionally in Chapter 5 and 6.

the nonlinear regime. Simulating the formation of large-scale structures in the universe, however, is still a challenge due to the enormous dynamic range in spatial and temporal coordinates, and the enormous computer resources required. The dynamic range is generally handled by the hybridization of numerical techniques. We deal with the computational requirements by connecting two supercomputers through an optical network and make them operate as a single machine. This is challenging, if only for the fact that the supercomputers of our choice are separated by half the planet, as one is located in Amsterdam and the other is in Tokyo. The co-scheduling of the two computers and the gridification of the code enables us to achieve a 90% efficiency for this distributed intercontinental supercomputer. We conclude that running cosmological N-body simulations on a limited number of (≤ 100) processors concurrently on more than 10 supercomputers would provide satisfactory performance and is politically favorable regarding the acquisition of the resources.

1.4.4 Chapter 5

In Chapter 5 we provide a detailed analysis of N-body integrators on supercomputer infrastructures. Here we present an extended version of the GreeM Tree/Particle Mesh integrator which runs on two or more supercomputers. Our code requires no underlying middleware, and can be tuned to efficiently use long distance optical networks. We have performed simulations using up to five sites, and found that the communication overhead of our code increases only marginally as we scale up in number of sites. We also present a time-complexity model for cosmological tree-particle mesh N-body simulations and compare the model predictions with our latest results. We conclude this chapter with a feasibility study of gridification of different N-body integrators. We examine the time complexity models of several integration methods (direct, tree and
TreePM) and apply the models to predict the performance of different $N$-body solvers on a globally distributed supercomputer.

1.4.5 Chapter 6

In Chapter 6 we present MPWide, a platform independent communication library for performing message passing between computers. Our library allows the coupling of several local MPI applications through a long distance network and is specifically optimized for such communications. The implementation is deliberately kept light-weight, platform independent and the library can be installed and used without administrative privileges. The only requirements are a C++ compiler and at least one open port to a wide area network on each site. In this chapter we present the library, describe the user interface, present performance tests and apply MPWide in a large scale cosmological $N$-body simulation on a network of two supercomputers, one in Amsterdam and the other one in Tokyo.
2 Distributed Direct \( N \)-body Simulations on a Global Grid of GRAPEs

Based on:
D.J. Groen, S.F. Portegies Zwart, S.L.W. McMillan and J. Makino,
Distributed N-body Simulation on the Grid Using Dedicated Hardware,

In this chapter we present our direct-method \( N \)-body simulations on a planet-wide network of GRAPEs connected by regular internet. The experiments were performed using grid middleware and MPI across sites.

2.1 Introduction

Star clusters are often simulated by means of direct-method \( N \)-body simulations [2]. The Newtonian gravitational force on individual stars in such simulations is calculated by aggregating the force contributions from all other particles in the system.

To enable faster execution of these simulations, specialized solutions such as GRAvity PipEs (GRAPEs) [36], Graphics Processing Units (GPUs) [108, 49, 18] and Field-Programmable Gate Arrays (FPGAs) [76] have been successfully developed and applied. These solutions are designed or tuned specifically for optimizing force calculations, and provide dramatic speedup. For example, the GRAPE-6Af features a dedicated hardware implementation that can calculate 42 force interactions simultaneously and with increased efficiency. As a result, the GRAPE is able to perform force calculations \( \sim 130 \) times faster than a single PC [86]. Recently, GPUs have shown gains in speed and flexibility, and they are now used for simulating self gravitating systems at speeds comparable to GRAPE [108, 49, 18].

Parallelization of GRAPEs appears to be an efficient way to reduce the wall-clock time for individual simulations [88, 46, 50]. The gravitational \( N \)-body problem has calculation time complexity \( O(N^2) \), whereas the communication scales only with
\(O(N)\). For sufficiently large \(N\), the force calculation time will therefore overtake the communication time. For a local cluster of GRAPEs with low-latency and high bandwidth network, break-even between calculation and communication is reached at \(N \sim 10^4\) [50].

Generally, GRAPE clusters are not cheap and few institutions can afford such dedicated hardware solutions. Still, more than 500 GRAPE modules, where one module is equivalent to one GRAPE-6Af, or 4 GRAPE-6 chips, are currently in use across 37 institutions in 12 countries world-wide. An alternative to purchasing a large GRAPE-6 or GPU cluster is provided by a computational grid. In a grid, several institutions assemble in a virtual organization, within which they share resources, and the costs for purchasing and maintaining these resources [31]. Grid middleware provides a secure wide area computing environment without requiring users to register for individual clusters. In addition, grid-enabled MPI implementations, such as MPICH-G2 [70] or OpenMPI [42], provide the ability to run MPI jobs across sites in the grid, using the existing MPI standards. Applying such grid technology to clusters of GPUs is an attractive option, because there are a large number of (frequently idle) GPUs in consumer machines. By connecting these consumer machines to the grid (as was done in a similar fashion with regular CPUs for the SETI@home project [9]) and using them for parallel \(N\)-body simulations, we can increase the computational power of the grid in a cheap and convenient manner.

Although there is a clear benefit of using grid technology in sharing financial burden, the real challenge is to develop new applications for astronomical problems that have yet to be solved. For example, the simulation of an entire galaxy, requires at least a few PFLOP/s of computational power and the development of a hybrid simulation environment [60]. Such an environment performs several astrophysical simulations on vastly different temporal and spatial scales. For example, a hybrid simulation environment could consist of a stellar evolution simulation to track how individual stars evolve over time, a smoothed particle hydrodynamics simulation [93] to simulate stellar collisions or close encounters, and a direct-method \(N\)-body calculation to simulate the remaining dynamics between stars.

To facilitate these tightly-coupled multi-physics simulations on the PFLOP/s scale, it will no longer be sufficient to do high-performance computing (HPC) on a local cluster, as we require an extensive grid infrastructure consisting of several of such clusters. Although grid technology has been largely applied to facilitate high-throughput computing [5], little research has been done on investigating how the grid can be efficiently applied to solve tightly-coupled HPC problems. By using grid technology for this specific set of problems, we can potentially fulfill the computational requirements for performing petascale multi-physics simulations.

Using a grid infrastructure for HPC has another drawback, as the communication between grid sites dramatically increases network overhead compared to a local cluster. For intercontinental communication, the network latency can become as large as 0.3s, which is especially impractical for applications, such as direct-method \(N\)-body codes, that require communication over all processes during every iteration. Still, even for such long communication paths there will be a problem size (\(N\)) for which wall-clock time is
Table 2.1: Specifications for the nodes in G3. The first column gives the name of the computer followed by its country of residence (NL for the Netherlands, JP for Japan and US for the United States). The subsequent columns give the type of processor in the node, the amount of RAM, followed by the operating system and the kernel version. The machines in Japan used Globus 3.2.1, the machines in the Netherlands used Globus 4.0.3 and the machine in the US used Globus 4.0.4. Each of the nodes is equipped with a 1 Gbit/s Ethernet card and GRAPE-6Af hardware. Local nodes are interconnected with Gigabit Ethernet.

<table>
<thead>
<tr>
<th>name</th>
<th>country</th>
<th>CPU type</th>
<th>RAM [MB]</th>
<th>OS</th>
<th>kernel version</th>
</tr>
</thead>
<tbody>
<tr>
<td>vader</td>
<td>NL</td>
<td>Intel P4 2.4GHz</td>
<td>1280</td>
<td>Ubuntu 5.10</td>
<td>2.6.5</td>
</tr>
<tr>
<td>palpatine</td>
<td>NL</td>
<td>Intel P4 2.67GHz</td>
<td>256</td>
<td>RHEL 3</td>
<td>2.4.21</td>
</tr>
<tr>
<td>yoda</td>
<td>JP</td>
<td>Athlon 64 3500+</td>
<td>1024</td>
<td>FC 2</td>
<td>2.6.10</td>
</tr>
<tr>
<td>skywalker</td>
<td>JP</td>
<td>Athlon 64 3500+</td>
<td>1024</td>
<td>FC 2</td>
<td>2.6.10</td>
</tr>
<tr>
<td>obi-wan</td>
<td>US</td>
<td>2x Xeon 3.6GHz</td>
<td>2048</td>
<td>Gentoo 06.1</td>
<td>2.6.13</td>
</tr>
</tbody>
</table>

dominated by the force calculation rather than by communication. Earlier experiments indicate that a grid of regular PCs across Europe improves overall performance for relatively small \( N \) \cite{46}. We address the question for which problem size a world-wide grid has practical usage, in particular if such a cluster is equipped with GPUs or GRAPEs.

2.2 Experiment setup

We have constructed a heterogeneous grid of GRAPEs, which we call the Global GRAPE Grid (or G3). The G3 consists of five nodes across three sites. Two nodes are located at Tokyo University (Tokyo, Japan), two are located at the University of Amsterdam (Amsterdam, the Netherlands) and one is at Drexel University (Philadelphia, United States). Each of the nodes is equipped with a GRAPE-6Af special purpose computer, which allows us to test several different resource topologies. Local nodes are connected by Gigabit Ethernet, whereas the different sites are connected with regular internet. In Table 2.1 we present the specifications of the G3. Each of the computers in the G3 is set up with Globus Toolkit middleware\footnote{http://www.globus.org} and MPICH-G2\footnote{http://www3.niu.edu/mpi/, in the future: http://dev.globus.org/wiki/MPICH-G2}.

In Table 2.2 we present the network characteristics, latency and bandwidth, of the connections within G3. We tested local area network (LAN) and wide area network (WAN) connections using the UNIX `ping` command to measure latency. We use `scp` for measuring the network bandwidth, transferring a 75 MB file, rather than referring to theoretical limits because the majority of bandwidth on non-dedicated WANs is used by external users. For our performance measurements, we used a standard implementation...
Table 2.2: Characteristics of local and wide network connections. Latency indicates the required time for sending 1 byte through the network connection. The bandwidth indicates the transfer capacity of the network connection. The bandwidth was measured with a 75MB `scp` file transfer.

<table>
<thead>
<tr>
<th>connection</th>
<th>latency [ms]</th>
<th>bandw. (theory) [MB/s]</th>
<th>bandw. (real) [MB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amsterdam LAN</td>
<td>0.17</td>
<td>125.0</td>
<td>11.0</td>
</tr>
<tr>
<td>Tokyo LAN</td>
<td>0.04</td>
<td>125.0</td>
<td>33.0</td>
</tr>
<tr>
<td>Amsterdam - Tokyo WAN</td>
<td>266.0</td>
<td>57.0</td>
<td>0.22</td>
</tr>
<tr>
<td>Amsterdam - Phil. WAN</td>
<td>104.0</td>
<td>312.5</td>
<td>0.56</td>
</tr>
<tr>
<td>Philadelphia - Tokyo WAN</td>
<td>188.0</td>
<td>57.0</td>
<td>0.32</td>
</tr>
</tbody>
</table>

of MPICH-G2 without specific optimizations for long-distance networking. As a result, the MPI communication makes use of only 40%-50% of the available bandwidth \(^3\). If we were to enhance MPICH-G2 with additional optimizations, or add support for grid security to already optimized MPI libraries, such as Makino’s tcplib\(^4\) or OpenMPI, our bandwidth use would be close to the bandwidth use of a regular file transfer.

The \(N\)-body integrator we have chosen for our experiments uses block time-steps \([90]\) with a 4th order Hermite integration scheme \([85]\). The time steps with which the particles are integrated are blocked in powers of two between a minimum of \(2^{-22}\) and a maximum of \(2^{-3}\). During each time step, the codes perform particle predictions, calculate forces between particles and correct particles on a block of active particles. Particle corrections include updates of positions and velocities, and computation of new block time steps of particles.

For our experiments we use three implementations of a parallel \(N\)-body integrator. One of these codes runs on a single PC with and without GRAPE. The two others are parallelized using MPI: one of these uses the copy algorithm \([83, 27]\) and the other uses the ring algorithm \([33, 10, 46]\). When using a copy algorithm, each process contains a copy of the full system, but integrates only a subset of the active particles. After integration, all active particles are exchanged between the processes to ensure that each copy of the \(N\)-body system is kept up-to-date. When using a ring algorithm, each process contains only a fraction of the full \(N\)-body system. Here, the active particles traverse across all processes in a ring topology and receive a partial update from each process. The copy algorithm has a smaller number of communication steps whereas the ring algorithm has lower memory usage on the nodes.

We initialize the simulations using Plummer \([103]\) spheres that were in virial equilibrium and performed our simulations using a softening parameter of \(2^{-8}\). Since our simulations are performed over one dynamical (\(N\)-body) time unit \([54]\), the realization

\(^3\) for more information we refer to a research report from INRIA: http://hal.inria.fr/inria-00149411/en/

\(^4\) see: http://grape.mtk.nao.ac.jp/~makino/softwares
of the \( N \)-body system is not critical to the timing results.

### 2.3 Results of grid calculations

We have performed a number of simulations on local machines and on the G3, which consists of simulations lasting one \( N \)-body time unit and shorter simulations lasting one integration time step. We measured the full wall-clock execution time for the longer simulations and we profiled the shorter simulations.

#### 2.3.1 Timing results of \( N \)-body calculations

We run the \( N \)-body codes, discussed in §2.2, on a single PC and across the network in parallel using \( N = 1024 \) to \( N = 65536 \) (a few additional calculations were performed with \( N > 65536 \)). The runs were performed with and without GRAPE. In Figs. 2.1 and 2.2 we present the results of the copy and ring algorithms. If a simulation is run multiple times with the same problem set, the execution time may be slightly different per run. This variation is relatively small, as the slowest of 4 repeated runs (using 32768 particles over two sites) was a factor 1.07 slower than the fastest run. The variation can be primarily attributed to fluctuations in the network bandwidth.

**Single PC**

The performance on a single PC (represented by the thick solid line with bullets in Fig. 2.1) is entirely dominated by force calculations, which scales as \( \mathcal{O}(N^2) \). As the number of steps per \( N \)-body time unit increases with \( N \), the execution time scales slightly worse than \( N^2 \).

**Grid of PCs**

The performance on the G3, without using GRAPE, is given by the thin dashed line with triangles. For \( N < 24576 \), the performance is dominated by network communication. Given that \( p \) indicates the number of processes, the network communication scales as \( \mathcal{O}(N \log p) \) (see [50]). For our grid-based simulation experiments without GRAPE, break-even between communications and force calculations is achieved around \( N \sim 3 \cdot 10^4 \) for the copy algorithm (Fig. 2.1), and at a somewhat higher value for the ring algorithm (Fig. 2.2). For larger \( N \), the execution time is dominated by force calculations, rather than network communication. For these high \( N \), the grid speedup \( \Gamma \) [61], which is the single-site execution time divided by the execution time over three grid sites, increases to 1.37 for the copy and 1.24 for the ring algorithm. As can be seen by comparing Figs. 2.1 and 2.2, the copy algorithm gives overall better performance than the ring algorithm. This can be explained by the smaller number of communication steps in the copy algorithm.

**Single PC with GRAPE**

The performance on a single PC with GRAPE is dominated by force calculations, although communication between host and GRAPE, and operations on the host machine have an impact on performance for \( N < 16384 \). In addition, the GRAPE performs less efficiently for low \( N \), because many blocks are too small to fill the GRAPE pipelines.
For larger $N$, force calculations become the performance bottleneck, and the scaling of the execution time becomes that of a single PC without GRAPE.

**Grid of PCs with GRAPE**

The performance on the G3 (with GRAPEs) using all three sites is given by the thin solid line with triangles. For all problem sizes $N$ we have measured, the grid speedup $\Gamma$ is less than 0.15, indicating that the performance is dominated by network communication. The network communication time scales better than the force calculation time, therefore, force calculation time will overtake the network communication time if $N$ is sufficiently large. However, this break-even point lies at much higher $N$ than for a Grid of PCs, because the use of GRAPE greatly decreases the time spent on force calculations.

For the copy algorithm (see Fig. 2.1), calculations between Tokyo and Philadelphia take less time than calculations between Amsterdam and Tokyo, due to a lower network latency (see Table 2.2). The calculations across three sites take more time than calculations across two sites. This is caused by the latency of all-to-all MPI communications in the copy algorithm, which scales with the number of processors.

According to our profiling measurements in Fig. 2.3, for $N < 12288$, a simulation on the G3 with GRAPEs using the ring algorithm spends most of its time in network latency. For larger $N$ more time is spent on using the network bandwidth. These results indicate that network bandwidth is the primary bottleneck for our simulations on the G3 using the ring algorithm. When we compare the results of the runs on the grid with GRAPEs with each other, we do not notice any systematic trend. The results confirm that the wall-clock time is dominated by using the network bandwidth, which is bottlenecked by the transpacific network line for all grid setups.

### 2.3.2 Profiling of the $N$-body simulations

We have chosen one parallel algorithm (ring) and one resource topology (3 nodes on 3 sites) to profile the simulation during one integration time step. The block size $n$ for every measurement was fixed using a formula for calculating average block size ($n = 0.20N^{0.81}$), which has been used for the same initial conditions in [108]. During execution, we measured the time spent on individual tasks, such as force calculations or communication latency between processes. We have profiled our simulations for $N = 1024$ up to $N = 196608$, using the timings measured on the process running in Tokyo. The results of these measurements are given in Fig. 2.3.

We find that for larger $N$, low bandwidth of our wide area network affects the outcome of the performance measurements, and that MPI calls are only able to use about a quarter of the available bandwidth for passing message content. For $N \gtrsim 5 \cdot 10^5$ we expect the force calculation to take more time than network latency. If we were to use the network bandwidth more efficiently for such a large number of particles, the execution time would be dominated by force calculations. The network bandwidth can be used much more efficiently, either by using a more efficient MPI implementation (e.g., one that supports communication over multiple TCP connections) or by using a dedicated network. Using our current networking and MPI implementation, we expect that for $N \gtrsim 2 \cdot 10^6$ particles the force calculation time overtakes the bandwidth time.
2.4 Modelling the performance of the grid

In order to further understand the results and to enable performance predictions for larger network setups, we decided to model the performance of the grid calculations. We model the performance of the simulation by adopting the parallel performance models described by [83] and [50] and combining it with the grid performance model described in [46]. Further extension and calibration of the model allows us to simulate the performance of our \(N\)-body simulations on a G3 or any other topology.
Distributed Direct N-body Simulations on a Global Grid of GRAPEs

2.4.1 Single PC

An N-body simulation over one N-body time unit [54] consists of the following steps:

1. Read the input snapshot and initialize the N-body system.
2. Compute the next system time $t$ and select the block of $n$ active particles in the system.
3. Predict the positions and velocities of all $N$ particles to time $t$.
4. Calculate the forces and their time derivatives between the $n$ active particles and all $N$ particles in the system.
5. Correct the positions, velocities and velocity derivatives of the $n$ active particles, and update their time steps.
6. Repeat from step 2 until $t$ has exceeded one N-body time unit.
7. Write the output of the simulation and terminate it.

As relatively little time is spent on program initialization and finalization, we focus on the time to integrate the system ($T_{\text{integrate}}$), which consists of the tasks performed in steps 2 to 5. Throughout this paper we use uppercase $T$ to refer to the time spent...
2.4 Modelling the performance of the grid

Figure 2.3: Share of wall-clock time spent on individual tasks during a single time-step. Solid lines indicate tasks performed on the local machine. The thick solid line with filled circles represents time spent on force calculations, and the thin solid lines give the result for time spent on communication between PC and GRAPE (open triangles), particle corrections (open circles) and particle predictions (open squares) respectively. Dotted lines indicate time spent on communication between nodes. The thin dotted line with asterisks indicates time spent on communication latency between nodes and the thick dotted line with solid squares indicates time spent on using the network bandwidth.

in \( n_{\text{steps}} \) integration steps, and the lowercase \( t \) for the time spent in a single step. The total execution time \( T_{\text{integrate}} \) is

\[
T_{\text{integrate}} = \sum_{i=1}^{n_{\text{steps}}} (t_{\text{pred}} + t_{\text{force}} + t_{\text{corr}}),
\]

(2.1)

with the time spent on predicting particles

\[
t_{\text{pred}} = \tau_{\text{pred}} N,
\]

(2.2)

the time spent on calculating forces

\[
t_{\text{force}} = \tau_{\text{force}} nN,
\]

(2.3)

and the time spent on correcting the active particles

\[
t_{\text{corr}} = \tau_{\text{corr}} n.
\]

(2.4)
Table 2.3: Machine performance specification and machine-specific constants. The first two columns show the name of the machine, followed by the country of residence. The third column indicates machine speed in MFlop/s, using the Whetstone benchmark. The last three columns give the time required for the CPU to perform one particle prediction ($\tau_{\text{pred}}$), the time required for one force calculation between two particles ($\tau_{\text{force}}$) and the time required for correcting one particle ($\tau_{\text{corr}}$) respectively, all in microseconds.

<table>
<thead>
<tr>
<th>name</th>
<th>location</th>
<th>speed</th>
<th>$\tau_{\text{pred}}$</th>
<th>$\tau_{\text{force}}$</th>
<th>$\tau_{\text{corr}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>vader</td>
<td>NL</td>
<td>377</td>
<td>0.247</td>
<td>0.216</td>
<td>4.81</td>
</tr>
<tr>
<td>palpatine</td>
<td>NL</td>
<td>422</td>
<td>0.273</td>
<td>0.193</td>
<td>2.39</td>
</tr>
<tr>
<td>yoda</td>
<td>JP</td>
<td>436</td>
<td>0.131</td>
<td>0.110</td>
<td>1.29</td>
</tr>
<tr>
<td>skywalker</td>
<td>JP</td>
<td>436</td>
<td>0.131</td>
<td>0.110</td>
<td>1.29</td>
</tr>
<tr>
<td>obi-wan</td>
<td>US</td>
<td>1191</td>
<td>0.098</td>
<td>0.148</td>
<td>1.14</td>
</tr>
</tbody>
</table>

Here $\tau_{\text{pred}}$ is the time to predict a single particle, $\tau_{\text{force}}$ is the time to calculate the forces between two particles, and $\tau_{\text{corr}}$ is the time spent to correct a single particle. The values for $\tau_{\text{pred}}, \tau_{\text{force}}$ and $\tau_{\text{corr}}$ have been measured by using a sample $N$-body simulation with 32768 particles, and are given in table 2.3 for the various nodes in the G3. For a more practical comparison, we also measured the compute speed (in floating point operations per second) for each of the nodes. These measurements were carried out using the Whetstone benchmark [23].

2.4.2 Grid of PCs with copy algorithm

The performance model for a single PC can be extended to include the parallel operation in the copy algorithm. In the copy algorithm, each process has a full copy of the system of $N$ particles, but only computes the active particles in a specific subset of $N/p$ particles. The result of this computation is sent to all other processes. We assume that all $p$ processes have comparable speed, and every process has an equally sized subset of $n/p$ active particles. For the copy algorithm, the host computation time ($T_{\text{integrate}}$) also consists of the time spent to communicate between processes ($T_{\text{MPI}}$). Therefore,

$$T_{\text{integrate}} = \sum_{i=1}^{n_{\text{steps}}} (t_{\text{pred}} + t_{\text{force}} + t_{\text{corr}}) + T_{\text{MPI}}, \quad (2.5)$$

A process computes forces for its subset of $n/p$ active particles, and corrects only these particles. Therefore, a process requires at most $Nn/p$ force calculations per time step

$$t_{\text{force}} = \tau_{\text{force}} \frac{N}{p}n, \quad (2.6)$$

and a process corrects at most $n/p$ particles, of which the time spent is given by

$$t_{\text{corr}} = \tau_{\text{corr}} \frac{n}{p}. \quad (2.7)$$
2.4 Modelling the performance of the grid

In a parallel system, time is spent not only on integrating the system ($T_{\text{integrate}}$), but also on exchanging messages between processes ($T_{\text{MPI}}$). This time is obtained by adding the time spent on overcoming network latency ($t_{\text{latency}}$) and the time spent transferring particles ($t_{\text{bandwidth}}$)

$$T_{\text{MPI}} = \sum_{i=1}^{n_{\text{steps}}} (t_{\text{latency}} + t_{\text{bandwidth}}).$$

(2.8)

In our implementation $t_{\text{latency}}$ is given by the sum of the latencies of each MPI call in the code. The copy algorithm uses 1 MPI\_Allgather and 1 MPI\_Allgatherv command (of which the latencies both scale with $\log_2 p$ [50]) per block time-step, resulting in a total time spent on latency of

$$t_{\text{latency}} = (l_{\text{MPI\_Allgather}} + l_{\text{MPI\_Allgatherv}}) \log_2 p.$$  

(2.9)

The time used for transferring particle data is given by $t_{\text{bandwidth}}$, which is obtained by taking the total size of the data that has to be communicated (which is assumed to scale with $2(p-1)$ for all-to-all communications), and dividing it by the network bandwidth ($\tau_{\text{bw}}$). Particles are typically stored in a 124 byte data structure, which contains 4 arrays of 3 doubles each to store position, velocity, acceleration and derivative of the acceleration, 1 double each for mass, time step and last time step, and 1 integer for the identification number. This results in

$$t_{\text{bandwidth}} = \frac{n248(p-1)}{p\tau_{\text{bw}}}.$$  

(2.10)

For a wide area computer $t_{\text{latency}}$ and $t_{\text{bandwidth}}$ may be quite substantial, but the separation in parts, as given here, enables us to optimize our network computer with respect to the communication characteristics.

2.4.3 Grid of PCs with ring algorithm

Unlike the copy algorithm, the ring algorithm (discussed in detail in [33, 10]) does not use a single all-to-all communication operation, and only requires the processes to have a partial copy of size $N/p$ of the system. Communication occurs in a total of $p$ steps (or shifts). During every shift, each process performs a partial force integration by calculating the forces between their local subset of $n/p$ active particles and the $N/p$ particles stored in local memory. Then, each process sends its updated particles to their neighbor.

To model the performance for the ring algorithm, we use the model for the copy algorithm and redefine the time spent on force calculations ($T_{\text{force}}$) and the time spent to communicate between processes ($t_{\text{latency}}$ and $t_{\text{bandwidth}}$), as the force calculation and the MPI communications occur in multiple shifts. The time spent on a partial force calculation is given by

$$t_{\text{force,1shift}} = \tau_{\text{force}} \frac{nN}{p^2}.$$  

(2.11)
The total time spent on the force calculation is given by the time for a partial force calculation \( (t_{\text{force,1shift}}) \) multiplied by the number of shifts \((p)\),

\[
t_{\text{force}} = \tau_{\text{force}} \frac{N}{p}. \tag{2.12}
\]

For every time step, our implementation of the ring algorithm uses \( 2 \text{ MPI\_Allreduce} \) communication commands for initialization, and \( 1 \text{ MPI\_Sendrecv} \) operation for each shift. The time spent overcoming network latency is then

\[
t_{\text{latency}} = 2 \log_2 p \text{MPI\_Allreduce} + pl\text{MPI\_SendRecv}. \tag{2.13}
\]

The ring algorithm is more bandwidth intensive than the copy algorithm, as all block subsets are sent and received during a ring shift, multiplying the time spent on transferring particles by \( 2p \). Per particle, 124 bytes have to be transferred, therefore the time spent on transferring particles is given by

\[
t_{\text{bandwidth}} = 248 n / \tau_{\text{bw}}. \tag{2.14}
\]

### 2.4.4 Single PC with GRAPE

The GRAPE-6Af is a dedicated hardware component developed by a group of researchers led by Junichiro Makino at the University of Tokyo [36]. The GRAPE-6Af is the smallest commercially available GRAPE configuration, consisting of a single GRAPE module with a peak speed of about 123 Gflop/s. It calculates the forces between particles, which is the bottleneck in the calculation, whereas the particle predictions and corrections are still mostly done on the host PC.

When a GRAPE is used, an \( N \)-body simulation over one \( N \)-body time unit consists of the following steps:

1. Read the input snapshot and initialize the \( N \)-body system.

2. Compute the next system time \( t \) and select the block of active particles \( n \) in the system.

3. Predict the positions and velocities of the \( n \) active particles on the PC, and send the predicted values and the next system time to the GRAPE.

4. Predict the other particles in the system on the GRAPE.

5. Calculate the forces and their time derivatives, using the GRAPE, between the \( n \) active particles and all \( N \) particles in the system.

6. Retrieve the forces and their time derivatives from the GRAPE.

7. Correct the positions, velocities and velocity derivatives of the \( n \) active particles, and update their time steps.

8. Repeat from step 2 until \( t \) has exceeded one \( N \)-body time unit.
2.4 Modelling the performance of the grid

9. Write the output of the simulation and terminate it.

When using GRAPE

\[ t_{pc} = t_{\text{pred}} + t_{\text{corr}}, \]

(2.15)

where \( t_{\text{pred}} = n \tau_{\text{pred}} \). The time spent to integrate particles for one \( N \)-body time unit is given by

\[ T_{\text{integrate}} = \sum_{i=1}^{n_{\text{steps}}} (t_{pc} + t_{\text{grape}} + t_{\text{comm}}). \]

(2.16)

Here

\[ t_{\text{grape}} = \tau_{\text{pipe}} n N, \]

(2.17)

is the time for calculating the forces on the GRAPE. The time needed by the GRAPE to calculate the force between two particles is given by \( \tau_{\text{pipe}} \). The communication between host and GRAPE is given by \cite{36}

\[ t_{\text{comm}} = 60 t_i n + 56 t_f n + 72 t_j n. \]

(2.18)

Here the time to transfeer 1 byte of data to the GRAPE are given by \( t_i \) for the prediction step, \( t_f \) for the force calculation step, and \( t_j \) for the correction step. During these steps respectively 60, 56 and 72 bytes per particle in the block are transferred. We assume that \( t_i = t_f = t_j \). We derive \( t_j \) by measuring \( \tau_{\text{Gsend}} \), which is the time to send one 72-byte data unit to the GRAPE. Therefore, \( t_j = \tau_{\text{Gsend}} / 72 \). By rewriting \( t_i \), \( t_f \) and \( t_j \) as factors of \( \tau_{\text{Gsend}} \), we can simplify the equation for \( t_{\text{comm}} \) to

\[ t_{\text{comm}} = (60 + 56 + 72) \left( \frac{\tau_{\text{Gsend}}}{72} \right) n. \]

(2.19)

Time spent on calculating forces on the GRAPE \( t_{\text{grape}} \) cannot be directly measured by timing parts of the code, because the GRAPE force calculation includes some communication between host and GRAPE as well. However, we can derive \( \tau_{\text{pipe}} \) from the total time of the force calculation as was done in \cite{50}. Therefore, we can rewrite \( \tau_{\text{pipe}} \) as

\[ \tau_{\text{pipe}} = \frac{1}{N} [t_{\text{force}} - (116 \tau_{\text{Gsend}} / 72)]. \]

(2.20)

The time spent on performing \( N \) force calculations \( t_{\text{force}} \) is given by

\[ t_{\text{force}} = \tau_{\text{Gforce}} N, \]

(2.21)

where \( \tau_{\text{Gforce}} \) is the time spent to calculate forces between two particles. We then introduce the time constant \( \tau_{\text{Gforce}} \) in the function for \( \tau_{\text{pipe}} \),

\[ \tau_{\text{pipe}} = \tau_{\text{Gforce}} - \left( \frac{116 \tau_{\text{Gsend}}}{72} \right) \frac{1}{N}. \]

(2.22)

Using our derived functions for \( t_{\text{grape}} \) and \( t_{\text{comm}} \), we are now able to model the performance of the GRAPE. As mentioned in \cite{36}, \( \tau_{\text{Gforce}} \approx 4.3 \cdot 10^{-10} \) s.
2.4.5 Grid of PCs with GRAPE and copy algorithm

When using GRAPE and a parallel algorithm, the time spent to integrate particles for one N-body time unit is given by

\[ T_{\text{integrate}} = \sum_{i=1}^{n_{\text{steps}}} (t_{\text{pc}} + t_{\text{grape}} + t_{\text{comm}}) + T_{\text{MPI}}. \] (2.23)

We determine time spent communicating between hosts \((T_{\text{MPI}})\) using Eq. 2.8. To determine the time spent on the host \((t_{\text{pc}})\) we use the equation for the single process with GRAPE (see Eq. 2.15). However, as we correct only \(n/p\) particles in parallel algorithms, we apply Eq. 2.7 to determine the time spent by the process on correcting particles.

In a parallel setup of GRAPEs, each process needs to communicate and calculate forces for a subset of \(n/p\) particle in every block. We replace \(n\) by \(n/p\) in our equations for \(t_{\text{grape}}\) as well as \(t_{\text{comm}}\). Therefore, the time spent on calculating forces is given by

\[ t_{\text{grape}} = \left[ N \tau_{\text{Gforce}} - (116 \tau_{\text{Gsend}}/72) \right] \frac{n}{p}, \] (2.24)

and communication between the hosts and the GRAPEs becomes,

\[ t_{\text{comm}} = (60 + 56 + 72) \left( \tau_{\text{Gsend}}/72 \right) \frac{n}{p}. \] (2.25)

2.4.6 Grid of PCs with GRAPE and ring algorithm

In a ring algorithm, each process computes the forces between a local set of \(n/p\) active particles and the local system of \(N/p\) particles during a shift. Then, it sends the results to its next neighbor and receives another \(n/p\) particles from its other neighbor. Each of the nodes spends \(t_{\text{grape},1\text{shift}}\) calculating the forces for \(n/p\) particles during one shift. Before the node has integrated the force on all \(n\) particles, a total of \(p\) shifts have passed, resulting in a total compute time for this node of

\[ t_{\text{grape}} = pt_{\text{grape},1\text{shift}}, \] (2.26)

where the time to calculate forces for a single shift \((t_{\text{grape},1\text{shift}})\) is

\[ t_{\text{grape},1\text{shift}} = \left[ (N \tau_{\text{Gforce}}/p) - (116 \tau_{\text{Gsend}}/72) \right] \frac{n}{p}. \] (2.27)

When GRAPE is used, the exchanged particles are 172 bytes each because they contain two additional arrays for storing the old acceleration and the old derivative of the acceleration. Due to this increased particle size,

\[ t_{\text{bandwidth}} = 172n/\tau_{\text{bw}}, \] (2.28)

whereas \(t_{\text{latency}}\) remains unchanged. The time spent communicating between hosts \((T_{\text{MPI}})\) is calculated as for the ring algorithm without GRAPE.
2.5 Results of the performance model

We have applied the performance model from the previous section to the results presented in §2.3. In Fig. 2.1 we compare the measured wall-clock time \( T_{app} \) for the copy algorithm on the grid with the performance model, Fig. 2.2 shows a similar comparison for the ring algorithm. To guide the eye, the results for a single GRAPE are also presented in both figures. The performance model tracks the real measurements quite satisfactorily, giving a slightly lower computation time for a single GRAPE while giving a slightly higher computation time for a simulation across grid sites.

The communication overhead of a distributed computer often renders high performance computing on a grid inefficient. However, in the \( N \)-body problem the compute time scales with \( N^2 \) whereas the communication scales linearly with \( N \). For sufficiently large \( N \), there will eventually be a point where relatively little time is lost communicating, and the compute resources are efficiently used.

In figures 2.1 and 2.2 we can see that, for GRAPE-enabled simulations, break-even between calculation and communication is reached around \( N \approx 10^6 \). For large \( N \), a grid of two GRAPEs will outperform a single GRAPE. Our grid setup included three GRAPE-enabled sites. The location of these sites (Asia, Europe and America) were as widely distributed as physically possible. A more modest grid across a single continent, will perform considerably better than a global grid. With the performance model that we constructed in §2.4, we can now study various grid topologies without the need to physically build the environment and create a virtual organization.

2.5.1 Future prospects

We applied the performance model to three hypothetical grids of GRAPE nodes. These three grids are: 1) a grid of all the available GRAPEs on the planet, 2) a grid of sites with more than 1 Tflop/s in GRAPE speed, and 3) a recently established Dutch grid (Dutch ASCII Computer, DAS-3\(^5\)) equipped with GRAPEs.

Since the launch of GRAPE-6, a total of 1115 GRAPE-6 modules have been deployed worldwide. Japan leads the GRAPE-yard with more than 800 modules, followed by the US (119) and Germany (62). At the moment there are 876 GRAPE-6 modules in Asia, 132 in North America and 107 in Europe. In Table 2.4 we list the sites with more than 1 Tflop/s peak-performance in GRAPE hardware and their network characteristics. The network round-trip with the longest latency is a round-trip between Japan and Ukraine, whereas the network link with the longest latency (not given in Table 2.4) is the transpacific line between Japan and the US.

Organizing all the GRAPEs on the planet would be a challenging political problem. Organizing only the 13 largest sites would be somewhat easier, therefore we included a performance prediction of such an infrastructure as well. Constructing a virtual organization within the 4 universities (University of Amsterdam, Free University of Amsterdam, Leiden University and Delft Technical University) that participate in the DAS-3 project (and equipping the 270 available DAS-3 nodes with specialized hardware)

\(^5\)http://www.starplane.org/das3/
Table 2.4: Overview of the major GRAPE clusters (> 1Tflop/s) on planet Earth, including their relative network latency characteristics. The first column identifies the site, followed by the name of the institute, the country and the number of GRAPE modules (8 modules provide ~ 1 Tflop/s peak performance.). The fifth column identifies the site with which the latency, given in column #6, is shortest. The one but last column (column #7) identifies the site with which the latency, given in column #8, is longest. The total number of GRAPE modules is 996.

<table>
<thead>
<tr>
<th>ID</th>
<th>institution</th>
<th>country</th>
<th>GRAPE count</th>
<th>nearest site</th>
<th>farthest site</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Nat. Astr. Obs.</td>
<td>Japan</td>
<td>576</td>
<td>B</td>
<td>M</td>
</tr>
<tr>
<td>B</td>
<td>Tsukuba Univ.</td>
<td>Japan</td>
<td>240</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>C</td>
<td>AMNH</td>
<td>US</td>
<td>40</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>Astr. Rechen Inst.</td>
<td>Germany</td>
<td>30</td>
<td>K</td>
<td>B</td>
</tr>
<tr>
<td>E</td>
<td>Rochester Inst.</td>
<td>US</td>
<td>26</td>
<td>F</td>
<td>B</td>
</tr>
<tr>
<td>F</td>
<td>McMaster Univ.</td>
<td>Canada</td>
<td>13</td>
<td>E</td>
<td>B</td>
</tr>
<tr>
<td>G</td>
<td>Drexel Univ.</td>
<td>US</td>
<td>12</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>H</td>
<td>Max Planck Inst.</td>
<td>Germany</td>
<td>12</td>
<td>D</td>
<td>B</td>
</tr>
<tr>
<td>I</td>
<td>Univ. of A’dam</td>
<td>Netherlands</td>
<td>10</td>
<td>K</td>
<td>B</td>
</tr>
<tr>
<td>J</td>
<td>Wien University</td>
<td>Austria</td>
<td>10</td>
<td>H</td>
<td>B</td>
</tr>
<tr>
<td>K</td>
<td>Bonn University</td>
<td>Germany</td>
<td>9</td>
<td>D</td>
<td>B</td>
</tr>
<tr>
<td>L</td>
<td>Cambridge Univ.</td>
<td>England</td>
<td>9</td>
<td>I</td>
<td>B</td>
</tr>
<tr>
<td>M</td>
<td>Main Astr. Obs.</td>
<td>Ukraine</td>
<td>9</td>
<td>J</td>
<td>B</td>
</tr>
</tbody>
</table>
would be much easier than doing this across various countries. The Dutch DAS-3 Grid is equipped with a fast Myrinet-10G internet and distributed across the Netherlands, connected by 10Gb light paths between clusters. The latency for the longest path is estimated to be 3 ms, and we estimate the bandwidth of the connections to be 0.5 GB/s.

In the sequential case we do not take memory limitations into account. This assumption is unrealistic for predicting the performance of a GRAPE, since $N < 2^{62144}$, but GPUs have a similar performance to GRAPE, and are able to store up to 13 million particles [108]. In late 2007, the launch of a double-precision GPU is expected, making GPUs usable for production-type direct-method $N$-body simulations. Additionally, in recent years the amount of memory on GPUs has been steadily increasing, and we expect this trend to persist in the near future.

We model the ring algorithm on both the G3 and on the 13 largest sites (see Table 2.4). We adopted palpatine (see Table 2.1) as the workhorse host for the GRAPEs and adopted the network characteristics as listed in Table 2.4. The ring algorithm for our hypothetical grid experiment was assumed to be optimized for the use of distributed clusters of GRAPEs. The algorithm avoids latency intensive network links by combining communication of local clusters from multiple shifts. Thus, communication for the local (across node) network is separated from the global (across sites) communication. Finally, we assume that all networks between the sites have reasonable support for MPI multicast and gather operations, and that the latency of these operations scales with $\log_2 p$.

The results of the hypothetical global GRAPE grid are presented in Fig. 2.4. Here we see that a global grid in which all GRAPEs participate outperforms a single GRAPE by about two orders of magnitude for $N \gtrsim 10^8$ particles. For a sufficiently large number of particles ($N \gtrsim 10^9$) the total peak performance of the global GRAPE grid approaches about 75 Tflop/s. Eventually, the grid with all the GRAPEs would outperform the grid with only the largest machines by about 25%, proportional to the number of GRAPEs in the two setups. When running simulations of $N \sim 10^8$ it is faster to run on three large GRAPE sites, than to use all the GRAPEs on the planet in parallel.

The dashed curve in Fig. 2.4 shows the performance of the model assuming that all the 270 nodes of the DAS-3 were equipped with GRAPE-6Af hardware. With such a setup, the maximum performance of about 35 Tflop/s is achieved for $N \sim 10^7$ particles. This is an interesting number for production simulations for astronomical research.

In Fig. 2.5 we present the wall-clock time for each of the different ingredients of a grid calculation with GRAPEs on the DAS-3, using the performance model. Break-even between calculation (straight solid curve) and communication (thick dotted curve) is achieved around $N \sim 3 \cdot 10^6$. For this large number of particles the communication between GRAPE and host, the predictor and the corrector steps require little CPU time compared to the force calculation on the GRAPE. For $N \gtrsim 6 \cdot 10^6$ this setup would give an efficient use of the special processors, and high performance calculations on the grid would then be quite efficient.
Distributed Direct $N$-body Simulations on a Global Grid of GRAPEs

Figure 2.4: Speedup prediction of possible GRAPE grid setups compared to a single CPU. The solid lines indicate execution time using 1 CPU (horizontal reference line) or 1 GRAPE with infinite memory (curved line). The double-dotted line indicates the predicted speedup if all 1115 GRAPEs are linked together to perform one simulation using an optimized ring algorithm. The dashed line indicates the predicted speedup if all GRAPE sites with more than 1 Tflop/s are linked together to perform one simulation using an optimized ring algorithm. The dotted line indicates the speedup if all 270 nodes in the Dutch DAS-3 grid would be equipped with GRAPEs.

2.6 Discussion and conclusions

We studied the potential use of a virtual organization in which GRAPEs are used in a wide area grid. For this purpose, we developed a performance model to simulate the behavior of a grid in which each of the nodes is equipped with special purpose GRAPE hardware. We tested the performance model with an actual grid across three sites, each of which is located on a different continent. We used GRAPE hardware in Japan, the Netherlands and the USA simultaneously for calculations of 1024 up to 196608 particles.

With these particle numbers we were able to have a better performance than a single computer without GRAPE. We measured a grid speedup of $\Gamma \sim 1.37$ for a grid of PCs, and a grid of GRAPEs performs another $\sim 4$ times faster. On the entire range of $N$ we were unable to reach superior speed compared to a single GRAPE. However, we estimate that a small intercontinental grid of GRAPEs will reach superior performance for $N \gtrsim 3 \cdot 10^6$ particles.
2.6 Discussion and conclusions

Figure 2.5: Predicted decomposition of performance of a DAS-3 GRAPE grid. The thick solid line indicates total execution time and the flat thick dashed line indicates time spent due to network latency. The thin dashed line indicates time spent on using the network bandwidth and the steep thick dotted line indicates time spent on calculating forces. The three bottom lines indicate time spent on communication between hosts and GRAPEs (upper thick dash-dotted line), correcting particles (middle dash-dotted line), and predicting particles (lower dotted line)
We used our grid calculations with GRAPE to construct and calibrate a performance model, with which we studied the performance of a world-wide grid of GRAPEs. When all the GRAPEs on the planet would participate in a virtual organization it is possible to utilize the total machine’s performance, but only for really large systems of \( N \approx 10^9 \). Though the total performance for such a setup would be about 75 Tflop/s, such large \( N \) would still be impractical to run for production astronomical simulations.

We conclude that organizing all the major GRAPEs on the planet in a virtual organization is probably not worth the effort. Organizing a few of the largest sites with GRAPEs within one continent, however, appears politically doable and computationally favorable. For the DAS-3, for example, the GRAPEs would be used at maximum performance for a feasible number of stars. Modern simulations of up to about a million stars have been done before using GRAPE [107], but these calculations were performed on a single cluster, rather than on a grid. A grid setup as proposed here would allow the simulation of a few million stars within a reasonable time span.

If we were to equip the full DAS-3 wide area computer in the Netherlands with GRAPEs, maximum performance would already be achieved for \( N \approx 6 \cdot 10^6 \) particles. Though still large, such simulations would be very doable and have practical applications. We estimate that running a system of \( N = 10^6 \) stars with a Salpeter mass spectrum [115] over a wide range of stellar masses to the moment of core collapse would take about 4 months. The simulation would still be mostly dominated by network latency, but the high-throughput networking in the DAS-3 completely removes the bandwidth bottleneck.

We have mainly discussed the use of GRAPEs in a virtual organization, but new developments in using graphical processing units appear to achieve similar speeds as GRAPEs [108, 49, 18]. In addition, GPUs are equipped with a larger amount of memory, which allows us to exploit more memory-intensive, but also faster, parallel algorithms. Future grids are likely to be equipped with GPUs, as the GPU will become part of the standard equipment for every PC.

Although our proof-of-concept infrastructure was of limited size, we have shown that it is possible to use dedicated hardware components located across clusters for high-performance computing. Though the current performance over globally connected grids leaves a lot to be desired and much optimization remains to be done, the concept of using dedicated hardware components worldwide in parallel has been shown to work. It can therefore be applied to solving individual tightly-coupled scientific problems or as ingredient of a complex multi-physics simulation, such as simulating a full galaxy, given that the problem size is sufficiently large to overcome the networking limitations.
In this chapter we present our hybrid N-body simulations on a planet-wide network of GRAPEs and GPUs connected by regular internet. The experiments were performed using grid middleware and self-developed living application software.

3.1 Introduction

A grid application consists of a range of tasks, each of which may run most efficiently using a different set of resources. Most of these applications, however, use a fixed resource topology even though certain tasks could benefit from using different resources. This can be due to the computational demands of these tasks or due to a change in resource availability over time. A wide range of work has been done on developing external management systems that allow applications to change grid resources during execution. This includes workflow systems [56, 78, 136], check pointers [67] or grid schedulers with migration capabilities [34, 6] that support resource switches that are either part of a predefined workflow or requested by the user.

An application management system that autonomously switches at run-time has been proposed by [95], where a hierarchically distributed application management system dynamically schedules and migrates a bag-of-tasks style MPI application, using a static hierarchy of schedulers to accomplish this.

A self-adaptive grid application that does not require external managers has been presented in [132]. Although this application does not use grid scheduling, it is able to
autonomously migrate to different locations and change its number of processes. This has been accomplished by allowing all processes to share knowledge and cooperate in managing the application’s topology.

In this work, we propose the living grid application, in which the application also decides where to run, and which is also able to migrate itself at run-time to another computer when needed. The intelligent migration from one computer to another can be realized over a long baseline, but does not need to be designed this way (see Sec. 3.2). We then apply this method to a multi-scale simulation and demonstrate its working on an intercontinental grid of semi-dedicated computers by simulating the merging between two galaxies, which provides a typical example for a multi-scale simulation (Sec. 3.3). In this simulation, we used a straightforward and autonomous resource selection scheme, where the optimal site is chosen from a predefined list of available resources. The simulation does not contain specific mechanisms to ensure fault tolerance or fault recovery.

3.2 Living application

3.2.1 Rationale

A flexible approach is needed to execute a complex grid application with multiple tasks and a diverse palette of resource requirements. The application should then be able to switch between tasks at run-time and between the resources required for each of these tasks, while maintaining the integrity of its data during these switches.

A switch requires the application to terminate its current execution, output its current state, and from that reinitialize the application using a new resource topology suited for the task at hand. Previously this has been done on a grid only in orchestration with a workflow manager. A job submitted by a workflow manager lacks the ability to change its resource topology during execution, as it does not have the privileges to make use of grid schedulers. When running an application with multiple tasks, this results in a ‘bouncing’ pattern where the manager submits jobs which return once a switch is required, only to be instantly submitted again to handle a different task. In the most favorable case, the performance loss introduced by bouncing and managerial overhead can be limited, but even then the successful completion of the simulation depends on the availability of an external manager, which is a potential single point of failure.

3.2.2 How the living application works

The living application switches between sites and tasks dynamically and without external dependencies. It is based on four principles:

1. It makes decisions on which tasks to do and which resources to use.
2. It makes these decisions based on knowledge it has acquired at run-time.
3. It changes resources and switches between tasks.

4. It operates autonomously.

As a living application operates autonomously on the grid, it obtains its privileges on its own without interacting with an external workflow manager or user.

Upon initialization, the application is locally equipped with the tools and data to perform the required tasks and the criteria for switching between tasks or resource topologies. It is then submitted as a job to the grid with the initial resource requirements defined by the launcher. The living application begins execution on the grid and continues to do so until either a switch or a termination is required.

The conditions for switching or termination are determined prior to the start of the calculation or during run-time, but they are not necessarily static. They can rely on the internal state of the application, or on information from external sensors. When the conditions for a switch have been met, the application will migrate to different grid resources, switch to a different task, or both.

The switching between tasks requires two steps, which are finalizing the old task (and any program it still uses) and starting up the new task. During this switch, the application-specific data should be left intact. The switching between sites requires a larger number of actions, which are:

1. Creating a set of files consisting of the current application, files with its parameters and data and a script that specifies the methods and conditions for switching and termination.

2. Creating a job definition for the application on the new resources.

3. Authenticating (independently) on the grid.

4. Transferring the files to the remote site (if this is not done automatically by a resource broker).

5. Submitting the job, either through a resource broker or by directly accessing the head nodes of grid sites.

6. Reinitializing the living application on the new site.

Additional file transfer may be required, if the application has locally written data that is required elsewhere. The application could initiate the transfer of output files either during run-time (e.g., if separate files are written) or just before a job terminates on one machine (if data is appended to a single large file or data transfer would cause overhead at run-time).

The living application requires some user privileges to initiate data transfers and to autonomously migrate from one site to another. We obtain these privileges by using a grid client interface to access a credential management service. The details of this method are discussed in Sec. 3.2.2.1. The application requires access to the grid client
interfaces on all participating nodes to request these privileges during execution. Once these privileges are granted, the application can perform authentication, data transfers and job submissions to the grid.

3.2.2.1 Security considerations

User privileges on the grid are provided by an X.509 grid proxy [131] which requires the presence of a certificate, a private key and a correct pass phrase typed in by the user. This proxy is represented by a temporary file with limited lifetime. The easiest way to provide user privileges to a living application would be to equip it with this file, transporting it as it migrates, allowing it to reuse the proxy on remote locations. However, this approach has three drawbacks:

First, the presence of a proxy file on a remote site poses a security risk. If the file is not read-protected or stored in a shared account, it may be possible for other grid users to copy the proxy. The possession of this proxy enables them to impersonate the living application user for the duration of the proxy’s lifetime, providing them with rights and resources that they could otherwise not use. Even if the proxy is on a dedicated account and read-protected, local users with admin rights are able to copy it and use it for impersonation.

Second, it is not possible to cancel the application after the first stage, as the proxy is initialized only at startup, after which it travels around on remote sites. This may cause a malfunctioning application to continue running and migrating until the proxy lifetime is exceeded. An application that is equipped for self-reproduction may iteratively spawns multiple successors which could lead to a grid meltdown.

Third, for the same reasons as before it is also not possible to prolong the lifetime of the proxy. This could cause the application to terminate prematurely once the proxy lifetime is exceeded. Specifying an excessively long lifetime relieves this problem, at the expense of increasing exposure to the other two drawbacks.

To reduce these drawbacks we have chosen to use an intermediary MyProxy server [15] in our implementation. The user initializes his or her proxy on the MyProxy server, which is encrypted using a unique password. This password is stored in the living application, which uses it to obtain short-lived user privileges from the MyProxy server. If the password is stolen, others may be able to get these short-lived privileges, but the user can remove access to these privileges at any time by destroying the credential.

During application execution, the user can also extend the lifetime of his MyProxy credential by renewing it. It is also possible to replicate the credentials to other MyProxy servers, which allows the application to use remote MyProxy servers if the local server has died, rather than terminating itself upon switching.

3.2.3 Living simulation

A special case of the living application is the living simulation. Today, simulations of complex systems, in which the dynamic range exceeds the standard precision of the computer, call for a wide range of numerical solvers [60]. Each of these solvers may run
most efficiently on a different computer architecture. Most such simulations, however, are run on a single computer even though they would benefit from running on a variety of architectures.

This can be solved by migrating the application at run-time from one computer to another, in other words, by creating a living simulation. Such a simulation loads the solvers as a library module and is able to probe the internal variables of these solvers, making migration decisions based on this information. We demonstrate the concept of the living application by applying it to the (living) simulation of two galaxies merging.

The term living simulation has been previously defined as simulations that fine-tune their behavior at run-time based on input from external sensors, for example to provide input for performing adaptive load balancing \[72\]. In our definition we provide the simulation with user privileges and expect it to function autonomously.

3.3 Simulating galaxy mergers as a living simulation

3.3.1 Motivation

A living simulation is based on the principle that it autonomously switches between sites and solvers whenever required. This switching is done dynamically and without external dependencies. The simulation is locally equipped with the required solvers, the switching criteria and the initial conditions. It is then submitted as a job to the grid with the initial resource requirements defined by the launcher. The living simulation begins calculating on the grid and continues to do so until either a switching condition or a termination condition has been met.

By using the idea of the living applications, we have implemented and tested a living simulation, in which the merger of two galaxies, each with a central supermassive black hole (SMBH), is simulated. This is a computationally expensive problem which requires integration with high accuracy during close encounters and in the final stages of merging, i.e. whenever the two SMBHs come close to each other. At an early phase and at large separation of the two galaxies, however, less accurate and therefore faster integration methods are sufficient. We improve the performance and the dynamic range of the tree code simulations (which are typically the method of choice for galaxy merger simulations) by hybridizing the tree code with a direct $N$-body solver.

In the scenario we are modelling, the two galaxies are initially well separated by hundreds of kiloparsec, but they approach each other on a bound orbit. Dynamical processes lead to a redistribution of energy and momentum which causes, among other things, the formation of tidal tails (see Fig 3.1). Eventually, these dynamical processes lead to the merger of the two galaxies.

In this merger, the two SMBHs, which reside in the galaxy cores, will be brought close together until they form a binary SMBH. Modelling the details of the formation of a binary SMBH and its subsequent evolution requires a very accurate integration. Therefore, we choose to switch from the tree code to a direct $N$-body solver at a prespecified separation $r_a$ between the two SMBHs (see also \[106\]). The switching allows us to follow the full galaxy merger. This would not be possible using a single
Living Simulations: Galaxy Merger using Direct and Tree N-body Integration

Figure 3.1: Simulation snapshot of a 260k particle simulation, where the two galaxies approach for an initial interaction.

solver due to the limited accuracy of the tree code and the computational costs of the direct method.

In our living simulation, we make use of a dedicated GRAPE (GRAvity PipE, [123]) special purpose computer to perform direct-method integration, and a graphics processing unit (GPU) to perform tree simulations. The living simulation initially integrates using a tree code on a GPU node, but switches to a direct integrator on a GRAPE node when the separation between the two SMBHs $r_{\text{SMBH}} < r_a$. The simulation switches back to the GPU once $r_{\text{SMBH}} \geq r_a$.

3.3.2 Implementation

We have used the Multiscale Software Environment (MUSE)\(^1\) [106] package to conduct our simulations. MUSE is a multi-scale/multi-physics astrophysical framework that connects a variety of astrophysical codes, enabling users to create combined simulations using Python scripts. The interfacing between existing solvers is realized using SWIG [17] with a uniformly defined interface for each domain. By writing scheduling scripts, users are able to access the different interfaces and create simulations that use multiple solvers for a wide range of astrophysical problems.

The modular approach of MUSE lends itself very well to the grid architecture. Modules run independently of each other and communicate through the scheduling script. A grid-enabled scheduler would then send each module to a different, suitable machine on the grid. Furthermore, many astrophysical solvers run most efficiently on dedicated and specialized computers. GRAPE boards, for example, have been used

---

\(^1\)see http://muse.li
extensively and very successfully in the field of stellar dynamics (e.g., see [45, 19, 16, 109]). In many cases, a MUSE application requires one or more specialized platforms to run on and is therefore best run on a grid of such specialized computers.

In previous work [106] we have extended MUSE with a grid interface, allowing users to transfer files and perform simulations on remote grid sites using a static and centralized scheduler which runs on the local user machine. The grid interface has currently been implemented using the PyGlobus API [68], and an alternative DRMAA-compliant interface is under development.

Our test implementation consists of two components, a launcher to initialize the living simulation and a job script that travels over the grid during simulation. The launcher:

- Loads MUSE and the required modules,
- reads the simulation input,
- stores the parameters for each solver and the initial data for the first simulation stage,
- transfers these files to the remote site, and
- submits the job script as a grid job to the remote site.

The living simulation grid job executes the Python job script, which:

- Initializes the simulation that will be used,
- reads and writes solver parameters and snapshots,
- uses MUSE and SWIG to execute a simulation,
- transfers files, and
- submits a job script that computes the next simulation stage.

The job script is able to periodically check internal variables of the local solver at run-time using MUSE and SWIG. Consequently, the script is sensitive to changes in these variables, and autonomously performs actions (e.g., migration to a different site or file transfers) if certain conditions are met.

### 3.3.3 Experiment setup

For our experiments we make use of two grid nodes, one node equipped with a GRAPE-6Af [36] at Drexel University in Philadelphia, United States and one node with an Nvidia 8800 Ultra GPU at the University of Amsterdam in the Netherlands. The GRAPE-6Af has a peak performance of approximately 123 Gflop/s and an effective performance of up to \( \sim 85 \) Gflop/s when performing a direct-method simulation [36]. The Nvidia 8800 Ultra has a theoretical peak performance of about 384 Gflop/s and a
Living Simulations: Galaxy Merger using Direct and Tree N-body Integration

Table 3.1: Specifications for the test nodes. The first column gives the name of the computer followed by its country of residence (NL for the Netherlands, US for the United States). The subsequent columns give the type of processor in the node, followed by the amount of RAM and the special hardware installed on the PC. The Linux distribution used on darkstar is Debian, while zonker uses the Gentoo distribution. Both nodes are connected to the internet with a 1 Gbit/s Ethernet card.

<table>
<thead>
<tr>
<th>name</th>
<th>location</th>
<th>CPU type</th>
<th>RAM [MB]</th>
<th>hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>darkstar</td>
<td>NL</td>
<td>Core2Duo 3.0GHz</td>
<td>2048</td>
<td>GeForce 8800 Ultra</td>
</tr>
<tr>
<td>zonker</td>
<td>US</td>
<td>2x Xeon 3.6GHz</td>
<td>2048</td>
<td>GRAPE 6A</td>
</tr>
</tbody>
</table>

sustained performance of up to $\sim$100 Gflop/s when performing an N-body tree code simulation using octgrav (E. Gaburov, personal communication). The specification of the nodes can be found in Table 3.1. On both nodes we have installed Globus 4.0.6 grid middleware [30] with GRAM, GridFTP and a MyProxy client, as well as the MUSE framework. The nodes are linked using a regular internet connection for which we have measured a latency of 100ms and a bandwidth of approximately 550 kB/s.

On these nodes we run galaxy collision simulations (using simplified galaxy models, see below) that each last for 20 N-body time units [54]. In all our runs, this duration was sufficient to perform a full collision between the two galaxies.

The initial conditions for the galaxy collision consist of two equally-sized Plummer sphere particle distributions [103], each of which has a central SMBH. We perform simulations with $N = 2k$ to $64k$ particles$^2$. The total mass of particles in each galaxy is $M = 1$ and the mass of individual particles is $m = M/N$. The SMBHs have each a mass of $m_{BH} = 0.01$ or 1% of the stellar mass of the galaxy.

When the two galaxies are far apart we use the tree code [14] in which further away particles are grouped together to enable a hierarchical reduction in the force computation. The equations of motion are solved using the 2nd order leap-frog particle integration scheme [59] with a fixed time step. The octgrav tree code we use is written to run on a graphical processing unit (Gaburov et. al., 2009, in preparation). The opening angle for the tree code is $\theta = 0.7$ and we use a time step of $1/64$ N-body time unit (1/128 for the largest data set). The direct-method integration is performed using phiGRAPE [50]. In phiGRAPE, particles have individual (block) time steps and the time step parameter $\eta$ was set to 0.02 [85]. We also defined a maximum time step of $2^{-5}$ and a minimum time step of $2^{-23}$ N-body time units. A softening of $\epsilon = 0.01$ is used in both integration methods.

We have performed two profiling experiments, using direct integration whenever the separation of the central black holes was less than $r_a$, and tree integration at all other times. The first experiment varies in the number of simulation particles, while maintaining $r_a = \sqrt{0.3}$. The other experiment uses 32k particles and a different $r_a$ for each run. For comparison, we have also included a full tree and a full direct run.

$^2$i.e. 1024 to 32768 particles per galaxy as well as 2 SMBH particles.
3.3 Simulating galaxy mergers as a living simulation

Figure 3.2: Timing measurements of the living simulation tasks as a function of the number of simulated particles. The two solid lines represent time spent on direct integration (bullets) and tree integration (circles). The thick dashed lines indicate grid overhead by data transfers (open squares) and job submissions (filled squares). Finally, the two thin dashed lines indicate overhead caused by local file I/O (open triangles) and code initializations (filled triangles).

3.3.4 Results

We have summarized the results of our living simulation in two figures. The absolute time spent on each task as a function of the number of particles is given in Fig. 3.2, and the relative time share of each task is shown in Fig. 3.3. For all the tested initial conditions, the simulation migrated itself three times, resulting in four initializations and three simulation migrations per run.

In this experiment, we find that the direct $N$-body integration dominates the simulation performance in all cases, and that for larger $N$, the relative overhead caused by grid data transfers and job submissions diminishes. Although the time spent on local I/O scales steeply due to unoptimized identifier lookup calls (this has recently been fixed in MUSE), this overhead remains relatively small throughout our runs. When using 64k particles, we found that $\sim 4$ percent of the simulation time is spent on overhead tasks.

We have performed several runs with 32k particles, using a different $r_a$ for each run. The results of this experiment are shown in Tab. 3.2. During the runs we observed several close interactions between the SMBHs, and a decreasing trend in the value of $r_{SMBH}$. This behavior caused the living simulation runs with smaller $r_a$ to switch more
Living Simulations: Galaxy Merger using Direct and Tree N-body Integration

Figure 3.3: Relative cumulative share of time spent by the living simulation tasks as a function of the number of simulated particles. From top to bottom the areas refer to the share of time spent on direct integration, tree integration, local file I/O, grid data transfer, grid job submissions and simulation initializations. Note that both axes are in log-scale.

Table 3.2: Timing and energy measurements of the living simulation tasks using 32k particles with a different value \( r_a \) during each run, given in the first column. The second column gives the number of switches during the simulation, while the subsequent columns respectively give the times spent on direct integration, tree integration and overhead tasks. The total execution time and the total relative energy error are respectively given in the last two columns.

<table>
<thead>
<tr>
<th>( r_a )</th>
<th># switches</th>
<th>direct [s]</th>
<th>tree [s]</th>
<th>other [s]</th>
<th>total [s]</th>
<th>( dE/E )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 (tree)</td>
<td>0</td>
<td>0</td>
<td>247</td>
<td>24</td>
<td>271</td>
<td>1.47 ( \cdot ) 10(^{-2} )</td>
</tr>
<tr>
<td>0.1</td>
<td>29</td>
<td>762</td>
<td>219</td>
<td>944</td>
<td>1925</td>
<td>5.93 ( \cdot ) 10(^{-3} )</td>
</tr>
<tr>
<td>( \sqrt{0.1} )</td>
<td>7</td>
<td>1820</td>
<td>160</td>
<td>257</td>
<td>2237</td>
<td>3.54 ( \cdot ) 10(^{-3} )</td>
</tr>
<tr>
<td>( \sqrt{0.3} )</td>
<td>3</td>
<td>2180</td>
<td>143</td>
<td>120</td>
<td>2443</td>
<td>2.88 ( \cdot ) 10(^{-3} )</td>
</tr>
<tr>
<td>1.0</td>
<td>3</td>
<td>2519</td>
<td>127</td>
<td>118</td>
<td>2764</td>
<td>2.49 ( \cdot ) 10(^{-3} )</td>
</tr>
<tr>
<td>( \sqrt{10} )</td>
<td>1</td>
<td>3624</td>
<td>64</td>
<td>54</td>
<td>3742</td>
<td>1.04 ( \cdot ) 10(^{-3} )</td>
</tr>
<tr>
<td>( \infty ) (direct)</td>
<td>0</td>
<td>4528</td>
<td>0</td>
<td>5</td>
<td>4533</td>
<td>2.77 ( \cdot ) 10(^{-6} )</td>
</tr>
</tbody>
</table>

frequently.
A pure tree integration ($r_a = 0$) leads to the highest cumulative energy error, whereas a pure direct integration ($r_a = \infty$) has the lowest error. When switching between both codes with the living simulation, the energy error is lower than using pure tree, but much higher than using a direct code. Even when using a $r_a = \sqrt{10}$, where the code switches only once after 4 $N$-body time units, we see a much larger error than when using only direct integration. The energy error is dominated by the execution of the tree code. This difference is caused by the tree-based force calculation as well as by the second-order leapfrog integration scheme used in the tree code. A detailed discussion on the energy behavior of these combined simulations can be found in Harfst et al. (2009, in preparation).

The simulation performance is dominated by $N$-body integration in all cases, although there is a relatively high overhead for $r_a = 0.1$, which is caused by the 29 switches. Each of these switches requires the particles to be saved locally, sent across the Atlantic using regular internet, and loaded on the new machine.

### 3.4 Conclusion

We introduced the living application as a way to manage complex applications on a large distributed infrastructure. Due to the autonomous nature of a living simulation, it is important to provide a mechanism that allows the user to terminate it. By having the simulation retrieve its extended privileges from a credential management service (MyProxy), users are able to revoke the privileges of the simulation regardless of its location. In addition, we can renew short-lived proxy credentials instead of using a long-lived credential, which may be attractive to malicious users.

We then apply this concept in a living simulation of two galaxies merging, using a straightforward and autonomous resource selection scheme which chooses from a predefined list of available resources. Our approach allows the simulation to use the optimal compute resources for each of the two solvers, switching resources whenever a different solver is required. In our example, the solvers were a tree code and a direct $N$-body method, which were optimized for two kinds of special-purpose hardware, namely a GPU (tree) and a GRAPE (direct). The switches take place autonomously without user intervention, remote output retrieval or external managers. In our experiments, the execution time was only affected marginally by overhead such as caused by job migration and data transfer over the grid. In the cases where each solver is best run on a different architecture and the overall simulation performance is not dominated by switching overhead, we find that the living simulation is a practical and resource efficient solution.

The creation of grid species enables us to give a simulation the ability to autonomously use the grid, acquire and apply internal knowledge, and migrate themselves. In this work we presented a first implementation, which we intend to extend in the near future. Possible extensions include connecting living applications with grid resource monitoring and discovery services to dynamically obtain information on resource availability, and developing a living application which is able to recover from failures of grid
nodes. These extensions allow us to apply the living application to evolve to a more complex organism, which can be applied to problems of greater complexity.
In this chapter we present our TreePM $N$-body simulations on a planet-wide network of supercomputers connected by dedicated lightpaths. The experiments were performed using the MPWide communication library.

4.1 Introduction

Since the beginning, the dimensions and complexity of the universe have been increasing continuously. As a consequence we observe today large structures consisting of galaxies. The visible superstructure of the universe is made of baryonic material, e.g., stars and gas. But the majority of the mass is in the form of dark matter, which is affected by gravity but does not interact electro-magnetically. The best model for the formation and evolution of this superstructure is called Λ cold dark-matter cosmology (ΛCDM)[48], according to which the universe is about 13.7 billion years old and comprises of about 4% baryonic matter, $\sim 23\%$ non-baryonic (dark) matter and 72% (dark) energy, indicated by the letter Λ [119]. The nature of dark matter is unknown and from an observational perspective it is hard to resolve this lacuna in our understanding. Evidence for dark matter comes from a variety of observations, which includes the rapid rotation of the Milky-Way and other galaxies: its stars would be flung out in the absence of dark matter.

Weakly-interacting massive particles form the most promising explanation for dark matter [74], in which case the entire universe is filled with a finely grained substance
that behaves like a gravitational fluid but is invisible otherwise. The gravitational force in the Newtonian limit $\propto 1/r^2$, which is a long-range force in particular since the enclosed mass scale $\propto r^3$. Together with the absence of a shielding mechanism, even very distant objects cannot be ignored. The way in which dark matter interacts is therefore rather simple and well understood. As a result, we can effectively study dark matter by simulation and use these results to understand observations and to make predictions.

One of the most favorable techniques to study the formation of large scale structure in the universe is by means of gravitational $N$-body simulations, in which each particle in the simulation represents the fluid of dark matter particles. The most widely used algorithm is the treePM method, in which the short range forces are resolved using a Barnes-Hut tree code, whereas the long range interactions are simulated using a Particle-Mesh method [133, 59]. This combination of methodologies provides good performance compared to using only a tree code, but still at a reasonable accuracy compared to a pure particle-mesh technique. This is because in the treePM method we resolve short as well as long range forces reasonably well. In addition, the algorithm scales well to a large number of processors by optimized domain decomposition (Ishiyama et al. in press). Since simulating the entire universe is not really possible (yet) we study a small part, using periodic boundary conditions to mimic ad infinitum.

The computational demand for our large scale cosmological $N$-body simulation is enormous, and instead of running on a single supercomputer, we opted for running concurrently on two widely separated supercomputers. We demonstrate that it can be efficient to run high-performance production simulations on a computational grid.

4.2 The intercontinental grid

The future of large-scale scientific computing infrastructures aims at distributing resources rather than concentrating supercomputers locally [60]. The higher cost effectiveness of distributing resources can be efficient for those applications in which the compute time scales steeper than the communication, or when the problem can be decomposed in domains [31].

Instead of starting the simulation on one location and switch half way to another computer to continue the calculation, we run the simulation on two supercomputers concurrently. One of our computers is located in Amsterdam (the Netherlands) and the other is in Tokyo (Japan). The challenge is to efficiently use a large number of processors separated by half the planet without running in the inter-processor and intercontinental communication bottlenecks. If we can run successfully among two supercomputers separated by half the planet, we can be confident about upscaling our virtual organization to include more than two supercomputers.

The management, political and technical issues of coupling the computers with an uncongested network and scheduling the resources proves to be extremely challenging, and one can wonder if it is worth the effort. The preparations for our calculations lasted about a year. However, running on a single supercomputer would have required
its entire capacity for several months, which would probably not have been granted. Acquiring relatively little time on a large number of supercomputers proves to be considerably easier. With the expertise obtained in running on two supercomputers we can now extend to more sites without much additional overhead. Of course, the political and technical issues of coupling the computers with an efficient network and scheduling the resources remain challenging, but many of these aspects will become easier once high-performance grid computing becomes mainstream. Additional complications arise by the required hybrid parallelization strategies, the diversity in topologies, scheduling, load balancing and the complications introduced by the different hardware architectures in particular if, as in our case, the code is machine dependent.

One of the complications we encountered is related to the network topology and internal hardware setup. Neither supercomputer is directly connected to the optical network, but communication is realized via a special node that is connected to the outside world with a 10GbE optical switch in Amsterdam and Neterion NIC in Tokyo. Upon every communication step each processor: identifies the particles which need to be communicated, packs them and sends the particles to the supercomputer’s internal communication node, which subsequently sends the entire package of particles to the communication node outside the firewall. This package is subsequently transmitted to the distant computer 9,400 km away, as the bird flies. The data however, travels \( \sim 27,000 \) km as the network links criss-crosses the Atlantic ocean, the USA and the Pacific ocean. In Fig. 4.1 we illustrate the network topology. With the speed of light in fiber this distance is covered in 0.138 seconds, resulting in a round-trip time of 0.277 seconds.

The network communication was realized by configuring two virtual LANs (vLAN) to create two paths connecting both supercomputers. One vLAN was used for the production traffic and runtime synchronization, while the second was configured for testing purposes and for collecting the simulation data at the PByte storage array in Amsterdam. Both paths are dedicated to our experiments which prevents packet loss, packet reordering and latency changes due to congestion. In the secondary path we had to execute vLAN translation in the JNG network, because the chosen vLAN numbers were unavailable along the entire path length. The choice of an available vLAN number for the end-to-end communication should have been trivial; but since our multi-domain light path lagged automatic configuration tools, human intervention was necessary. The time required for solving problems depends on the quick responses to email and phone calls, which is hindered by working over several time zones; debugging link failure at Layer2 in a multi-domain multi-vendor infrastructure is impractical and extremely time consuming. A self-healing or automatic setup would enormously help future experiments.

The Research and Education Networks (REN) provided the links for free, which is motivated by their vision to advertise and broaden their services to the scientific community. In the last few years many RENs have adopted a hybrid model for their architecture, expanding it to routed IP services where users have access to dedicated light paths in which communication occurs at lower layers of the open system-interconnection reference model. We settled for a flat Ethernet path, but still had to make the
Figure 4.1: Network topology for the CosmoGrid experimental setup. We used two main network connections between JGN2plus and StarLIGHT: one of then goes via L-LEX and Gigapop, the other connects directly from StarLIGHT to JGN2plus. Both paths have a network bandwidth of 10Gb/s, carried over 10 GE-LANPHY and 10 GE-WANPHY segments depending on the type of interfaces present at the various switching points. Thick blue lines (LANPHY) and thick black lines (WANPHY).

network reservations 3 weeks in advance of each simulation restart, and then it would generally take about a week before it was fully operational. Our overall experience with the use of light paths however, is quite positive. We could not have performed our calculation without this type of network setup and certainly perceived the advantage of the unrestricted and exclusive use of the links.

4.2.1 Demand on the network

While performing a cosmological $N$-body simulation in which the computational domain is shared by two (or more) computers, the positions and velocities of all particles on both machines have to be synchronized throughout the simulation. This introduces an enormous demand on the network between the computers. Luckily only a small boundary layer between the two halves of the universe and the layer nearest the periodic boundary are communicated between Amsterdam and Tokyo. The amount of data that has to be communicated per step is then

$$S_{\text{comm}} = 144N^{2/3} + 4N^3 + 4N/S_r \text{ Byte.} \quad (4.1)$$
Here $N$ is the number of particles, $N_p$ is the resolution of the mesh in one dimension and $S_r$ is the sampling rate, which for the production simulation $S_r = 5000$. The first term is required for communicating the tree structure, the second term is for exchanging the particles in the border region, and the last term is for exchanging the mesh. The first term in this estimate depends slightly on redshift ($z$) and on the opening angle in the treecode ($\theta$), but is accurate for $\theta = 0.5$.

While running, dense dark-matter clumps may not be distributed evenly across the computational domain and load balancing is done by guaranteeing that the calculation time per step is the same on each computer, generating a variable boundary layer between the two computers. Where initially both computer resolve exactly half the universe, in due time one of the two computers tends to deal with a larger volume.

The communication within each of the supercomputers is realized by domain decomposition, using the Message Passing Interface (mpich [43]). To warrant efficient and stable data transfer we developed the parallel socket library MPWide to facilitate the communication outside the local MPI domains (see Chapter 6). This communication consists of data transfers between the local compute nodes and the local communication node, and for the data transfer over the light-path between the supercomputers.

The socket library is included in all processes, providing an interface similar to regular MPI. A static communication topology is established at startup which uses multiple tcp connection for each intra-cluster communication path and multiple tcp connections for paths between supercomputers. The concurrent use of multiple streams is realized by running a separate thread for each stream. For the smaller runs we used 16 tcp streams, and 64 streams for the larger runs (see Tab. 4.1).

### 4.3 The simulation environment

We adopted the treePM code GreeM which was initially developed by [134] and rewritten by Ishiyama et al (in press) to run efficiently on the selected hardware. The equations of motion are integrated in co-moving coordinates using the leap-frog scheme with a shared but variable time-step. Good performance at sufficient accuracy is then achieved when the time step is at least an order of magnitude smaller than the crossing time in the densest halo [59]. Spurious relaxation effects in the high-density clumps are prevented by introducing a Plummer softening to the force, which is comparable to the local inter-particle distance.

Since most simulation time is spent in calculating the gravitational forces between particles, we optimize this operation using the single precision X86-64 Streaming SIMD Extensions (SSE). The inverse square-root SSE instruction provides the best speedup in calculating Newtonian gravity. We further improve performance by minimizing RAM access through the 16 XMM registers for the force operations, and by operating on pairs of two 64-bit floating point words concurrently (Nitadori & Yoshikawa in preparation).

With these optimizations the Power6 with symmetric multiprocessing is per core about 4.2% slower than the Intel based Cray XT4. This small difference in speed is achieved by adopting the x86 SSE version and the Power AltiVec architectures, but
Table 4.1: The performed simulations in a computational box of 30 Mpc starting at $z \simeq 65$, a softening of 300 pc and the opening angle in the tree code $\theta = 0.3$ for $z > 10$ after which $\theta = 0.5$. The first column gives the number of particles in the simulation followed by the number of mesh-cells in one dimension. Then we give the number of processors in Amsterdam (A) and Tokyo (T). The next two columns give the average wall-clock time for one step ($t_{\text{tot}}$) and the time spent computing the forces ($t_{\text{cpu}}$), both in seconds. The last two columns give the speed-up $\eta_p$ which is defined as the wall-clock time of a single computer as fraction of the wall-clock time of the grid of supercomputers. Here we define $\eta_p$ for a grid where each computer has the same number of processors $p$. For the last two entries $\eta_p$ is then not properly defined.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$N_p$</th>
<th>$p$ (A+T)</th>
<th>$t_{\text{tot}}$</th>
<th>$t_{\text{cpu}}$</th>
<th>$\eta_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>16777216</td>
<td>128</td>
<td>30+30</td>
<td>31.2</td>
<td>23.58</td>
<td>1.51</td>
</tr>
<tr>
<td>134217728</td>
<td>256</td>
<td>30+30</td>
<td>116.8</td>
<td>105.7</td>
<td>1.81</td>
</tr>
<tr>
<td>134217728</td>
<td>256</td>
<td>60+60</td>
<td>71.9</td>
<td>58.8</td>
<td>1.64</td>
</tr>
<tr>
<td>134217728</td>
<td>256</td>
<td>120+120</td>
<td>53.0</td>
<td>36.8</td>
<td>1.39</td>
</tr>
<tr>
<td>1073741824</td>
<td>256</td>
<td>500+250</td>
<td>60.0</td>
<td>40.0</td>
<td>—</td>
</tr>
<tr>
<td>8589934592</td>
<td>256</td>
<td>500+250</td>
<td>380.0</td>
<td>310.0</td>
<td>—</td>
</tr>
</tbody>
</table>

for the former we use in-line assembly whereas for the latter we adopted the intrinsic functions (Nitadori & Yoshikawa in preparation).

We measure the performance of the code using the Amsterdam and Tokyo machines separately with $p = 1$ to $p = 1024$ processors and $N = 256^3$ to $N = 1024^3$ particles. The wall clock time $t_{\text{cpu}}$ is then fitted by $t_{\text{cpu}} \simeq 14,500p^{-0.91}$ seconds, with a ten fold increase of $t_{\text{cpu}}$ for every increase of the number of particles by a factor of $2^3$. For relatively small $N$ we lose scalability with respect to the number of processors for $p \simeq 10^3$ and $N \lesssim 256^3$.

### 4.4 Simulating the universe

We are interested in structures of kpc to Mpc size. To minimize the effect of the periodic boundary conditions on such dark matter distributions we selected our simulation box at $z = 0^1$ to have sides of 30 Mpc. The initial dark-matter distribution is generated at $z \simeq 65$, assuming that the relation between the velocity and the potential in Zel’dovich approximation is the same as in the linear theory [69]. The density field is then realized by multi-scale Gaussian random fields, which is described in terms of its power spectrum and was generated using MPGRAFIC [113].

We further adopted cosmological parameters which are consistent with the 5-year WMAP results [28]. For clarity we opted for the nearest round values, which yields: matter (including dark) density $\Omega_m = 0.3$, dark energy density $\Omega_A = 0.7$, the slope for the scalar perturbation spectrum $n_s = 1.0$, and the amplitude of fluctuations $\sigma_8 = 0.8$.

---

1Redshift is the standard measure of time in cosmology: the universe was born $z \rightarrow \infty$ and today $z \equiv 0$. 

---
4.5 Concluding remarks

With these parameters the universe is about 13.7 billion years old and the Hubble constant $H_0 \simeq 70 \text{km/s/Mpc}$.

We ran several realizations at different resolution (in mass as well as spatially) to measure the performance before we completed a simulation to $z = 0$. An overview of the performed simulations is presented in Tab. 4.1.

The simulation with $N = 16777216$ took about 10 hours to complete from $z \simeq 65$ to $z = 0$. In Fig. 4.2 we show the wall-clock time of this run decomposed in the time spent in calculation, communication and storing the data to the file system. About 25% of the time is spent in communication (see Tab. 4.1), which is roughly constant throughout the simulation. We encountered a few problems between $z \simeq 16$ and $z \simeq 15$, which are probably related to packet loss in the network transport protocol suite and resulted in an additional performance loss of a few per-cent.

During the communication the speed of data transfer averages 21.1Mbit/s with peaks of about 7Gbit/s and per step between 16MB and 26MB is transferred. For the larger runs the average throughput increases to about 208Mbit/s since the size of the packages increases with $N$ (see Eq. 4.1).

The result of the simulation with $N = 16777216$ at $z = 0$ is presented in two panels in Fig. 4.3. The two panels show the universe as it is seen by the two supercomputers, with the black parts on the right of the left panel (left on the right image) indicating the part of the universe that resides on the other machine.

4.5 Concluding remarks

The formation of large scale structure in the universe can be studied effectively by means of simulation but requires phenomenal computer power. Even with fully optimized numerical methods and approximations the wall-clock time for such simulations can exceed several months. Our largest simulation with $N = 2048^3$ has been running for about 1 million CPU hours (more than 1 month on 1024 processors) to reach $z \simeq 1.5$ and we expect to spend another $\sim 2$ million CPU hours to reach $z = 0$.

The runs in Tab. 4.1 are performed on a grid of supercomputers. With our implementation the latency and throughput of the communication poses a relatively small overhead to the calculation cost, in our largest simulation the communication overhead $\lesssim 10\%$. We have therewith demonstrated that large-scale structure formation simulations are excellently suited for high-performance grid computing.

The adopted setup would in principle have reduced our wall-clock time by almost a factor of two compared to running on a single supercomputer, and our simulations in Tab. 4.1 have indeed effectively benefitted from using the grid. However, we have spent more than a year preparing and optimizing the code, and acquiring and scheduling the resources, none of which proved to be trivial. On the other hand, if we would not have opted for running on a grid it would have proven extremely difficult to perform the simulations at all since acquiring 1024 processors on a supercomputer for half-year via a ‘normal’ proposal would be challenging. Even acquiring half the resources on two supercomputers with the promise to switch computers half-way the simulation would
be difficult. Our strategy to run on a grid enabled us to secure the required CPU time on both supercomputers.

During this project we all became very enthusiastic about the computational grid as a high-performance resource, despite the time we have spent with preparations. The success of our grid is in part a consequence of the realization that latency forms no bottleneck, even if the computers are separated by half the planet. We cannot beat the speed of light, but bandwidth is likely to improve with time, making high-performance grid computing very attractive in the foreseeable future. Much work, however, is needed in improving practical matters, like scheduling issues, network acquisition and cooperation between supercomputer centers, each of which are major bottlenecks. We seriously consider to perform our next run on a grid with many supercomputers.

Based on our measurements for each of the two supercomputers and the intercontinental grid we constructed a performance model, which is composed of two main components: the calculation ($t_{\text{cpu}}$) and the data transfer over the grid $t_{\text{comm}} = \nu t_{\text{lat}} + S_{\text{comm}}/b$, where $t_{\text{lat}}$ is the network latency and $b$ is the network throughput (see Eq. 4.1). The parameter $\nu$ is introduced to correct for the inefficiencies in our code which require several transmissions ($\nu$) to the other computer. In Fig. 4.4 we present the results of the performance model based on the characteristics of the adopted computers, network and software environment for $N = 2048^3$. The speedup is limited by the bandwidth, whereas latency, for which we adopted $t_{\text{lat}} = 0.138s$ with $\nu = 6$, poses no limiting factor to our simulations. Reducing the total latency will hardly improve the performance, but improving the throughput by a factor 10 would allow us to use an order of magnitude more processors per supercomputer while still acquiring acceptable speedup. The two topologies adopted in Fig. 4.4 are selected based on an ideal setup where the supercomputers are interconnected via a ring topology. A sub-optimal star topology leads to congestion as the communication tends to go through a single site. With the optimal topology (solid curves in Fig. 4.4) running on a 10 supercomputers with $\gtrsim 128$ processors each results in a speed-up of a factor of $\sim 8$, whereas running on 100 supercomputers the speedup would be only a factor of 10.

The best strategy for performing cold dark matter simulations using a treePM code on a grid appears to be by acquiring $p \lesssim 100$ processors on $n_{\text{sc}} \sim 10$ supercomputers in a ring network topology. The main reason for adopting this strategy would be to be able to acquire the required compute resources; it is considerably easier to obtain $p \sim 100$ processors for an extended period ($\sim 1$ year) from a dozen supercomputer centers, than to acquire the required CPU hours on a single supercomputer. This strategy, however, would require drastic changes in the co-scheduling policy of the supercomputer centers.

With our cosmological cold dark matter simulation we make the dream of Foster and Kesselman [31] come true, our calculation benefits from having multiple widely separated computers interconnected with a high-bandwidth network, effectively operating as a single machine.
Figure 4.2: Wall-clock time as a function of redshift ($z$) for the simulation with $N = 16777216$. The solid curve gives the wall-clock time, including the network wait stages, the dashed curve gives calculation time and the dotted curve indicates the time spend communicating. The lower thin dotted curve gives the time for storing data.
Figure 4.3: Final snapshot of the simulation with $N = 256^3$ with Tokyo (left) and Amsterdam (right). The black part of each figure represents the memory share that is located at the other site.
4.5 Concluding remarks

Figure 4.4: The speedup ($\eta_p$) for a grid of $n_{sc}$ supercomputers each with $p$ processors. The speedup is defined as the wall-clock time of the grid computer as fraction of the wall-clock time of one single supercomputer, assuming that each computer has $p$ processors. The thick curves are calculated with $p = 128$ the thin curves with $p = 1024$. The solid curves are calculated assuming the ring topology network, whereas for the dotted curves we adopted a star connected network. We adopted an average bandwidth of $b = 200$ Mbyte/s with a latency of $\nu t_{\text{lat}} = 0.828$ s for the external (grid) communication and 10Gbyte/s for internal communication. The diagonal thin dashed curve indicates the ideal scaling.
High Performance Gravitational $N$-body Simulations on a Planet-wide Distributed Supercomputer

Based on:
D. Groen, S. Portegies Zwart, T. Ishiyama and J. Makino
High Performance Gravitational $N$-body simulations on a Planet-wide Distributed Supercomputer,
submitted to Computational Science and Discovery.

In this chapter we present our TreePM $N$-body simulations on a planet-wide network of supercomputers connected by dedicated and shared lightpaths. The experiments were performed using the MPWide communication library.

5.1 Introduction

Some applications for large scale simulations require a large amount of compute power. This is often hard to acquire on a single machine. Combining multiple supercomputers to do one large calculation can lift this limitation, but such wide area computing is only suitable for certain algorithms. And even then the political issues, like arranging the network, acquiring the compute time, making reservations, scheduling runtime and synchronizing the run start, and technical limitations are profound. Earlier attempts based on interconnecting PC clusters were quite successful [29, 46, 89, 12, 13], but lacked the raw supercomputer performance required for our application.

Running simulations across multiple supercomputers has been done a few times before [98, 112, 121, 104], though the performance of simulations across three or more supercomputers has not yet been measured in detail. Here we report on the performance of our parallel astronomical simulations which use up to 4 supercomputers and predict the performance for simulations which use 5 or more supercomputers.

In our experiments we use an international infrastructure of supercomputers. These machines include an IBM Power6 supercomputer located at SARA in Amsterdam (the
High Performance Gravitational N-body Simulations on a Planet-wide Distributed Supercomputer

Netherlands) and three Cray XT-4 supercomputers located at the Edinburgh Parallel Computing Centre in Edinburgh (United Kingdom), the IT Center for Science in Espoo (Finland) and the Center For Computational Astrophysics in Tokyo (Japan). The machines are connected to an optical network which uses more than 32,000 km of cable in total. The Edinburgh site is equipped with a 1 Gbps interface while the other three sites are equipped with a 10 Gbps interface. We achieved a peak performance of 0.610 TFLOP/s and a sustained performance of 0.375 TFLOP/s using 120 cores distributed over 4 sites. To provide a comparison with the international tests we also run the code over up to 5 sites on a national grid of Beowulf clusters. Our wide area simulations are realized with the development of a software environment for Simulating the Universe Structure formation on Heterogeneous Infrastructures, or SUSHI for short.

5.2 Overview of SUSHI

Our code is based on the GreeM cosmological N-body integrator, which was originally developed for special-purpose GRAPE hardware [134]. The code integrates the equations of motion for dark matter particles using a shared and adaptive time step scheme and a Tree/Particle-Mesh (TreePM) force calculation method [133] which assumes periodic boundary conditions. The short range force interactions are resolved using a Barnes-Hut tree algorithm [14] while the long range interactions are resolved using a Particle Mesh (PM) algorithm [58].

The tree integration method places particles in a three-dimensional sparse octree structure, where each cell contains the center of mass and the mass aggregate of the particles therein. The method then resolves long range force interactions using particles and tree cells instead of using direct particle-particle evaluation. The accuracy of the tree integration method can be tuned by changing the opening angle ($\theta$), which determines how small and distant a group of particles needs to be to use the approximate particle-tree cell evaluation. A higher value for $\theta$ results in fewer particle-particle evaluations, and a lower accuracy of the simulation. Particle integration using a tree algorithm is more compute-intensive than integration using a PM algorithm, but we speed up the calculations by a factor $\sim 8$ using assembler-level optimizations [97]. The PM algorithm maps the particles to a grid of mesh cells and calculates the gravitational potential using a Fast Fourier Transform. It accurately calculates the forces of long distance interactions, but is less accurate in computing forces over short distances, for which the tree algorithm is applied instead.

The code has been modified to allow simulations on massively parallel machines [65], in which the code uses a recursive multi-section scheme [84] to divide the workload over the processes. The workload is redistributed during each step of the simulation so that the force calculation time remains equal for all processes.

5.2.1 Parallelization across supercomputers

We have developed SUSHI to efficiently use multiple supercomputers for our simulations. We coupled the TreePM code with the MPWide communication library and developed
a cross-supercomputer parallelization scheme. Because the wide area network has performance and topological characteristics that are different from local networks, the communication scheme between sites is different from the scheme used between nodes. When SUSHI is deployed across sites, each site is connected to two neighboring sites to form a ring topology.

5.2.1.1 Communication scheme

A simulation using SUSHI consists of four communication phases per step. During these phases the simulation:

1. Exchanges mesh densities.
2. Collects sample particles to determine the site boundaries,
3. Exchanges tree structures with neighboring sites.
4. Migrates particles between neighboring sites.

When exchanging mesh densities, the mesh cells from all sites are aggregated to obtain the global mesh density. This mesh density is then used to perform PM integration. At the time of writing the PM integration is still a serial operation, but we are in the process of implementing a parallelized version. The mesh densities are gathered using a ring communication over all sites. The force calculation time and time step information of each site are also accumulated during this phase.

At each step the site boundaries are updated, based on the obtained force calculation times and the current particle distribution. To gain information on the particle distribution, the communication processes on each site gather sample particles from all other processes. These sampled particles are then gathered from all sites using a ring communication.

Before the tree force calculations can be performed, each site constructs a local essential tree structure. This local essential tree is a set of particles and tree cells which are used to compute the force exchanges, and partially resides on neighboring sites. To obtain a complete local essential tree, each site therefore requires the missing tree data from its neighbors. The simulation gathers the tree data using one neighbor exchange for two-site runs, or two exchanges for runs across three or more sites.

After the force calculations have been performed, the simulation updates the positions of all particles. At this point, some particles may be located outside the site boundaries and need to be migrated to a neighboring site. This communication requires one neighbor exchange for two site runs, or two exchanges for runs across three or more sites.

5.2.1.2 Domain decomposition

We have implemented a hierarchical decomposition scheme to distribute the particles among supercomputers. This scheme uses a one-dimensional slab decomposition to
distribute the particles among the sites, and a recursive multi-section scheme over three dimensions to distribute the particles among the processes. Because the domain decomposition between sites is one-dimensional, each supercomputer only exchanges particles and tree structures with two other machines. The data exchanges between sites can therefore be done efficiently in a ring topology. Most supercomputers are connected by optical paths in either a ring or star topology, if they are connected at all.

The simulation adjusts the distribution of particles among supercomputers at run-time, such that the force calculation time is kept equal on all sites. The number of particles \( N_i \) on a single site \( i \) for a run performed over \( s \) sites is therefore given by

\[
N_i \sim \frac{N (t_{\text{calc}, i})^{-1}}{\sum_{j=0}^{s} (t_{\text{calc}, j})^{-1}}.
\] (5.1)

The force calculation time on site \( i \) during the previous step is given by \( t_{\text{calc}, i} \). The load balancing algorithm can be suppressed by explicitly limiting the boundary moving length per step, thereby limiting the communication traffic between sites.

5.2.1.3 Implementation of communication routines

We use the MPWide communication library, which is described in Chapter 6, to perform wide area message passing within SUSHI. The implementation of the communication routines has few external dependencies, which makes it easy to install on different platforms. The wide area communications in MPWide are performed using parallel TCP streams. In cases where the supercomputers can only be indirectly connected, we use MPWide-based port forwarding programs on the intermediate nodes to establish a communication path. During the development of SUSHI, we found that it is not trivial to obtain optimal communication performance between supercomputers. Therefore we added several features that can be used to improve the communication performance. The communication routines within SUSHI can be customized for individual paths between supercomputers. Settings that can be adjusted for each connection include the number of parallel TCP streams, the TCP buffer sizes and the size of data packages that are written to or read from the sockets. To improve the performance on some long distance networks, MPWide also supports software-based packet pacing.

5.3 Performance model

We have developed a performance model for SUSHI. The model can be applied to predict the execution time and scalability of simulations that run across supercomputers. To make an accurate prediction we require several architecture-dependent parameters. These include machine-specific parameters such as the time spent on a single tree interaction (given by \( \tau_{\text{tree}} \)) and a single particle mesh interaction (\( \tau_{\text{pm}} \)). In addition, we need a few parameters for the networks used in the simulation, which are the round-trip time (given by \( \lambda_{\text{lan}} \) and \( \lambda_{\text{wan}} \)) and the available bandwidth (\( \sigma_{\text{lan}} \) and \( \sigma_{\text{wan}} \)). The values
5.3 Performance model

of these parameters can be obtained through minor network tests and a small single
process test simulation on each site. The values used for our experiments are found in
Tab. 5.5 for each supercomputer, in Tab. 5.2 for each national grid site and in Tab. 5.6
for the local and wide area networks.

5.3.1 Single supercomputer

The time required for one TreePM integration step using \( p \) processes on a single
supercomputer \((t_{\text{exec}}(1,p))\) consists of time spent on tree force calculations \((t_{\text{tree}})\), time
spent on PM integration \((t_{\text{pm}})\) and the communication overhead \((t_{\text{comm}})\):

\[
t_{\text{exec}}(1,p) = t_{\text{tree}} + t_{\text{pm}} + t_{\text{comm}}.
\]  

(5.2)

The time spent on tree integration \((t_{\text{tree}})\) is dominated by force calculations. The
force calculation time is obtained by multiplying the time required to perform a single
force interaction \((\tau_{\text{tree}})\) with the total number of tree interactions \((n_{\text{int}})\) and dividing it
by the number of processes \((p)\). Creating interaction lists and constructing the tree
introduce additional overhead that scales with the number of interactions. To account
for this in a simplified way, we multiply the time spent on force calculations with a
factor 1.2\(^1\). The time spent on tree integration then becomes

\[
t_{\text{tree}} = 1.2 \times \frac{\tau_{\text{tree}} n_{\text{int}}}{p}.
\]  

(5.3)

The number of interactions per simulation step depends on many parameters
including, but not limited to, the number of particles \((N)\), the opening angle of the
tree integration \((\theta)\) and the number of mesh cells \((M)\). We have performed several
runs over a single supercomputer and fitted the number of interactions for cosmological
datasets, which results in

\[
n_{\text{int}} = 4.6 \times 10^{-4} N^{1.0667} \theta^{1.35} \frac{N^{1/12}}{M^{1/12} \sqrt{2}}.
\]  

(5.4)

In general, \( n_{\text{int}} \propto \theta^{-1.35} \) although this estimate may not be accurate if \( \theta \approx 0.2 \) or
\( \theta \approx 0.75 \). In these regimes, the number of interactions depends more strongly on other
tree integration settings, such as the maximum number of particles allowed to share
interaction lists.

We calculate the time spent on PM integration \((t_{\text{pm}})\) by multiplying the number of
mesh cells \((M)\) with the time required to do PM integration for one mesh cell \((\tau_{\text{pm}})\).
The total time spent on PM integration then becomes

\[
t_{\text{pm}} = \tau_{\text{pm}} M.
\]  

(5.5)

\(^1\)This value is based on timings from single site runs using \( N = 128^3 \) up to \( N = 2048^3 \)
We decompose the communication time ($t_{\text{comm}}$) into time spent to initially traverse the networks ($t_1$), which is latency-bound and time spent on data throughput ($t_b$), which is limited by the available bandwidth of the local network. Therefore,

$$t_{\text{comm}} = t_1 + t_b. \quad (5.6)$$

For each step, the code performs 18 collective operations containing $\log p$ communication steps and two all-to-all communications with $p$ communication steps. The time spent in latency ($t_1$) is calculated by multiplying the number of communication steps with the network round-trip time ($\lambda_{\text{lan}}$). As a result,

$$t_1 = \lambda_{\text{lan}} (18 \log p + 2p), \quad (5.7)$$

We determine the time spent on data throughput ($t_b$) by dividing the data volume of the local area communications by the network bandwidth ($\sigma_{\text{lan}}$). The communication data volume consists of three dominant parts. These are the grid of mesh cells (which consist of $4M$ bytes in total), the local essential tree structures (estimated to be $(48/\theta + 24) N^{2/3}$ bytes for $0.2 < \theta < 0.75$), and the sample particles which are used to determine the node boundaries ($12N r_{\text{samp}}$ bytes in total). Here, $r_{\text{samp}}$, which we set to $1/10000$ for large calculations, is the ratio of sampled particles relative to $N$. The time spent on data throughput is then

$$t_b = \frac{4M + (48/\theta + 24) N^{2/3} + 12N r_{\text{samp}}}{\sigma_{\text{lan}}}. \quad (5.8)$$

Additional communication is required to migrate particles between sites. The data volume of this communication is relatively large during initial simulation steps, but becomes negligible once sufficient steps have been taken to adequately balance the workload. A detailed review of the communication characteristics of the code is presented in [65].

### 5.3.2 Multiple supercomputers

We calculate the wall-clock time required for a single TreePM integration step using $p$ processes in total across $s$ supercomputers ($t_{\text{exec}}(s,p)$) by adding the wide area communication overhead $w_{\text{comm}}$ to the time spent on tree integrations ($t_{\text{tree}}$), the time spent on PM integration ($t_{\text{pm}}$) and the time spent on local area communications ($t_{\text{comm}}$). The execution time per step is therefore

$$t_{\text{exec}}(s,p) = t_{\text{tree}} + t_{\text{pm}} + t_{\text{comm}} + w_{\text{comm}}. \quad (5.9)$$

Here, we calculate $t_{\text{tree}}$ using Eq. 5.3, $t_{\text{pm}}$ using Eq. 5.5 and $t_{\text{comm}}$ using Eq. 5.6. The communication overhead on the wide area network ($w_{\text{comm}}$) consists of the time
spent in latency ($w_l$) and the time spent on data throughput ($w_b$) on the wide area network. As a result,

$$w_{\text{comm}} = w_l + w_b.$$  \hfill (5.10)

The code performs five blocking gather operations over all sites per step. These gather operations are performed using a ring scheme, which requires $s - 1$ neighbor exchanges per gather. We also require four blocking exchanges with each of the two neighboring sites. The total number of exchanges is then equal to $5s + 3$ and, the total time spent in latency ($w_l$) then becomes

$$w_l = \lambda_{\text{wan}} (5s + 3).$$  \hfill (5.11)

Here, $\lambda_{\text{wan}}$ is the network round-trip time between sites.

We calculate the time spent on wide area data throughput ($w_b$) by dividing the data volume of the wide area communications by the bandwidth capacity of the wide area network ($\sigma_{\text{wan}}$). The volume of the exchanged data between sites is similar to the data volume between nodes with two exceptions. First, the exchange of mesh densities requires one float per mesh cell per site and second, because SUSHI uses a 1D decomposition between sites in the $x$ direction we exchange and compare only the Cartesian $x$ coordinates of sampled particles. The data volume for the exchange of sampled particles is therefore three times smaller. The total time spent on wide area data throughput is

$$w_b = \frac{4sM + (48/\theta + 24) N^{2/3} + 4N_{\text{samp}}}{\sigma_{\text{wan}}}.  \hfill (5.12)$$

### 5.3.3 Scalability across sites

The speedup of a simulation across sites, $S(s)$, is defined by dividing the time required for an integration step on 1 site using $p$ processes ($t_{\text{exec}}(1,p)$) by the time required for an integration step over $s$ sites using a total of $s \cdot p$ processes ($t_{\text{exec}}(s,sp)$). It is therefore given by

$$S(s) = \frac{t_{\text{exec}}(1,p)}{t_{\text{exec}}(s,sp)}. \hfill (5.13)$$

The efficiency of a simulation across sites, $E(s)$, is calculated by dividing the time required for an integration step on 1 site using $p$ processes by the time required for an integration step over $s$ sites using a total of $p$ processes ($t_{\text{exec}}(s,p)$). The efficiency is then

$$E(s) = \frac{t_{\text{exec}}(1,p)}{t_{\text{exec}}(s,p)}. \hfill (5.14)$$
Table 5.1: Initial condition and accuracy parameters used for our test simulations. The maximum number of particles allowed to share an interactions list in the tree integration is given by ncrit.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matter density parameter ($\omega_0$)</td>
<td>0.3</td>
</tr>
<tr>
<td>Cosmological constant ($\lambda_0$)</td>
<td>0.7</td>
</tr>
<tr>
<td>Hubble constant ($H_0$)</td>
<td>70.0</td>
</tr>
<tr>
<td>Box size</td>
<td>$(30\text{Mpc})^3$</td>
</tr>
<tr>
<td>Mass fluctuation parameter ($\sigma_8$)</td>
<td>0.9</td>
</tr>
<tr>
<td>Softening for $N = 256^3/512^3/1024^3$ run</td>
<td>5/2.5/1.25 Kpc</td>
</tr>
<tr>
<td>Sampling rate $r_{samp}$ for $N = 256^3$</td>
<td>2500</td>
</tr>
<tr>
<td>Sampling rate $r_{samp}$ for $N &gt; 256^3$</td>
<td>10000</td>
</tr>
<tr>
<td>Tree opening angle ($\theta$), $z &gt; 10$</td>
<td>0.3</td>
</tr>
<tr>
<td>Tree opening angle ($\theta$), $z \leq 10$</td>
<td>0.5</td>
</tr>
<tr>
<td>Tree ncrit</td>
<td>1000</td>
</tr>
</tbody>
</table>

5.4 Experiments

We have tested SUSHI for performance on a grid of 5 Beowulf clusters, as well as an infrastructure consisting of four supercomputers. Each simulation lasts for 100 integration steps, and uses an opening angle of $\theta = 0.3$ when $z > 10$ and $\theta = 0.5$ when $z \leq 10$. For each opening angle we measured the total wall-clock time and communication time per step, averaged over 10 steps. All measurements were made near $z = 10$, approximately 460 million years after the Big Bang. A full listing of the simulation parameters and initial condition characteristics of our experiments is given in Tab. 5.1.

We compare the results of our experiments with predictions from our performance model. To do so, we measured the value of several machine constants using local tests and provide them in Tab. 5.2 and Tab. 5.5. The network constants used in the performance model are given in Tab. 5.6. As all simulations lasted for only 100 integration steps, particle exchanges were still performed to improve the distribution of work. We have added the measured average data volume of these exchanges to the data volume for wide area communications in our model. For full-length simulations, the data volume of these particle exchanges is a negligible part of the total communication volume.

5.4.1 DAS-3 experiment setup

The Distributed ASCI Supercomputer 3 (DAS-3 [24]) is a Dutch infrastructure that consists of 5 PC clusters within The Netherlands. The clusters use 10 Gbps networking internally, while the head nodes of each site are connected to regular internet in a star topology. Using end-to-end message passing tests we were able to achieve a performance of up to 1.25 Gbps between sites. The exact specifications of the five DAS-3 sites can
5.4 Experiments

Table 5.2: Technical specifications of five sites of the DAS-3 Dutch Grid. Three of the five DAS-3 sites reside in Amsterdam.

<table>
<thead>
<tr>
<th>Site</th>
<th>VU</th>
<th>UvA</th>
<th>LIACS</th>
<th>TU</th>
<th>MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
<td>A’dam</td>
<td>A’dam</td>
<td>Leiden</td>
<td>Delft</td>
<td>A’dam</td>
</tr>
<tr>
<td>AMD CPU model</td>
<td>DP280</td>
<td>DP275</td>
<td>DP252</td>
<td>DP250</td>
<td>DP250</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>85</td>
<td>41</td>
<td>32</td>
<td>68</td>
<td>46</td>
</tr>
<tr>
<td>Cores per node</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>CPU freq. [GHz]</td>
<td>2.4</td>
<td>2.2</td>
<td>2.6</td>
<td>2.4</td>
<td>2.4</td>
</tr>
<tr>
<td>Memory / core [GB]</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Peak [TFLOP/s]</td>
<td>3.26</td>
<td>1.44</td>
<td>0.66</td>
<td>1.31</td>
<td>0.88</td>
</tr>
<tr>
<td>(\tau)tree ([\times 10^{-9} \text{s}])</td>
<td>5.9</td>
<td>6.4</td>
<td>5.4</td>
<td>5.9</td>
<td>5.9</td>
</tr>
<tr>
<td>(\tau)mesh ([\times 10^{-7} \text{s}])</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>4.4</td>
<td>4.4</td>
</tr>
<tr>
<td>Ordering</td>
<td>1\text{st}</td>
<td>2\text{nd}</td>
<td>3\text{rd}</td>
<td>4\text{th}</td>
<td>5\text{th}</td>
</tr>
</tbody>
</table>

We performed experiments using three problem sizes and two opening angles. For our tests, we performed one set of runs with \(N = 256^3\) particles and \(M = 128^3\) mesh cells using 60 processes distributed evenly among the sites and two sets of runs with \(N = 512^3\) particles using 120 processes in total. One of the runs with \(N = 512^3\) uses \(M = 128^3\) and the other uses \(M = 256^3\). We maintained a fixed site ordering for all our runs as given in the bottom row of Tab. 5.2. For our experiments on the DAS-3, we have configured MPWide to use a single TCP stream per communication channel, send messages in chunks of 8 kB and receive messages in chunks of 32 kB. We did not use software-based packet pacing during these tests.

5.4.2 DAS-3 results

The timing measurements of our experiments can be found respectively in the left panel of Fig. 5.1 for the runs with \(N = 256^3\) and in Fig. 5.2 for the runs with \(N = 512^3\). Here we see that the communication overhead becomes marginally higher as we increase the number of sites in the simulation with \(N = 256^3\) and \(M = 128^3\) mesh cells, both for runs with \(\theta = 0\) and with \(\theta = 0.5\). The measured communication overhead for simulations with \(N = 512^3\) increases more steeply with \(s\), because the larger communication data volume results in a higher traffic load on the internet lines between the sites.

The calculation time per step is almost twice as high for runs using \(\theta = 0.3\) compared to runs with \(\theta = 0.5\), while the communication overhead is approximately equal. Integration steps with a smaller \(\theta\) therefore scale more favorably when increasing \(s\).

We provide a comparison between our timing results from the experiments and predictions from our performance model in Tab. 5.3 and Tab. 5.4. Here we see that the achieved performance roughly matches the model predictions. For the runs with \(N = 512^3\), the model provides slightly optimistic predictions for \(M = 128^3\) and slightly pessimistic predictions for \(M = 256^3\). This may be caused by the time required for
Figure 5.1: Wall-clock time (given by circles) and communication time (given by squares) per step averaged over 10 steps as a function of the number of sites $s$ for two different simulations. Results are given for a simulation with $N = 256^3$ and $M = 128^3$ over 60 processes using the DAS-3 (left panel) and across multiple supercomputers (right panel). The runs using $\theta = 0.5$ are given by open symbols and runs using $\theta = 0.3$ are given by solid symbols. The standard deviation of each measurement is shown by an error bar.
Figure 5.2: As Fig. 5.1, but for simulations on the DAS-3 using $N = 512^3$ and $M = 128^3$ (left panel), and simulations using $N = 512^3$ and $M = 256^3$ (right panel). All simulations were run over 120 processes in total.
the exchange of the local essential tree. In our model the size of the local essential tree is only dependent on $N$ and $\theta$. However, in our experiments we also find a minor correlation with the number of mesh cells used, as the range of the tree integration is equal to three mesh cell lengths.

### 5.4.3 Gravitational Billion Body Project experiment setup

We have run a number of test simulations across multiple supercomputers to measure the performance of our code and to test the validity of our performance model. The simulations, which use datasets consisting of $N = 256^3$, $N = 512^3$ and $N = 1024^3$ dark matter particles, were run across up to four supercomputers. We provide the technical characteristics of each supercomputer in Tab. 5.5. The three European supercomputers are connected to the DEISA shared network, which can be used without prior reservation although some user-space tuning is required to get acceptable performance. The fourth supercomputer resides in Japan and is connected to the other three machines with a 10 Gbps intercontinental light path.
5.4 Experiments

<table>
<thead>
<tr>
<th>$N^{1/3}$</th>
<th>$M^{1/3}$</th>
<th>np</th>
<th>sites</th>
<th>comm. measure</th>
<th>exec. measure</th>
<th>$t_{\text{comm}}$</th>
<th>$t_{\text{exec}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td># [s]</td>
<td>[s]</td>
<td>[s]</td>
<td>[s]</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>1</td>
<td>0.63</td>
<td>8.35</td>
<td>0.18</td>
<td>6.83</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>2</td>
<td>0.88</td>
<td>8.49</td>
<td>0.71</td>
<td>7.61</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>3</td>
<td>2.13</td>
<td>9.56</td>
<td>1.53</td>
<td>8.18</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>4</td>
<td>1.59</td>
<td>9.45</td>
<td>2.68</td>
<td>9.38</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>5</td>
<td>1.94</td>
<td>9.98</td>
<td>4.14</td>
<td>10.9</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>1</td>
<td>1.80</td>
<td>32.7</td>
<td>0.42</td>
<td>33.5</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>2</td>
<td>3.90</td>
<td>38.2</td>
<td>2.78</td>
<td>37.2</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>3</td>
<td>7.08</td>
<td>43.1</td>
<td>5.44</td>
<td>38.5</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>4</td>
<td>11.1</td>
<td>49.3</td>
<td>8.41</td>
<td>41.5</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>5</td>
<td>15.7</td>
<td>54.1</td>
<td>11.7</td>
<td>44.9</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>1</td>
<td>2.78</td>
<td>31.5</td>
<td>0.98</td>
<td>34.0</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>2</td>
<td>9.07</td>
<td>40.9</td>
<td>5.58</td>
<td>39.8</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>3</td>
<td>9.59</td>
<td>38.9</td>
<td>12.7</td>
<td>45.8</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>4</td>
<td>13.3</td>
<td>46.0</td>
<td>22.4</td>
<td>55.9</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>5</td>
<td>26.5</td>
<td>57.7</td>
<td>34.7</td>
<td>68.3</td>
</tr>
</tbody>
</table>

Table 5.4: As Tab. 5.3, but with $\theta = 0.5$ instead of $\theta = 0.3$.

5.4.3.1 Network configuration

In the shared DEISA network we applied the following settings to MPWide: First, all communication paths used at least 16 parallel streams and messages were sent and received in chunks of 256 kB per stream. These settings allow us to reach $\sim 100$ MB/s sustained throughput on the network between Amsterdam and Edinburgh. Second, we used software-based packet pacing to reduce the CPU usage of MPWide on the communication nodes. This had little impact on the communication performance of the application, but was required because some of the communication nodes were non-dedicated.

Although the light path between Amsterdam and Tokyo did not have an optimal TCP configuration, we were able to achieve a sustained throughput rate of $\sim 100$ MB/s by tuning our MPWide settings. To accomplish this throughput rate, we used 64 parallel TCP streams, limited our burst exchange rate to 100 MB/s per stream using packet pacing and performed send/receive operations in chunks of 8kB per stream. In comparison, when using a single TCP stream, our throughput was limited to 10 MB/s, even though the TCP buffering size was set to more than 30 MB on the end nodes. We believe that this limitation arises from TCP buffer limitations on one of the intermediary nodes on the light path.

Since most of the supercomputers are connected to the high speed network through specialized communication nodes, we are required to forward our messages through these nodes to exchange data between supercomputers. This forwarding is done in user space with MPWide Forwarder programs. A graphical overview of the network topology,
including the communication nodes as well as latency and bandwidth characteristics for each network path, can be found in Fig. 5.3.

### 5.4.4 GBBP results

The timing measurements of several simulations using \( N = 256^3 \) are given in the right panel of Fig. 5.1 and measurements of simulations using \( N = 512^3 \) are given in Fig. 5.4. We provide the timing results and model predictions for all our experiments in Tables 5.7 and 5.8.

We have performed a number of runs with \( N = 256^3 \) across up to three supercomputers. The runs over two supercomputers in the DEISA network have a communication overhead between 1.08 and 1.54 seconds. This constitutes between 8 and 13% of the

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>DAS-3 Value</th>
<th>GBBP Value</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda_{\text{lan}} )</td>
<td>LAN round-trip time</td>
<td>( 1.0 \times 10^{-4} )</td>
<td>( 8.0 \times 10^{-3} )</td>
<td>[s]</td>
</tr>
<tr>
<td>( \lambda_{\text{wan}} )</td>
<td>WAN round-trip time</td>
<td>( 3.0 \times 10^{-3} )</td>
<td>( 2.7 \times 10^{-1} )</td>
<td>[s]</td>
</tr>
<tr>
<td>( \sigma_{\text{lan}} )</td>
<td>LAN bandwidth</td>
<td>( 1.0 \times 10^{8} )</td>
<td>( 2.2 \times 10^{9} )</td>
<td>[bytes/s]</td>
</tr>
<tr>
<td>( \sigma_{\text{wan}} )</td>
<td>WAN bandwidth</td>
<td>( 5.0 \times 10^{7} / s - 1 )</td>
<td>( 5.0 \times 10^{7} )</td>
<td>[bytes/s]</td>
</tr>
</tbody>
</table>

Table 5.6: List of network parameters used for modelling the performance of our runs. The name of the constant is given in the first column, followed by a brief description of the constant in the second column, the value used for modelling the DAS-3 runs in the third column, the value used for modelling the GBBP runs in the fourth column and the unit used in the fifth column. Since the wide area network in the DAS-3 resembles a star topology, we divide the available bandwidth by \( s - 1 \).
5.4 Experiments

Figure 5.3: Network overview of a cosmological simulation across four supercomputers. The network latency and maximum sustained throughput for 64 MB message exchanges over each network path is given on the left, and an overview of the network topology is given on the right. Here, the communication nodes are indicated by the solid gray boxes.

Figure 5.4: As Fig. 5.2, but these simulations were run on the planet-wide network of supercomputers $s$. The wall-clock time is given as a function of the number of supercomputers. The ordering and technical specifications of the supercomputers we used are given in Tab. 5.5.
Figure 5.5: Snapshots of a simulation with $N = 512^3$ over four supercomputers. The snapshots are taken at $z = 5.65$, approximately one billion years after the Big Bang, when some dark matter structures can already be observed. The slices have been placed to match the volumes on the supercomputers that reside in Edinburgh (top left), Espoo (top right), Tokyo (bottom left) and Amsterdam (bottom right).
5.4 Experiments

The total runtime for runs with $\theta = 0.3$ and between 15 and 20% for runs with $\theta = 0.5$. The run with $N = 256^3$ between Edinburgh and Amsterdam has less communication overhead than the run between Espoo and Amsterdam, despite the 1 Gbps bandwidth limitation on the connection to Edinburgh. Our run between Amsterdam and Espoo suffered from a high background load, which caused the communication time to fluctuate over 10 steps with $\sigma \sim 0.25$ s, compared to $\sigma \sim 0.04$ s for the run between Amsterdam and Edinburgh. The run over three sites has a higher overhead than the runs over two sites, mainly due to using the direct connection between Edinburgh and Espoo, which is poorly optimized.

The wall-clock time of the simulations with $N = 512^3$ is generally dominated by calculations, although the communication overhead becomes higher as we increase $s$. The runs over two sites (Espoo and Amsterdam) spend about 4-5 seconds on communication, which is less than 10 percent of the total execution time for the run using $\theta = 0.3$. For simulations with $M = 128^3$, the use of three supercomputers rather than two does not significantly increase the communication overhead. However, when we run simulations with $N = 512^3$ and $M = 256^3$ the use of a third supercomputer doubles the communication overhead. This increase in overhead can be attributed to the larger mesh size, as the now larger data volume of the mesh exchange scales with $s$.

We provide an overview of the time spent per simulation step for the run with $N = 512^3$ and $M = 128^3$ over three sites (Edinburgh, Espoo and Amsterdam) in Fig. 5.6. For this run the total communication overhead of the simulation code remains limited and relatively stable, achieving slightly lower values at later steps where $\theta = 0.5$. A decomposition of the communication overhead for this run is given in the bottom panel of Fig. 5.6. The time required for exchanging the local essential tree constitutes about half of the total communication overhead. The simulation changes the opening angle at step 46, from $\theta = 0.3$ to $\theta = 0.5$. As a result, the size of the local essential tree becomes smaller and less time is spent on exchanging the local essential trees. The time spent on the other communications remains roughly constant throughout this run.

We have had a brief opportunity to run across all four supercomputers, which allowed us to run a simulation with $N = 512^3$. We provide a density plot of the particle data present at each site at the end of this run in Fig. 5.5. In addition, the timing measurements of the run across four sites are given in the left panel of Fig. 5.4. Here we observe an increase in communication time from 3.8-4.8 seconds for a run over the three DEISA sites to 12.9-19.3 seconds for the run over all four sites. However, the runs over three sites relay messages between Edinburgh and Espoo through the supercomputer in Amsterdam, whereas the four site run uses the direct connection between these sites, which is poorly optimized. If we use the direct connection for a run across three sites the communication overhead increases from 3.8-4.8 seconds to 10.1-13.2 seconds per step (see Tables 5.7 and 5.8). Therefore, the communication overhead of a run across four sites may be reduced by a factor of two if we would relay the messages between Espoo and Edinburgh over Amsterdam.

We also ran several simulations with $N = 1024^3$. The load balancing increases the communication volume of these larger runs considerably, as 850MB of particle data was exchanged on average during each step. However, the communication time
Figure 5.6: Performance results of a simulation with $N = 512^3$ and $M = 128^3$ using 120 processes in total across supercomputers in Amsterdam, Edinburgh and Espoo as a function of the simulation step number. The wall-clock time spent per step on calculations (top red area) and communications (bottom black area) is given in the top figure. Time spent on the four communication phases is given in the bottom figure. These phases are (from top to bottom) the migration of particles between sites, the exchanges of sample particles for determining the site boundaries, the local essential tree exchanges (PP) and the mesh cell exchanges (PM).
Table 5.7: List of runs performed for GBBP that use $\theta = 0.3$. The cube root of the number of particles and mesh cells are given in the first and second column, the number of processes and the supercomputers involved in the third and fourth column. Here, the letters correspond to supercomputers in Espoo (given by “H”), (E)dinburgh, (A)msterdam and (T)okyo. For runs over three and four sites, we relayed all traffic through the supercomputer in Amsterdam except for the runs marked with an asterisk, which used the direct network between Edinburgh and Espoo. The last four columns contain respectively the measured average communication time per step over 10 steps, the average wall-clock time per step over 10 steps, and the communication and wall-clock time as predicted by our performance model.

<table>
<thead>
<tr>
<th>$N^{1/3}$</th>
<th>$M^{1/3}$</th>
<th>np</th>
<th>sites</th>
<th>comm. measure</th>
<th>exec. measure</th>
<th>$t_{\text{comm}}$ model</th>
<th>$t_{\text{exec}}$ model</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>A</td>
<td>0.21</td>
<td>11.8</td>
<td>0.03</td>
<td>11.2</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>HA</td>
<td>1.09</td>
<td>12.2</td>
<td>1.05</td>
<td>12.2</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>EA</td>
<td>1.54</td>
<td>12.7</td>
<td>1.05</td>
<td>12.2</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>AT</td>
<td>3.74</td>
<td>15.5</td>
<td>4.22</td>
<td>15.4</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>HEA</td>
<td>5.96</td>
<td>17.1</td>
<td>1.66</td>
<td>12.8</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>A</td>
<td>1.05</td>
<td>71.0</td>
<td>0.05</td>
<td>59.4</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>HA</td>
<td>4.59</td>
<td>61.6</td>
<td>3.15</td>
<td>54.3</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>HEA</td>
<td>4.78</td>
<td>56.3</td>
<td>4.21</td>
<td>53.0</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>HEA*</td>
<td>13.2</td>
<td>71.0</td>
<td>4.21</td>
<td>53.0</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>HEAT*</td>
<td>19.3</td>
<td>70.1</td>
<td>9.43</td>
<td>57.8</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>A</td>
<td>1.37</td>
<td>46.0</td>
<td>0.08</td>
<td>52.6</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>HA</td>
<td>5.76</td>
<td>48.5</td>
<td>5.42</td>
<td>50.9</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>EA</td>
<td>7.09</td>
<td>46.4</td>
<td>5.42</td>
<td>50.9</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>AT</td>
<td>11.0</td>
<td>51.7</td>
<td>8.59</td>
<td>55.9</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>HEA</td>
<td>13.5</td>
<td>56.0</td>
<td>7.59</td>
<td>51.1</td>
</tr>
<tr>
<td>1024</td>
<td>256</td>
<td>240</td>
<td>E</td>
<td>3.22</td>
<td>190</td>
<td>0.16</td>
<td>204</td>
</tr>
<tr>
<td>1024</td>
<td>256</td>
<td>240</td>
<td>A</td>
<td>3.40</td>
<td>273</td>
<td>0.16</td>
<td>274</td>
</tr>
<tr>
<td>1024</td>
<td>256</td>
<td>240</td>
<td>EA</td>
<td>22.0</td>
<td>252</td>
<td>23.9</td>
<td>263</td>
</tr>
<tr>
<td>1024</td>
<td>256</td>
<td>240</td>
<td>HEA</td>
<td>31.3</td>
<td>295</td>
<td>26.0</td>
<td>246</td>
</tr>
</tbody>
</table>
High Performance Gravitational N-body Simulations on a Planet-wide Distributed Supercomputer

<table>
<thead>
<tr>
<th>$N^{1/3}$</th>
<th>$M^{1/3}$</th>
<th>np</th>
<th>sites</th>
<th>comm. measure</th>
<th>exec. measure</th>
<th>$t_{\text{comm}}$ model</th>
<th>$t_{\text{exec}}$ model</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>A</td>
<td>0.20</td>
<td>6.45</td>
<td>0.03</td>
<td>5.81</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>HA</td>
<td>1.04</td>
<td>7.09</td>
<td>0.97</td>
<td>6.75</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>EA</td>
<td>1.48</td>
<td>7.57</td>
<td>0.07</td>
<td>6.75</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>AT</td>
<td>3.58</td>
<td>9.86</td>
<td>4.14</td>
<td>9.93</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>60</td>
<td>HEA*</td>
<td>5.83</td>
<td>12.1</td>
<td>1.58</td>
<td>7.36</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>A</td>
<td>0.99</td>
<td>35.8</td>
<td>0.05</td>
<td>30.0</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>HA</td>
<td>4.52</td>
<td>32.8</td>
<td>2.82</td>
<td>28.7</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>HEA</td>
<td>3.80</td>
<td>29.0</td>
<td>3.88</td>
<td>28.5</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>HEA*</td>
<td>10.9</td>
<td>40.5</td>
<td>3.88</td>
<td>28.5</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
<td>120</td>
<td>HEAT*</td>
<td>12.9</td>
<td>39.7</td>
<td>9.23</td>
<td>33.7</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>A</td>
<td>1.31</td>
<td>26.8</td>
<td>0.07</td>
<td>27.9</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>HA</td>
<td>5.38</td>
<td>29.5</td>
<td>5.09</td>
<td>29.3</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>EA</td>
<td>6.99</td>
<td>29.4</td>
<td>5.09</td>
<td>29.5</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>AT</td>
<td>9.64</td>
<td>32.2</td>
<td>8.26</td>
<td>33.4</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>120</td>
<td>HEA</td>
<td>12.7</td>
<td>36.1</td>
<td>7.26</td>
<td>30.4</td>
</tr>
<tr>
<td>1024</td>
<td>256</td>
<td>240</td>
<td>EA</td>
<td>19.1</td>
<td>128</td>
<td>22.6</td>
<td>144</td>
</tr>
<tr>
<td>2048</td>
<td>256</td>
<td>750</td>
<td>AT</td>
<td>46.5</td>
<td>483</td>
<td>17.7</td>
<td>464</td>
</tr>
</tbody>
</table>

Table 5.8: As Tab. 5.7 but with $\theta = 0.5$ instead of $\theta = 0.3$. The run with $N = 2048^3$ used an older version of the code and different settings, and is described in Chapter 4.
still constitutes only one-tenth of the total run-time for the simulation across three supercomputers. We also performed one run with $N = 2048^3$ using an older version of the code and where $r_{\text{samp}}$ was set to 5000. The run used 500 processes in Amsterdam and 250 processes in Tokyo and is described in Chapter 4. We measured a communication overhead of $\sim 10\%$.

5.5 Scalability of $N$-body simulations across supercomputers

In this section we use our performance model to predict the scalability of $N$-body code across grid sites. In the first part of this section we examine the predicted speedup and efficiency when scaling three cosmological $N$-body problems across multiple supercomputers. In the second part we apply existing performance models for tree codes (with block and shared time step schemes) and direct-method codes to predict the scalability of three stellar $N$-body problems across supercomputers. In the third part we predict the efficiency of cosmological $N$-body simulations over 8 sites as a function of the available bandwidth between supercomputers. We provide an overview of the three cosmological problems in Tab. 5.9, and an overview of the three stellar problems in Tab. 5.11.

These problems are mapped to a global grid infrastructure, which has a network latency of 0.3 s and a bandwidth capacity of 400 MB/s between supercomputers. The machine constants are similar to the ones we used for our runs, and can be found in Tab. 5.10. To limit the complexity of our analysis, we assume an identical calculation speed for all cores on all sites. Our performance predictions use an opening angle $\theta = 0.5$.

5.5.1 Speedup and efficiency predictions for TreePM simulations

The predicted speedup $S(s)$ (as defined in Eq. 5.13) for three cosmological simulations as a function of the number of supercomputers $s$ can be found in Fig. 5.7. As the number of processes per site remains fixed, the total number of processes $p$ increases linearly with $s$. All example problems run efficiently over up to 3 sites. The simulation

\begin{tabular}{lllll}
\hline
Integrator & $N$ & $M$ & np 1 & np 2 \\
\hline
TreePM & $2048^3$ & $256^3$ & 128 & 2048 \\
TreePM & $2048^3$ & $1024^3$ & 128 & 2048 \\
TreePM & $8192^3$ & $1024^3$ & 4096 & 32768 \\
\hline
\end{tabular}

Table 5.9: Description of the cosmological $N$-body problems used for the scalability analysis across supercomputers. The type of integrator is given in the first column, the number of particles in the second column and the number of mesh cells in the third column. The last two columns contain respectively the number of processes per site for the speedup analysis and the total number of processes for the efficiency and bandwidth analysis.
Table 5.10: List of network parameters used for the scalability predictions of our code. The name of the constant can be found in the first column, the value used for our global grid model in the second column and the units used for each value in the third column.

<table>
<thead>
<tr>
<th>Name of constant</th>
<th>Value</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{\text{tree}}$</td>
<td>$5.0 \times 10^{-9}$</td>
<td>[s]</td>
</tr>
<tr>
<td>$\tau_{\text{pm}}$</td>
<td>$1.6 \times 10^{-7}$</td>
<td>[s]</td>
</tr>
<tr>
<td>$\lambda_{\text{lan}}$</td>
<td>$8.0 \times 10^{-5}$</td>
<td>[s]</td>
</tr>
<tr>
<td>$\lambda_{\text{wan}}$</td>
<td>$3.0 \times 10^{-1}$</td>
<td>[s]</td>
</tr>
<tr>
<td>$\sigma_{\text{lan}}$</td>
<td>$2.3 \times 10^{9}$</td>
<td>[bytes/s]</td>
</tr>
<tr>
<td>$\sigma_{\text{wan}}$</td>
<td>$4.0 \times 10^{8}$</td>
<td>[bytes/s]</td>
</tr>
</tbody>
</table>

with $N = 2048^3$ and $M = 256^3$ scales well as $s$ increases and obtains a speedup of $\sim 13$ when $s = 16$. When scaling up beyond $s \sim 25$, the speedup diminishes as the simulation becomes dominated by communication.

The simulation with $N = 2048^3$ and $M = 1024^3$ does not scale as well and only achieves a good speedup when run across a few sites. Here, the speedup curve as a function of $s$ begins to flatten at $s = 5$, due to the serial integration of the larger mesh. For $s \gtrsim 16$, the communication overhead begins to dominate performance and the speedup decreases for higher $s$. The speedup of the run with $N = 8192^3$ and $M = 1024^3$ scales better with $s$ than the speedup of the run with $N = 2048^3$ because it spends more time per step on tree force calculations.

We provide the predicted efficiency of three simulations over $s$ supercomputers relative to a simulation over one supercomputer, $E(s)$, using the same number of processes in Fig. 5.8. Here, the run with $N = 2048^3$ and $M = 256^3$ and the run with $N = 8192^3$ and $M = 1024^3$ retain an almost identical efficiency as we scale up with $s$. If $s \gtrsim 10$, the run with $N = 8192^3$ is slightly less efficient than the simulation with $N = 2048^3$ and $M = 256^3$. The simulation using $N = 2048^3$ and $M = 1024^3$ is less efficient than the other two runs. The data volume of the mesh cell exchange is 64 times higher than that of the run with $M = 256^3$, which results in an increased communication overhead.

Cosmological production simulations use a vast number of particles, and require compute resources that provide sufficient memory for particle storage and integration. The memory required for tree integration is equal to 60 bytes per particle and 52 bytes per tree node. As we use an $N_{\text{leaf}}$ value of 10 (where $N_{\text{leaf}}$ is the number of particles where the interaction tree will not be divided further), we have on average 0.75 tree nodes per particle, and therefore require a total of 99 bytes of memory per integrated particle (see [65] for further details).

Using 99 bytes per particle, a simulation with $N = 2048^3$ requires 850 GB of RAM for tree integration while a run with $N = 8192^3$ requires at least 54 TB of RAM. In addition, 4.5 bytes per mesh cell is required to do PM integration. These memory constraints place a lower limit on the number of processes that can be used, and indirectly determine the memory required on communication nodes. For particularly
5.5 Scalability of $N$-body simulations across supercomputers

Figure 5.7: Predicted speedup $S(s)$ of simulations using the TreePM method as a function of the number of sites $s$ in a global grid. The total number of processes scales linearly with $s$.

large exchange volumes, the code can conserve memory on the communication nodes by communicating in multiple steps.

5.5.2 Speedup and efficiency predictions for tree and direct-method simulations

We present predictions of three example $N$-body simulations of stellar systems, each of which uses a different integration method. The integration methods we use in our models are a Barnes-Hut tree algorithm with shared time steps, a tree algorithm using block time steps [90] and a direct-method algorithm [85] using block time steps. For the tree algorithm we choose the problem sizes and process counts previously used for the cosmological simulation models with $N = 2048^3$. Modelling the performance of direct-method simulations using $N = 2048^3$ is unrealistic, because such a run would take many years. We instead predict the performance of direct-method simulations using a more realistic problem size of 5 million particles.

We model the tree algorithm using a slightly extended version of the models presented in [84, 82] and the direct-method algorithm using the grid-enabled model presented in Chapter 2. An overview of the three problems used for our predictions is given in Tab. 5.11.

The simulations using the tree algorithm are mapped to the same global grid infrastructure that we used for modelling the cosmological simulation (see Tab. 5.10
Figure 5.8: Predicted efficiency $E(s)$ of simulations using the TreePM method as a function of the number of sites $s$ in a global grid. The total number of processes is kept fixed for all simulations.

Table 5.11: Description of the stellar $N$-body problems used for the scalability analysis across supercomputers. The type of integrator is given in the first column and the number of particles in the second column. The number of processes per site for respectively the speedup analysis and the efficiency analysis are given in the third and fourth column. The time step scheme used is given in the last column. Note that the direct-method simulation is modelled for a grid of GPUs.
Scalability of $N$-body simulations across supercomputers

for the machine constants used). The direct-method simulation is modelled using the network constants from the global grid model, but here we assume that each node has a Graphics Processing Unit (GPU) with a force calculation performance of 200 GFLOP/s. As each force interaction requires 66 FLOPs on a GPU, we spend $3.3 \times 10^{-10}$ s per force interaction (see [40] for details on direct $N$-body integration on the GPU).

5.5.2.1 Performance model for the tree algorithm

We extend the tree code models given in [84, 82] to include the grid communication overhead by adding the equations for wide area communication of the tree method from our SUSHI performance model. In accordance with our previous model, we define the wide area latency-bound time for the tree method ($w_{l,\text{tree}}$) as,

$$w_{l,\text{tree}} = \lambda_{\text{wan}} (4 (s - 1) + 4). \quad (5.15)$$

The bandwidth-bound communication time ($w_{b,\text{tree}}$) consists of the communication volume for the local essential tree exchange, which we estimate to be double the size used in TreePM simulations (see Section 5.3.1) due to the lack of PM integration, and the communication volume required for particle sampling. It is therefore given by,

$$w_{b,\text{tree}} = \frac{(96/\theta + 48) N^{2/3} + 4N r_{\text{samp}}}{\sigma_{\text{wan}}}, \quad (5.16)$$

where $r_{\text{samp}} = 1/10000$ and $\theta = 0.5$. We use the equation given in [82] for Plummer sphere data sets to calculate the total number of force interactions.

5.5.2.2 Modelling of block time steps

Tree and direct $N$-body integrators frequently use a block time step scheme instead of a shared time step scheme. Block-time step schemes reduce the computational load by integrating only a subset of particles during each step [54, 81]. To equalize the number of integrated particles between simulations, we therefore compare a single shared time-step integration with $N/n_b$ steps of a block time-step integration, where $n_b$ is the average block size. We adopt an average block size of $n_b = 0.2N^{0.81}$ for all block time-step integrators, the same value that we used in Chapter 2.

5.5.2.3 Predictions

We give the predicted speedup $S(s)$ of the three simulations as a function of the number of supercomputers $s$ in Fig. 5.9. Here, the tree code with shared time steps scales similarly to the cosmological simulation with $N = 2048^3$ and $M = 256^3$ (see Fig. 5.7). We predict a speedup of $\sim 13$ when $s = 16$. When we model the tree code with a block time step scheme, the scalability becomes worse because it requires $N/n_b$ times as many communication steps to integrate $N$ particles. The large number of communications combined with the high latency of wide area networks result in a high communication.
Figure 5.9: Predicted speedup $S(s)$ of $N$-body simulations using the tree and direct method as a function of the number of sites $s$ in a global grid. The total number of processes scales linearly with $s$. 
5.5 Scalability of $N$-body simulations across supercomputers

overhead. The direct $N$-body run on a global grid of GPUs also does not scale well over $s$, as predicted earlier in Chapter 2, due to the use of block time steps.

We give the predicted efficiency of the three simulations over $s$ supercomputers relative to a simulation over one supercomputer, $E(s)$ in Fig. 5.10. The efficiency of these runs is mainly limited by the latency-bound communication time. The tree code with shared time steps retains a high efficiency for $s \lesssim 16$, while the simulations with block time steps are less efficient due to the larger number of communications. We predict a slightly higher efficiency when $s \gtrsim 4$ for the direct-method simulation than for the tree code simulation with block time steps. This is because the lower $N$ results in a lower number of communication steps, and therefore in less communication overhead.

5.5.3 Bandwidth analysis for cosmological simulations

We have shown that our TreePM code scales well across up to $\sim 16$ sites if the ratio between the number of particles and the number of mesh cells is sufficiently large. Here we examine the efficiency of four cosmological simulations over 8 sites. The efficiency compared to a single-site run is predicted as a function of the available bandwidth. We have included three predictions for a global grid with 0.3 s network latency as well as one prediction for a simulation with $N = 2048^3$ and $M = 256^3$ over a medium-range grid with 30 ms network latency. The details of these simulations are given in Tab. 5.9,
and the results are shown in Fig. 5.11.

We find that the efficiency of cosmological simulations across sites is heavily dependent on the available bandwidth. The run with \( N = 2048^3 \) and \( M = 256^3 \) on a global grid has \( E(8) = 0.9 \) (where \( E(8) \) is the efficiency as defined in Eq. 5.14) when the supercomputers are connected with a 100 MB/s network. Using a network with a higher bandwidth has little effect on the achieved efficiency, as the communication overhead is then dominated by network latency. The effect of network latency is clearly visible when we look at the prediction for the same simulation on a grid with a shorter baseline. When using a grid with 30 ms network latency the simulation reaches \( E(8) = 0.97 \) if the wide area network achieves a throughput of 1000 MB/s (which is possible with a fine-tuned and dedicated 10 Gbps optical network). We predict an efficiency \( E(8) > 0.8 \) for both simulations if the available network bandwidth between sites is 50 MB/s.

The run with \( N = 2048^3 \) and \( M = 1024^3 \) is more communication intensive, and requires a network throughput of at least 700 MB/s to achieve an efficiency of 0.8. The simulation using \( N = 8192^3 \) and \( M = 1024^3 \) runs more efficiently than the simulation using \( N = 2048^3 \) and \( M = 1024^3 \) independent of the obtained bandwidth. Although the exchanged data volume is larger in the run with \( N = 8192^3 \), this increased overhead is offset by the higher force calculation time per step. The large communication volume reduces the efficiency considerably for low bandwidth values, and an average network throughput of at least 150 MB/s is required to achieve an efficiency of \( E(8) = 0.8 \).

5.6 Conclusion

We have run a few dozen cosmological \( N \)-body simulations and analyzed the scalability of our SUSHI integrator on a national distributed computer and across a global network of supercomputers. Our results confirm that SUSHI is able to efficiently perform simulations across supercomputers. We were able to run a simulation using 1024\(^3\) particles across three supercomputers with \( \sim 10\% \) communication overhead. The communication performance can be further improved by tuning the optical networks.

Based on our model predictions we conclude that a long-term cosmological simulation using 2048\(^3\) particles and 256\(^3\) mesh cells scales well over up to \( \sim 16 \) sites, given that sufficient bandwidth is available and the number of cores used per site is limited to \( \sim 256 \). We also predict that tree codes with a shared time step scheme run efficiently across multiple supercomputers, while tree codes with a block time step scheme do not.

Considerable effort is still required to obtain acceptable message passing performance through a long distance optical network. This is due to three reasons. First, it may take up to several months to arrange an intercontinental light path. Second, optical networks are generally used for high volume data streaming such as distributed visualization or bulk data transfer, and are therefore not yet tuned to achieve optimal message passing performance. Third, intercontinental networks traverse a large number of different institutes, making it politically difficult for users to diagnose and adjust settings on individual sections of the path. For our experiments we therefore chose to optimize the wide area communications by tuning our application, rather than requesting system-level
Figure 5.11: Predicted efficiency of four $N$-body simulations using the TreePM method over 8 sites ($E(8)$) as a function of the wide area network throughput ($\sigma_{wan}$) in MB/s. Three simulations are run on a global grid. One simulation uses $N = 2048^3$ particles and $M = 256^3$ mesh cells (given by the black dashed line), one uses $N = 2048^3$ and $M = 1024^3$ (black dotted line) and one run uses $N = 8192^3$ and $M = 1024^3$ (thick black line). A simulation using $N = 2048^3$ and $M = 256^3$ modelled for a grid with 30 ms network latency between sites is given by the thin gray line. Runs with $N = 2048^3$ particles are predicted using a total of 2048 processes, runs with $N = 8192^3$ particles are predicted using a total of 32768 processes.

modifications to the light path configuration.

The main challenges in running simulations across supercomputers are now political, rather than technical. During the GBBP project, we were able to overcome many of the political challenges in part due to good will of all organizations involved and in part through sheer patience and perseverance. However, orchestrating a reservation spanning across multiple supercomputers is a major political undertaking. The use of a meta-scheduler and reservation system for supercomputers and optical networks greatly reduces this overhead, and also improves the workload distribution between supercomputer centers. Once the political barriers are overcome, we will be able to run long lasting and large scale production simulations over a grid of supercomputers.
In this chapter we present the MPWide communication library, which we developed to facilitate high performance computing across multiple supercomputers.

6.1 Introduction

A parallel application can run concurrently on multiple supercomputers provided one is able to coordinate the tasks between them and limit the performance overhead of the wide area communications. The advantage of using a distributed infrastructure lies in the enormous amounts of storage, RAM and computing performance it makes available. Distributed computing therefore allows us to solve large scale scientific problems [60]. Starting from the coupling of Intel Paragons over an ATM network [112] in the early 1990s, distributed parallel applications have become very popular.

An efficient method to program a parallel application is the Message Passing Interface (MPI [118]), a language-independent communication protocol that coordinates the computing tasks in parallel programs. MPI is often used for intra-site parallelization, but it can also be used for message passing in a distributed infrastructure. In this case, processes exchange data with their local peers, as well as processes at other sites. Prior efforts in the use of MPI on distributed infrastructures are abundant [57, 89, 12] and several implementations have emerged which support execution across sites [37, 70]. With respect to N-body simulations Gualandris et al. [46] have demonstrated that it is possible to use grid-enabled clusters of PCs connected via regular internet, grid middleware and MPICH-G2 [70]. However, the vast majority of MPI implementations require
all participating nodes to have public IP addresses, which is generally undesirable for supercomputer environments for security reasons. Furthermore these implementations do not have a built-in optimization to fully exploit dedicated network circuits, a central component in multi-supercomputer infrastructures.

The lack of flexibility in deployment and link-specific optimizations of grid-oriented MPI implementations in distributed supercomputer environments led us to develop MPWide, a light-weight socket library specially aimed for wide-area message passing between supercomputers. In this paper we present several performance results and apply MPWide to parallelize a large-scale cosmological N-body simulation across two supercomputers.

6.2 Related work

Several grid message-passing libraries and frameworks have been developed with the intent to make distributed computing possible between sites that have restrictive firewall policies. PACX-MPI [38] is specifically geared for parallelization across sites and does not require compute nodes to have a public IP address. Instead, it forwards inter-site communications through two forwarding demon processes on each site. Such a setup works reasonably well for applications that have been parallelized over multiple supercomputers using regular internet [121], but the restriction to two communication processes is less optimal when using multiple sites in a dedicated network environment. The Interoperable MPI (IMPI) [41] standard has also been designed to specifically facilitate execution across sites, but at the time of writing very few of the vendor-tuned implementations on supercomputers support IMPI. Also, IMPI requires the installation of a centralized and globally accessible server and does not support path-specific optimizations.

NetIbis [11] and PadicoTM [26] are two communication frameworks which are able to establish connections using bootstrap links, thus not requiring public IP addresses. However, PadicoTM also requires the use of a centralized rendez-vous node for bootstrapping, and thereby some means of centralized connectivity. Both Ibis [127] and NetIbis are sufficiently flexible to use in a restricted supercomputer environment, but introduce a communication overhead compared to regular socket communications. These libraries are therefore less suitable for high-performance message passing over dedicated inter-supercomputer networks.

6.3 Architecture of MPWide

6.3.1 Design

MPWide is a light-weight communication library which connects multiple applications on different supercomputers, each of them running with the locally recommended MPI implementation. It can be installed by a local user without administrative privileges, has a very limited set of software requirements, and the application programmer interface is
6.3 Architecture of MPWide

similar to that of MPI. The applications are deployed separately for each supercomputer, and use MPWide to connect with each other upon startup. We are considering to add support for automated deployment, but to accomplish this we require a method to initiate applications on remote sites. The development of such a mechanism is not straightforward, because the access and security policies tend to be different for each supercomputer.

MPWide has been designed to facilitate message passing between supercomputers and construct/modify custom communication topologies. The MPWide library is linked to the application at compile time and requires only the presence of UNIX sockets and a C++ compiler. MPWide provides an abstraction layer on top of regular sockets with methods to construct a communication topology, to adjust the parameters of individual communication paths and to perform message passing and forwarding across the topology. MPWide does not link against local MPI implementations, but can be used to combine multiple programs parallelized with MPI. Maintaining separate implementations for intra- and inter-site message passing makes it easier to specifically optimize and debug long-distance communication paths while relying on well-tested and vendor-tuned software for optimal intra-site communication performance.

6.3.1.1 Data transport in the wide area network

Dedicated network circuits are excellently suited for facilitating data transport between supercomputers. The highly deterministic bandwidths of optical circuits (or lightpaths) reduce the communication time while properly tuned transport layer protocols increase the application throughput in the absence of competing traffic.

During the development of MPWide, we have examined the communication performance of several protocols by transferring data between two nodes in the Netherlands using a 10 Gbps optical network that was looped via Chicago, USA. We ran tests using both the TCP and the UDP network protocols. Plain UDP does not ensure the integrity of data packets however, and is therefore unsuitable for message passing. As an alternative, we instead tested the performance of two modified UDP implementations which feature mechanisms to ensure data integrity. These are Reliable Blast UDP (RBUDP [52], which is part of the Quanta toolkit [51]) and the UDP-based Data Transfer protocol (UDT [44]). The tests using TCP were run with both a single communication stream and with multiple streams in parallel.

We achieved a network throughput of less than 1 Gbps using RBUDP or UDT, and a throughput of up to 6 Gbps using parallel TCP streams. A full technical report on these preliminary performance tests can be found in [21]. Based on these results we decided to rely on multiple streams with a TCP-based protocol. This is a well-known and proven techniques to improve network performance in the WAN [114].

6.3.1.2 Functionality and programming interface

In MPWide, the communication takes place through channels. Each channel makes use of a single socket and provides a bidirectional connection between two ports on two hosts.
On network paths where the use of parallel TCP streams provides a performance benefit, it is possible to use multiple channels concurrently on the same path. The message passing and forwarding functions in MPWide are designed to operate concurrently on multiple channels when needed.

Channels are locally defined at initialization and may be closed, modified and reopened at any time during execution. This allows us to alter the communication topology at run-time, for example to restart or migrate part of the MPWide-enabled application.

Once one or more communication channels have been established, the user can transfer data using the communication calls in the MPWide API. Table 6.1 provides an overview of the functionality provided by MPWide.

<table>
<thead>
<tr>
<th>command name</th>
<th>functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPW_BARRIER()</td>
<td>Synchronize between two ends of the network.</td>
</tr>
<tr>
<td>MPW_CYCLE()</td>
<td>Send buffer over one set of channels, receive from other.</td>
</tr>
<tr>
<td>MPW_DSENDRECV()</td>
<td>Send/receive buffers of unknown size using caching.</td>
</tr>
<tr>
<td>MPW_INIT()</td>
<td>Set up channels and initialize MPWide.</td>
</tr>
<tr>
<td>MPW_FINALIZE()</td>
<td>Close channels and delete MPWide buffers.</td>
</tr>
<tr>
<td>MPW_RECV()</td>
<td>Receive a single buffer (merging the incoming data).</td>
</tr>
<tr>
<td>MPW_RELAY()</td>
<td>Forward all traffic between two channels.</td>
</tr>
<tr>
<td>MPW_SEND()</td>
<td>Send a single buffer (split over the channels).</td>
</tr>
<tr>
<td>MPW_SENDRECV()</td>
<td>Send/receive a single buffer.</td>
</tr>
</tbody>
</table>

Table 6.1: List of MPWide function calls. In addition to this list, each function has a variant call with a prefix ‘P’ which operates on one send and/or recv buffer per channel.

Since message passing can be performed over multiple channels in parallel, it is possible to communicate with multiple hosts simultaneously. For example, the user can scatter data across multiple processes with a single MPW_SEND() call or gather data from multiple hosts with a single MPW_RECV(). Each function has a variant call with a prefix ‘P’ (e.g., MPW_PSEND()) which takes an array of buffer pointers instead of one buffer pointer. These functions use one pointer for each channel, and the size of each separate buffer can be explicitly specified. Consequently, MPW_PSEND() or MPW_PRECV() functions can be used to respectively scatter and gather data which is not equally distributed across the hosts.

Both MPW_CYCLE() and MPW_DSENDRECV() also support the receiving of data buffers which are of unknown size (but not larger than a given size limit provided by the user). This feature may provide some performance improvement in long distance environments at the expense of possible excessive memory consumption, as separate calls to exchange data size information are no longer required.

An MPWide code example is shown in Fig.6.1. There we initialize MPWide with one single-stream path to a local network address, the LANChannels, as well as a double stream path to a different site, the WANChannels (lines 1-12). The program then initializes two buffers (line 14-15); it reads 100 bytes of data from the local connection.
6.3 Architecture of MPWide

int NumChannels = 3; // Total number of channels.
int NumLAN = 1; // Number of LAN channels.
int NumWAN = 2; // Number of WAN channels.
int MsgSize = 100;

int Hosts = {"10.0.0.100","123.45.67.89", "123.45.67.89");
int Ports = {6000, 6001, 6002};

MPW_Init(Hosts, Ports, NumChannels);

int LANChannels[NumLAN] = {1};
int WANChannels[NumWAN] = {2,3};
char* SendBuf = new char[MsgSize];
char* RecvBuf = new char[MsgSize];

// Recv from LAN.
MPW_Recv (SendBuf, MsgSize, LANChannels, 1);
// WAN exchange.
MPW_SendRecv(SendBuf, MsgSize, RecvBuf, MsgSize, WANChannels, 2);
// Send to LAN.
MPW_Send (RecvBuf, MsgSize, LANChannels, 1);
/* ( ... Process data and delete SendBuf and RecvBuf. ... ) */
MPW_Finalize();

Figure 6.1: Example code of the MPWide functionality

with MPW_Recv() (line 18); it exchanges this data with the remote WAN communication node using MPW_SendRecv() (line 20). At this point, the program has received the data from the remote WAN node, and forwards this data to the local connection (line 22).

6.3.2 Forwarder

When running an application across multiple sites, the processes on one site are not always directly able to communicate with the other site. In many cases this problem can be resolved by forwarding the messages to intra-cluster communication nodes, which do have access to all other sites through the wide area (dedicated) network. However, when the application uses a topology containing multiple dedicated networks, it will be necessary to forward messages from one network to another. The MPW_Relay() function provides such message forwarding for MPWide channels, and has been incorporated into the MPWide Forwarder. The Forwarder provides message forwarding for MPWide
in user space, connecting an MPWide channel from one network to that of another. It can therefore serve as relay process between nodes that are otherwise unable to contact each other or be put on intermediate nodes on very long network lines to mitigate the performance impact of packet loss. This latter method has been implemented and applied previously in the Phoebus project [7].

6.3.3 Implementation

We implemented MPWide using C++ in combination with GNU C sockets and POSIX threads [94]. MPWide creates and destroys threads on the fly whenever a communication call is made. With modern kernels, the overhead of creating and destroying threads is very small, and using MPWide we were able to reach nearly 10 Gigabit per second (Gbps) with message passing tests over local networks. For longer network paths, the high latency results in an even smaller relative overhead for thread creation/destruction. We have considered creating threads only at startup and managing them at runtime, but these modifications would increase the complexity of the code and only offer a limited performance benefit, as threading overhead plays a marginal role in wide area communication performance.

Aside from the ability to hardwire each communication, the library also supports a number of customizable parameters:

- number of concurrent streams for each communication call
- data feeding pace of sending and receiving.
- TCP window size for each individual socket

The maximum number of streams and the TCP window size may be restricted by local system policies. However, we were able to use up to 128 streams on most systems without requiring administrative rights. The code has been packaged and is publicly available at http://castle.strw.leidenuniv.nl/software/mpwide.html.

6.4 Benchmarking MPWide

We have run a series of tests to measure the performance of MPWide between two local supercomputer nodes, as well as two nodes connected by a 10 Gbps international network connection. We have chosen not to compare the performance with that of other message-passing libraries. These libraries often require system level optimizations by administrators, while MPWide can largely be optimized in user-space. A direct comparison will therefore be biased, depending on the amount of optimization done by system administrators.

For the local tests, we use two nodes of the Huygens supercomputer in Amsterdam, the Netherlands [64], where the nodes are connected by 8 parallel Infiniband links, each of which supports a maximum bandwidth of 20 Gbps. Our local tests use one
6.4 Benchmarking MPWide

out of these 8 Infiniband links. Each run consists of 100 two-way message exchanges, where we record the average throughput and the standard error. First we performed 8 different tests using messages of 8 MB and respectively 1, 2, 4, 8, 16, 32, 64 and 124 TCP streams in parallel. Due to system limitations, we were unable to perform tests using more than 124 streams on this particular site. We then repeated the same series of runs with message sizes of 64 and 512 MB.

The national tests were carried out between two sites of the Distributed ASCI Supercomputer 3 (DAS-3\(^1\)), one at the University of Amsterdam and one at the Delft University of Technology. Both sites are connected to regular internet with a 10 Gbps interface from the head node, and with a 1 Gbps interface from each compute node. A detailed specification can be found in columns 4 and 5 of Table 6.2. We performed the tests using the system default TCP window sizes (16 kB send and 85 kB recv).

For the international tests, we performed the same series of message exchanges, but now using one Huygens node and one node of the Louhi supercomputer in Helsinki, Finland [77], which are both connected to the DEISA shared network with a 10 Gbps interface. The round trip time of this network between Huygens and Louhi is 37.6 ms and we applied a TCP window size of 16 MB. The specifications of both supercomputers can be found in columns 2 and 3 of Table 6.2.

<table>
<thead>
<tr>
<th>Huygens</th>
<th>Louhi</th>
<th>DAS-3 Ams</th>
<th>DAS-3 Delft</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU vendor</td>
<td>IBM</td>
<td>Cray</td>
<td>AMD</td>
</tr>
<tr>
<td>Architecture</td>
<td>Power6</td>
<td>XT4</td>
<td>Opteron</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>104</td>
<td>1012</td>
<td>41</td>
</tr>
<tr>
<td>Cores per node</td>
<td>32</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>CPU frequency [GHz]</td>
<td>4.7</td>
<td>2.3</td>
<td>2.2</td>
</tr>
<tr>
<td>Memory per core [GB]</td>
<td>4/8</td>
<td>1/2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.2: Specifications of the Huygens and Louhi supercomputers, as well as the two sites of the DAS-3 Dutch Grid.

6.4.1 Results

6.4.1.1 Local tests

The results of the local performance tests, performed in March 2009, are found in Fig. 6.2. The local network line has a very low latency (< 0.1 ms) and is therefore quickly saturated when using multiple streams. In our results we found an increase in throughput when using 2 or 4 streams, but using more concurrent streams results in a performance decrease. When increasing the number of streams, the overhead caused by creating and destroying threads also increases, and may have contributed to this performance loss. However, if this were the case, we would observe a much steeper decline in performance for 8 MB messages than for 512 MB messages, as these communications take less time overall, and are thus more easily dominated by threading.

\(^1\)DAS-3: http://www.cs.vu.nl/das3/
overhead. We therefore conclude that this overhead is caused by saturation of the local network line. The maximum throughput achieved in these tests is close to the theoretical maximum bandwidth of 10 Gbps. This proves that the MPWide library can efficiently utilize the available bandwidth, if optimal settings are used.

![Graph](image)

Figure 6.2: Measured throughput in Gbps as a function of the number of communications streams used between two nodes on Huygens. The throughput is given for runs with 1 to 124 threads and message sizes of respectively 8, 64 and 512 MB.

6.4.1.2 National tests

We carried out the national tests over two sites of the DAS-3 Dutch Grid. One site resides at the University of Amsterdam and the other site is located at the Delft University of Technology. The round-trip time of the path between Amsterdam and Delft is 2.1 ms. The results of these tests are found in Fig. 6.3. Although the tests used the regular internet backbone, the fluctuations in our measurements are limited. When exchanging messages of 8 MB size, we obtain the best performance using a single stream, as the use of additional streams results in a lower and more variable performance. This is caused by the fact that message passing performance over multiple streams is limited by the slowest streams. For larger message sizes, however, using a single stream does not result in an optimal performance. Instead, we find that the best results are obtained using 8 streams (for 64 MB) to 32 streams (for 512 MB). Although a high peak performance is obtained when using 64 or more streams, the sustained performance is lower because the excess streams can cause network congestion. The round-trip time of 2.1 ms did not significantly reduce the achieved throughput in our tests.
6.5 Testing performance in a production environment

We originally developed MPWide to manage the long-distance message passing in the CosmoGrid project (see Chapter 4). CosmoGrid is a large-scale cosmological project which aims to perform a dark matter simulation of a cube with sides of 30 Mpc using supercomputers on two continents. In this simulation, we use the cosmological Λ Cold Dark Matter model \cite{48} which defines a constant fraction of the overall energy density for dark energy to model the accelerating expansion of the universe. We apply this
A Light-Weight Communication Library for Distributed Computing

Figure 6.4: Measured throughput in Gbps as a function of the number of communication streams used between Huygens and Louhi. The throughput is given for runs using 1 to 124 streams and message sizes of respectively 8, 64 and 512 MB.

model to simulate the dark matter particles with a parallel tree/particle-mesh N-body integrator, GreeM [65]. This integrator can be run either as a single MPI application, or as multiple MPI applications on different supercomputers. In the latter case, the wide area communications are performed using MPWide. We use GreeM to calculate the dynamical evolution of \(2048^3\) (~8.590 billion) particles over a period of time from redshift \(z = 65.35\) to \(z = 0\). More information about the parameters used and the scientific rationale can be found in [104].

Before the simulation is launched, the initial condition is decomposed in slices for each site, and in blocks within that slice for each process. Each block contains an equal number of particles but may vary in volume. A simulation process loads one block during startup, and calculates tree and particle mesh force interactions at every step. These force calculations require the exchange of particles with neighboring processes (and sites, see Fig.6.5) as well as the exchange of mesh cells. In addition, a number of smaller communications are performed to balance the load across all processes.

We have used GreeM together with MPWide in a set of test runs, which consist of full-lengths simulations of a limited scale (\(256^3\) particles). Also, we have performed a run across two supercomputers which consists of a limited part of the production simulation described earlier.

6.5.1 Test experiments

We have run three test simulations, of which each one uses a different infrastructure. All runs were carried out over two sites, with 30 calculation processes and one communication process per site. We performed one run on the DAS-3 testbed and one run across the Huygens and Louhi supercomputers. Both infrastructures are described in section
and for both infrastructures we carried out simulations with communication over 1 TCP stream, as the average data volume is only a few MB per communication.

For the third run we have used the Huygens supercomputer in combination with a Cray XT-4 supercomputer located at the Center for Computational Astrophysics in Tokyo, Japan. The Cray XT-4 consists of 740 nodes which run on a quad-core 2.2GHz AMD Opteron and have 8GB RAM each. To exchange data between the sites we reserved and used a 10 Gbps dedicated light path in the GLIF network [117], which has a round trip time of 273 milliseconds. This run was performed prior to the other two runs, using an older version of the code and the library. Unfortunately we were unable to reserve the lightpath for a new test run using our improved setup. For this test we used 64 concurrent TCP streams.

A detailed overview of the communication topology during the simulation can be found in Fig.6.6. Each of the supercomputers has been equipped with one specialized communication node. These nodes are each connected to the high-speed local supercomputer network and are linked together by the 10 Gbps light path. MPWide is used to transfer the locally gathered data to the communication node, forward it to the other site using the light path, and finally to deliver the data to the remote MPI simulation.

6.5.1.1 Results on DAS-3 Dutch grid

The performance results of our test simulation on the DAS-3 can be found in Fig. 6.7. Here we find that the simulation performance is dominated by calculation, with a communication overhead less than 20 percent of the overall wall-clock time throughout the run. As we used regular internet for the wide area communication, our simulation
Figure 6.6: Network topology example of the CosmoGrid simulation when run on two supercomputers, one in Amsterdam, the Netherlands and one in Tokyo, Japan. Data transfers within the local supercomputer are performed using MPI (thin arrows), whereas other communications are performed using MPWide (thick arrows). The communication nodes (indicated by the light gray boxes) reside outside of the MPI domains, and therefore use MPWide for all communications. Before the data is transferred to the communication node, it is gathered on a central process on the local supercomputer (indicated by the small gray boxes).

performance is subject to the influence of background network traffic. The two performance dips which can be found around step 1300 and 1350 are most likely caused by incidental increases in background traffic.

6.5.1.2 Results on Amsterdam and Helsinki supercomputers

The performance results of our test simulation between Amsterdam and Helsinki are shown in Fig. 6.8. The obtained performance is similar to that on the DAS-3, although two differences can be noted. First, the calculation time is \(\sim 25\) percent lower due to the superior performance of the supercomputer nodes. Second, although the average communication performance is similar to that observed on the DAS-3, we observe more variability in the communication performance. We are at this point uncertain about the exact nature of this variability. The DEISA network is shared with other institutions, so the presence of background traffic may have decreased our communication performance.

6.5.1.3 Results on Amsterdam and Tokyo supercomputers

The run between Huygens and the Tokyo Cray-XT4 was carried out in October 2008, before any of the other experiments in this paper, and served as a dress rehearsal for both the Tree-PM simulation code and MPWide. The simulated problem was of equal size to the previous simulations and uses the same number of processes. However, we performed the run using an older version of the code and different initial condition files. The performance results of this run can be found in Fig. 6.9. During this test run, the time spent on calculation is roughly constant throughout the run, with a peak
occurring during startup and a few points where snapshots are written. The time spent in communication is generally lower than the calculation time, taking about 7 to 10 seconds per step. However, we also observe a number of communication performance drops. These temporary decreases in performance were almost exclusively caused by single communications stalling for an extended period, which in turn were caused by periods of packet loss.

6.5.2 Production

We have executed a production-sized simulation between Amsterdam and Tokyo to measuring the performance of the code when it is used for production. Based on the results described in Sec. 6.5.1.3, we made a few changes to the network settings before performing the second run. We disabled the TCP memory suppression mode, increased the TCP window sizes, and increased the `sysctl` queue limit for upper-layer processing. Due to the limited length of our network reservation, we were not able to test the effects of modifying each of these settings in detail.

The production-sized run was performed on 752 cores in total. The topology of this run was asymmetric, using 500 cores on Huygens and 250 cores on the Cray for calculation. The full run lasted just under 12 hours, during which we performed 102 simulation steps. The performance results of this run can be found in Fig. 6.10. In this full-scale run, the calculation time dominated the overall performance, and was slightly higher at startup and during steps where snapshots were written. The communication performance is generally more constant than in the small-scale run between Amsterdam and Tokyo, with fewer and less severe performance drops and a slight increase in time after step 30 in the simulation. This increase may have been caused by the TCP buffering sizes, which the local system may change during run-time. Overall, the total communication time per step was between 50 and 60 seconds for most of the simulation, and constituted about one eighth of the total execution time.

6.6 Conclusions and future work

We present MPWide, a communication library to perform message passing between supercomputers. MPWide provides message passing that is intrinsically parallelized, and can be used for high-performance computing across multiple supercomputers. The library allows for customization of individual connections and has a light-weight design, which makes it well-suited for connecting different supercomputer platforms. We have shown results from local and wide area performance tests, and applied MPWide to combine two MPI applications into a very large parallel simulation across several wide area compute infrastructures. During our tests, we reached a sustained throughput of up to 4.64 Gbps over a long-distance 10 Gbps network. In addition, we were able to run an $N$-body simulation across two continents with $2048^3$ particles. During this simulation, about one eighth of the execution time was spent on communications.

Given that the parallel application is sufficiently scalable (which is the case for the $N$-body integrator used in this work), MPWide can be used to efficiently parallelize
Figure 6.7: Measured wall-clock time spent (in log-scale) on each simulation step for a 256³ particle test run on the DAS-3 between Amsterdam and Delft. The full-length run was performed using 62 cores, with 30 cores residing on each supercomputer and 2 cores used for communication only. The top dotted line indicates total time spent, the dashed line indicates time spent on calculation and the bottom solid line represents time spent on communication with MPWide.

Figure 6.8: Measured wall-clock time spent (in log-scale) on each simulation step for a 256³ particle test run on the DEISA network between Amsterdam and Helsinki. See Fig. 6.7 for an explanation of the lines.
Figure 6.9: Measured wall-clock time spent (in log-scale) on each simulation step for the $256^3$ particle test run. See Fig. 6.7 for an explanation of the lines.

Figure 6.10: Measured wall-clock time (in log-scale) for each simulation step for a partial $2048^3$ particle run. The run, which uses some adjusted TCP settings, was performed using 750 cores, with 500 cores used on Huygens and 250 cores used on the Tokyo Cray. An explanation of the lines can be found in the caption of Fig. 6.7.
production applications across multiple supercomputers. Future efforts to improve the usability of MPWide may include the integration with debugging tools and visualization toolkits, the introduction of group communicators and collective operations (similar to `MPI_COMM_WORLD` in MPI implementations), and the addition of automatic deployment mechanisms.
7 Conclusions

7.1 Thesis summary

In this thesis we examine the feasibility of running N-body simulations on globally distributed computers. We have run simulations of star clusters, galaxies and cosmological volumes on a globally distributed infrastructure and analyzed their performance in these wide area environments. Furthermore, we have developed time complexity models for direct and Tree/Particle-Mesh N-body simulations and applied them to predict the performance of N-body simulations across a large number of computational sites.

Our first experiments, which are described in Chapter 2, have been performed to determine whether direct-method N-body simulations can be efficiently run across a global grid. We present a time complexity model that can be applied to determine the execution time of direct N-body simulations on the grid. The model supports predictions for simulations that use either the copy or the ring communication scheme. We have run our direct-method simulations on a testbed consisting of three sites with special GRAPE hardware and measured the performance of our experiments. The overall run-time is dominated by the communication overhead in all our experiments. We find that the network response time is the primary performance bottleneck for runs over three sites that simulate fewer than $10^4$ particles. When running larger simulations, the communication overhead becomes more dependent on the throughput rate of the network, which is limited in these runs due to the use of regular internet. Based on the timing results of our experiments and predictions of our performance model, we conclude that direct-method simulations are not suitable for execution on a global grid of GRAPEs, but that a national grid of GRAPEs or GPUs can efficiently be used to run N-body simulations consisting of a few million particles.

In Chapter 3 we map N-body simulations to a computational grid by adopting a dynamic code that switches between N-body integration methods and computational sites at run-time. This switching mechanism allows the application to use the resources
Conclusions

best suited for each integration method it applies. We implement this concept as a living simulation that switches between direct-method \( N \)-body integration on GRAPE hardware in the United States and tree code \( N \)-body integration using GPU hardware in The Netherlands. The switching between sites is not initiated by an external manager, but rather done by the application at run-time. Here, the simulation obtains grid credentials, transfers files and submits a clone of itself to the new location. We have applied the living simulation to simulate a merger between two galaxies, using up to 65538 particles. The overhead of switching between the two integrators is less than 5 percent of the total execution time for most runs.

During our initial experiments we found that simulations run more efficiently in wide area environments if they contain more particles. However, a direct \( N \)-body simulation with more than a few million particles is computationally difficult because the execution time of direct-method integrations scales with \( O(N^2) \). Tree/Particle-Mesh (TreePM) simulations are used to model the structure formation of dark matter in the Universe over time and have a calculation complexity of \( O(N \log N) \). Due to the lower complexity, simulations with a large number of particles require less computing time when a TreePM method is applied. By using a TreePM code it becomes possible to simulate billions of particles on a single supercomputer.

In Chapter 4 we report on our experiences in running a TreePM simulation on a globally distributed supercomputer. We have combined the GreeM cosmological \( N \)-body integrator with a custom communication library to allow parallel simulations across two sites. We describe the setup and results of experiments that concurrently use an IBM Power6 machine in Amsterdam and a Cray-XT4 machine in Tokyo. Both sites were interconnected with a 10 Gbps optical network. Our simulations have achieved a calculation efficiency of \( \sim 90\% \) for production-sized problems and we predict that running the simulation across more than ten supercomputers would still provide satisfactory performance.

In Chapter 5 we analyze the performance of \( N \)-body integrators in general, and TreePM codes in particular, across multiple supercomputers. We have enhanced the cosmological code described in Chapter 4 to allow runs across any number of supercomputers. The code, named SUSHI, connects the sites in a ring topology and divides the simulation volume into one slice for each site. The workload on each site is adjusted at run-time to maintain an equal calculation time on all sites. We have tested SUSHI in parallel using up to 5 Beowulf clusters in a national grid, and up to 4 supercomputers in a global grid. Our simulations achieve an efficiency of \( 87\% \) across three supercomputers when using \( 1024^3 \) particles, and an efficiency of \( 73\% \) across four supercomputers when using \( 512^3 \) particles. We have developed a time complexity model for SUSHI and applied it to predict the performance of our code across a large number of supercomputers. Based on our predictions, we expect that the SUSHI scales well up to \( \sim 16 \) supercomputer sites for a simulation with \( 2048^3 \) particles and \( 256^3 \) mesh cells. To provide a comparison we also model the performance of tree and direct \( N \)-body codes across multiple supercomputers. We predict that these codes do not run efficiently across multiple sites when a block time step scheme is used. We conclude that using a widely distributed supercomputer to acquire more computing power is technically
feasible, and that the local scheduling and reservation policies are the primary obstacles for running long-lasting production simulations across supercomputers.

In Chapter 6 we present the MPWide communication library, which we have developed to perform message passing over long-distance optical networks. MPWide is a light-weight library that has few dependencies and can be quickly installed on different supercomputer platforms. It can be used to combine several local MPI applications into an application that runs across multiple sites. Connections established with MPWide can be customized individually without modifying the settings of the underlying system. We have tested MPWide on a local network, between two sites in a national grid, and between two supercomputers. The supercomputers were respectively located in Amsterdam (The Netherlands) and Helsinki (Finland). We achieved a sustained bandwidth of up to 4.8 Gbps on the 10 Gbps shared network between Amsterdam and Helsinki. In addition, we applied the library to run cosmological simulations across supercomputers.

7.2 Conclusions and Recommendations

In the introduction we asked whether it would be possible to map high performance \( N \)-body simulations to infrastructures that span up to several thousands of kilometers, and how we could make efficient use of such widely distributed resources. In this section we first review the performance characteristics of \( N \)-body simulations across a globally distributed computer, and then comment on the challenges faced when creating and using a planet-wide distributed supercomputer. We conclude with a discussion on future perspectives and applications of globally distributed (super)computing.

7.2.1 Simulations on a planet-wide distributed supercomputer

We have mapped star cluster, galaxy and cosmological simulations to a globally distributed infrastructure, combining computational resources from multiple sites to perform one simulation. By interconnecting computational sites to form a distributed system, we are able to run a simulation using multiple architectures and obtain more computing power than we could otherwise get from a single site.

One class of programs that efficiently uses global infrastructures consists of living \( N \)-body simulations, which dynamically combine multiple \( N \)-body solvers to tackle a complex \( N \)-body problem. Living simulations are used on heterogeneous grids, where each solver is best run on a different computational site. The program is then able to detect when a different solver is required to advance the simulation, dynamically switch to that solver and migrate to an architecture best suited to run it. A living simulation does not require the intervention of an external manager to switch between sites, and can be used to run hybridized simulations on grids without occupying any excess nodes. We have measured a switching overhead of only a few percent of the total runtime for our hybrid galaxy merger simulation, which has been presented in Chapter 3.

Another way to use the additional resources provided by a global computer is to run one simulation distributed over multiple sites. Such runs are able to use more
compute power, if the communication overhead introduced by wide area networks does not dominate the overall run-time. The overhead introduced by wide area networks is considerably larger than that introduced by local area networks, because the round-trip time of one message is much higher. For example, a light path between Amsterdam and Tokyo has a round-trip time of about 0.27 s, at least three orders of magnitude higher than that of a local supercomputer network. In addition, the bandwidth capacity of light paths is about one order of magnitude smaller than that of local supercomputer networks, and ranges between 1 and 10 Gbps at the time of writing. Due to the inferior performance characteristics of long distance networks, simulations run over a global grid are less efficient than if they were run on a local site, except in rare cases where local networks are prone to congestion [13].

The efficiency of $N$-body simulations run on a globally distributed machine can be improved in part by adjusting the parameters of the simulation. Many of the adjustments that reduce the relative communication overhead on a single site, also improve the efficiency of simulations run over a planet-wide distributed supercomputer. Examples of these adjustments are:

- Increasing the number of particles in the simulation.
- Decreasing the opening angle $\theta$ (if using tree integration).
- Distributing the particle load such that all processes spend an equal amount of time on calculations.

However, a simulation optimized for parallel execution on a single site does not necessarily run efficiently on a global grid. The increased response time of wide area networks cause each communication call to spend additional time. This especially reduces the efficiency of simulations that require a large number of communications with small data volumes. An example of such a simulation is the integration of a star cluster using a direct method (see Chapter 2). The round-trip time of wide area networks is constrained by the speed of light, and can only be reduced to some extent by improving the network. For example, a connection between Amsterdam and Tokyo, which are 9300 kilometers apart, will always have a round-trip time of at least 0.062 s.

The lower bandwidth capacity of wide area networks mainly increases the communication overhead of simulations that exchange large buffers of data, and limits the scalability of such runs. However, the difference in bandwidth capacity between local and wide area networks is sufficiently small to allow efficient execution of production simulations across a global infrastructure consisting of multiple supercomputers (see Chapters 4-6). Also, the bandwidth of wide area networks can be increased by optimizing the network configuration, or using multiple network paths concurrently.

We have run cosmological simulations across up to 4 supercomputers, and predict that simulations can be performed efficiently across up to $\sim 16$ sites in a global grid if a wide area throughput rate of 400 MB/s is obtained.
7.2.2 Recommendations for planet-wide $N$-body simulations

In this work we have used a wide range of $N$-body applications, planet-wide computational infrastructures, network interconnections and middleware. The optimal choice of interconnections and middleware is crucial for the performance and deployability of a planet-wide simulation, and depends strongly on the chosen computational resources and $N$-body application. Here we provide recommendations for four $N$-body applications, three of which run in parallel across multiple sites. These are direct-method $N$-body integration, tree integration and tree/particle-mesh integration. The fourth application is a hybrid $N$-body simulation which runs on a single site at a given time, but migrates between sites and can switch to a different integration method at runtime (as described in Chapter 3). The $N$-body applications are linked to three types of planet-wide infrastructures, including a network of supercomputers, a network of Beowulf PC clusters and a network of sites equipped with specialized hardware. We provide a list of recommended network interconnections for these applications and resources in Table 7.1.

Supercomputers are best connected using lightpaths, as a connection to the internet provides sub-optimal performance and is considered undesirable by many supercomputer centers for security reasons. The use of a shared light path is sufficient in most cases, though the large amount of data exchanged in large TreePM simulations justifies the use of a dedicated path. Simulations run over a planet-wide network of Beowulf clusters or specialized hardware sites are more limited in size, and the security policies of these sites tends to be less strict. The communication performance of direct and hybrid simulations is often not bound by network bandwidth, but rather by network latency. Since the network latency of light paths is almost identical to that of regular internet connections, it is more convenient to use the publicly available internet.

We provide a list of recommended middleware for the aforementioned applications and resources in Table 7.2. We recommend MPWide or an other customized socket library when running simulations across sites connected by light paths, as these connections require manual tuning of each communication channel to achieve optimal performance. Since the throughput rate is of lesser importance for direct $N$-body simulations, the ease of deployment of a cross-site MPI implementation often outweighs the performance benefit provided by MPWide. Using a cross-site MPI implementation is also an option for Tree or TreePM simulations across a network of Beowulf clusters, if the MPI implementation has been sufficiently optimized for this network.

7.2.3 Creating a planet-wide distributed supercomputer

The creation of a globally distributed computer requires the connection of computational sites over long distances. Connecting these sites is straightforward when the sites are accessible from regular internet and the user has administrative rights on each site. Administrative rights are easily obtained for small clusters of PCs, GPUs or GRAPEs and the bandwidth of regular internet may be sufficient for distributed runs using these smaller sites. However, the overall efficiency of a distributed simulation improves
Table 7.1: List of recommended interconnections for four application and three types of computational resources. The recommended interconnections are given for direct $N$-body integration (second column), shared time-step tree $N$-body integration (third column), tree/particle-mesh $N$-body integration (fourth column) and hybrid $N$-body simulations which switch between multiple solvers (fifth column). The computational resources considered include a planet-wide network of supercomputers (second row), a network of Beowulf PC clusters (third row) and a network of clusters equipped with specialized hardware (e.g. GPUs or GRAPEs, fourth row). The recommended types of interconnection include regular internet connection (given by “internet”), shared light path (given by “shared path”) and dedicated light paths (given by “ded. path”).

Table 7.2: List of recommended middleware for four application and three types of computational resources. An explanation of the rows and columns is given in Tab. 7.1. The recommended types of middleware include cross-site MPI implementations (given by “cMPI”) such as MPICH-G2 [70] or OpenMPI [37], local-site vendor MPI combined with MPWide (given by “MPW”) and Living Application middleware (given by “LA”).

for larger problem sizes. To run distributed simulations more efficiently we therefore require sites with more computational capacity, such as supercomputers.

### 7.2.3.1 Configuration of a global supercomputer

Interconnecting larger computing sites to create a globally distributed supercomputer takes considerable effort. The communication performance of regular internet is insufficient to efficiently exchange large data volumes between sites, and optical paths with a capacity of at least 1 Gbps need to be arranged to connect the supercomputers.

A considerable part of the work performed for this thesis involved arranging and testing wide area networks. In the most extreme case it took over a year to arrange a connection between two supercomputers. This overhead for reserving temporary dedicated light paths over long distances does not necessarily decrease for later reservations of the same network. Arranging a shared and permanent optical network also takes a long time, but shared networks require little effort to use once established and
properly configured. A drawback of shared networking is the presence of background communication traffic, which causes the performance of data exchanges to become less predictable. Analyzing and predicting the communication performance of long-lasting runs can be made easier if the communications performed over shared optical paths by each user would be tracked by a public monitoring system.

Optimizing communication performance over long distance paths requires the cooperation of all organizations involved. For intercontinental paths that span across several organizations, configuring the network for optimal performance becomes hard to accomplish for administrators and virtually impossible for users. In part, the connection can be optimized in user space by using a communication library that allows for path-specific tuning (see Chapter 6). Centralizing the management of intercontinental paths reduces the need for user-space tuning, but may be difficult to accomplish politically.

7.2.3.2 Scheduling and running on a global supercomputer

A simulation that runs across multiple supercomputers not only requires the sites to be interconnected, but also requires its processes on all sites to be run simultaneously. If one site takes longer to start its part of the simulation than others, the application will waste CPU hours awaiting the availability of the missing site. As supercomputers are heavily used, they normally have a queue of jobs waiting to be executed and few or no idle nodes at a given time. This load is lower during nighttime and during the weekends, upon which test runs of limited scale become possible without prior reservation.

Long-lasting production runs are currently difficult to perform on a globally distributed supercomputer, as the scheduling and reservation policies are different for each site. For example, the Cray machine in Tokyo enforces a maximum runtime of 8 hours per job but allows users to chain jobs together in a way that follow-up jobs do not need to be requeued. In contrast, the Cray machine in Edinburgh enforces a maximum runtime of 12 hours, but when jobs are chained together on this machine the follow-up jobs are treated as newly submitted jobs and requeued accordingly. Any time spent in the queue by a job on one site causes jobs on other sites to wait, and therefore waste CPU hours. Scheduling long-lasting production runs across supercomputers is therefore quite inefficient unless all sites adopt a uniform scheduling policy that does not requeue follow-up jobs or a reservation system is put in place.

At the time of writing, many supercomputers do not yet have a system for advance reservation. This makes it difficult to reserve resources on multiple supercomputers for a distributed run. The most convenient way to reserve resources for distributed runs would be through the use of a system that centrally coordinates reservations on multiple sites (e.g., HARC [79], GridARS [125] or GRMS [73]). However, such a system may impose modifications on local reservation policies or require direct access to supercomputers, which can be a security risk. Alternatively, supercomputer centers can implement local reservation systems, allowing users to manually reserve nodes on each site for a given period. This increases the overhead for users to arrange a job across supercomputers compared to a centralized system, but eliminates the waste of CPU hours that is incurred when running across sites without reservations.
We have shown that a globally distributed supercomputer can be used to efficiently run large scale $N$-body simulations. The technology to run long-lasting production simulations across the planet is available, but some additional political and organizational effort is required to establish a planet-wide distributed supercomputer suitable for production runs.

7.2.4 Future perspectives and applications

Although the focus of this thesis is $N$-body simulations on planet-wide infrastructures, the results and conclusions presented in this work may also apply to other classes of applications. Using a globally distributed computer is especially attractive for research fields where new insights can be obtained by running parallel applications on an unprecedented scale (e.g., cosmology).

This thesis provides methods and examples to map applications to planet-wide infrastructures, and obtain a sustained boost in compute performance. The living simulation method presented in Chapter 3 and the MPWide communication library presented in Chapter 6 can be applied to allow the execution of other application classes on a planet-wide infrastructure. We have shown that $N$-body simulations can be run efficiently over planet-wide infrastructures, but mostly considered the ease-of-use, security and fault-tolerance aspects to be outside the scope of this work. These factors are of limited importance for the application performance, but are essential to consider when making a planet-wide distributed supercomputer suitable for production runs. The recent work performed in the fields of grid and cloud computing provides a good starting point to investigate these aspects of distributed supercomputer environments.

Using planet-wide infrastructures for high performance computing also introduces new parallelization challenges. The network topologies between supercomputers are sparse and have high response times. These factors should be taken into account when parallel algorithms are ported to a global computer environment. Good scalability can be achieved both within, and between supercomputer sites by adopting hierarchical parallelization schemes such as the one presented in Chapter 5. By developing hierarchical schemes for other application classes, and enabling them to run efficiently in a distributed environment, we will be able to execute a much wider range of scientific applications on powerful planet-wide distributed supercomputers.
De zwaartekracht is van fundamenteel belang voor de vorming van sterrenhopen, sterrenstelsels en het universum als geheel. Zij zorgt ervoor dat deeltjes elkaar aantrekken en samenkloppen tot complexe structuren. Door te begrijpen hoe de zwaartekracht werkt zijn we in staat om het verleden van ons Universum te ontrafelen en voorspellingen te doen over hoe deze er in de toekomst uit zal zien. Het voorspellen van de beweging van materie in het universum is een ingewikkelde taak, en wordt ook wel het $N$-body probleem genoemd. Het $N$-body probleem heeft een exacte wiskundige oplossing voor systemen met 2 deeltjes, maar niet voor systemen met 3 of meer deeltjes. De bewegingen van deze $N$-body systemen worden daarom op numerieke wijze benaderd door $N$-body simulaties.

Vrijwel alle grootschalige $N$-body simulaties worden uitgevoerd met behulp van een enkele parallelle computer, terwijl er duizenden parallelle computers in de wereld aanwezig zijn. Dit proefschrift bevat een uitgebreid onderzoek naar de haalbaarheid van het uitvoeren van $N$-body simulaties over een wereldwijd netwerk van parallelle computers. We hebben hiertoe computers in verschillende werelddelen aan elkaar gekoppeld om $N$-body simulaties van sterrenhopen, sterrenstelsels en donkere materie mogelijk te maken. Daarnaast hebben we een tijdcomplexiteitsmodel ontwikkeld van enkele methoden die gebruikt worden om $N$-body systemen te simuleren. Deze modellen passen we toe om te analyseren onder welke omstandigheden $N$-body systemen efficiënt geraadpleegd kunnen worden op een wereldwijd netwerk van computers.

Hoofdstuk 2 bevat een analyse van $N$-body experimenten over een wereldwijd netwerk van GRAPEs. De GRAPE (GRAvity PipE) is een hardware component die speciaal ontwikkeld is om $N$-body simulaties te versnellen. We hebben een gedistribueerd systeem opgezet van drie GRAPE sites in drie verschillende continents. Hierop hebben we een aantal simulaties van sterrenhopen uitgevoerd, en de prestaties van deze simulaties gemeten. Daarnaast hebben we een tijdcomplexiteitsmodel gemaakt van onze simulaties. De simulaties gebruikten de directe integratiemethode, waarbij de
krachtsuitwisselingen tussen alle deeltjes expliciet wordt uitgerekend. Uit onze resultaten blijkt dat de hoge responstijd van de intercontinentale netwerken een overheersende factor is in de uitvoertijd van simulaties met \( \lesssim 10^4 \) deeltjes. Ook de uitvoertijd van simulaties met meer dan 10\(^4\) deeltjes wordt gedomineerd door communicatie, maar bij deze experimenten is het vooral de beschikbare bandbreedte die van groot belang is. Doordat we de communicaties tussen de GRAPE sites over regulier internet uitgevoerd hebben, en niet over een optisch netwerk, was de bandbreedte tussen sites beperkt. Op basis van onze metingen en voorspellingen concluderen we dat een N-body simulatie met de directe methode niet geschikt is om efficiënt over een wereldwijd gedistribueerd systeem uit te voeren. Wel is het mogelijk om een directe N-body berekening efficiënt op een nationaal netwerk van GRAPEs (of grafische kaarten) uit te voeren, mits de simulatie tenminste een paar miljoen deeltjes bevat.

In Hoofdstuk 3 beschrijven we een alternatieve manier om N-body simulaties over meerdere sites in een wereldwijd grid uit te voeren. We presenteren de *living simulation*, een dynamisch simulatieprogramma dat in staat is om zelfstandig te kiezen tussen verschillende N-body simulatiecodes en zich zelfstandig kan verplaatsen tussen verschillende grid sites. Om dit te bewerkstelligen hebben we de simulatie mogelijkheden gegeven om tijdelijk gebruikersrechten te krijgen voor het grid. Deze rechten worden dan door het programma gebruikt om bestanden te verplaatsen, en zichzelf te "kloon" naar een andere site. De mogelijkheid om zelfstandig te wisselen tussen N-body codes en te wisselen tussen locaties maakt het mogelijk om elke N-body code uit te voeren op de site die daarvoor het meest geschikt is. We hebben het living simulation concept toegepast om een hybride simulatie van de samensmelting van twee sterrenstelsels uit te voeren. Deze simulatie maakt gebruik van een Barnes-Hut tree N-body code met beperkte precisie op het moment dat de sterrenstelsels ver van elkaar verwijderd zijn, en stapt over naar het gebruik van een directe N-body code met hogere precisie op het moment dat de sterrenstelsels dicht bij elkaar komen. Wanneer de tree code gebruikt wordt draait de simulatie op een GPU in Nederland, maar zodra de directe code toegepast wordt verplaatst de simulatie zich naar een GRAPE in de Verenigde Staten. We hebben enkele experimenten met deze code uitgevoerd, waarbij we maximaal 65538 deeltjes gebruikt hebben. Tijdens de meeste simulaties wisselde de simulatie enkele keren tussen de verschillende sites, en was de totale overhead die door de wisselingen veroorzaakt werd minder dan 5% van de totale uitvoertijd.

Op basis van onze eerste experimenten concluderen we dat N-body simulaties efficiënter uitgevoerd kunnen worden over een wereldwijd netwerk van computers naarmate deze simulaties meer deeltjes bevatten. De tijdscomplexiteit van directe N-body simulaties schaalt echter met het kwadraat van de hoeveelheid deeltjes, waardoor simulaties met meer dan een paar miljoen deeltjes onpraktisch veel rekentijd vergen. Een alternatieve manier om N-body systemen te modelleren is met behulp van een *Tree/Particle-Mesh* (TreePM) code. Simulaties met een TreePM code hebben een lagere nauwkeurigheid, maar schalen qua tijdscomplexiteit met \( O(N \log N) \), waarbij \( N \) gelijk is aan het aantal deeltjes in de simulatie. Door de lagere tijdscomplexiteit van TreePM is het mogelijk om enkele miljarden deeltjes te simuleren op een supercomputer. De TreePM methode wordt voornamelijk toegepast voor het simuleren van
grootschalige systemen met veel deeltjes, zoals kosmologische volumes bestaande uit meerdere sterrenstelsels.

In Hoofdstuk 4 doen we verslag van onze ervaringen bij het implementeren en uitvoeren van TreePM N-body simulaties over twee supercomputers. Om dit mogelijk te maken hebben we de GreeM simulatie code gekoppeld aan een communicatiebibliotheek. Deze bibliotheek hebben we specifiek ontwikkeld om te communiceren over lange afstand via optische netwerken. Onze experimenten zijn uitgevoerd met een IBM Power6 supercomputer in Amsterdam en een Cray-XT4 supercomputer in Tokyo, welke verbonden zijn met een 10 Gigabit/s optisch netwerk. We hebben een simulatie met $2048^3$ deeltjes uit kunnen voeren over beide supercomputers. Hierbij is $\sim 90\%$ van de totale simulatietijd besteed aan berekeningen en $\sim 10\%$ aan communicaties. Volgens onze voorspellingen is het mogelijk om de simulatie op efficiënte wijze over 10 of meer supercomputers uit te voeren.

Hoofdstuk 5 bevat een uitgebreide analyse van N-body simulaties die uitgevoerd worden over een wereldwijd netwerk van supercomputers. We hebben de kosmologische code, die beschreven is in Hoofdstuk 4, uitgebreid om simulaties over 3 of meer supercomputers mogelijk te maken. De code, genaamd SUSHI, simuleert een plak van het kosmologisch deeltjesvolume op elke supercomputer en communiceert tussen de verschillende machines door de machines te schakelen in een ringstructuur. Daarnaast is SUSHI in staat om tijdens de simulatie de werklast evenredig te verdelen zodat de berekeningskracht gelijk blijft. We hebben SUSHI getest op een nationaal netwerk van 5 computerclusters en op een wereldwijd netwerk van 4 supercomputers. Onze simulaties behalen een efficiëntie van $87\%$ voor een simulatie met $1024^3$ deeltjes over 3 supercomputers, en een efficiëntie van $73\%$ voor een simulatie met $512^3$ deeltjes over 4 supercomputers. We hebben een tijdscomplexiteitsmodel opgesteld voor SUSHI en dit toegepast om de uitvoertijd te voorspellen van onze simulaties over een groter aantal supercomputers. Aan de hand van onze voorspellingen concluderen we dat een TreePM simulatie met $2048^3$ deeltjes met een raster van $256^3$ mesh cellen efficiënt uitgevoerd kan worden over maximaal $\sim 16$ supercomputers. Ter vergelijking hebben we ook de tijdscomplexiteitsmodellen van simulaties met de tree en directe methode toegepast voor een netwerk van meerdere supercomputers. Uit deze voorspellingen blijkt dat de simulaties die een schema van tijdstappen in blokken toepassen niet efficiënt uitgevoerd kunnen worden over meerdere sites. We concluderen dat het technisch mogelijk is om meer rekenkracht te verkrijgen door een wereldwijd netwerk van supercomputers te gebruiken. Het op elkaar afstemmen van het beleid van individuele supercomputercentra, voor het reserveren en plannen van simulaties, vormt de grootste hindernis om langdurige productiesimulaties over meerdere supercomputers mogelijk te maken.

In hoofdstuk 6 presenteren we de MPWide communicatiebibliotheek. We hebben MPWide ontwikkeld om een efficiënte uitwisseling van berichten tussen supercomputers mogelijk te maken. MPWide is geoptimaliseerd voor gebruik met optische netwerken over lange afstanden. Het is een compact programma met weinig vereisten dat gemakkelijk geïnstalleerd kan worden op verschillende soorten supercomputers. Met MPWide is het mogelijk om meerdere parallelle MPI programma’s te combineren tot een groter programma dat over meerdere sites uitgevoerd kan worden. Daarnaast is het mogelijk
om vanuit de applicatie de instellingen van individuele verbindingen aan te passen. We hebben de bibliotheek getest over een lokaal netwerk, tussen twee sites in een nationaal netwerk en tussen twee supercomputers in een internationaal netwerk. De twee supercomputers staan respectievelijk in Amsterdam (Nederland) en Helsinki (Finland). We hebben een gemiddelde snelheid van 4.8 Gigabit/s behaald op het 10 Gigabit/s gedeelde netwerk tussen Amsterdam en Helsinki. Daarnaast hebben we de bibliotheek toegepast om kosmologische simulaties over meerdere supercomputers uit te voeren.

We concluderen dat het in veel gevallen mogelijk is om $N$-body simulaties uit te voeren op een wereldwijd netwerk van computers. De grootste beperkende factor voor de efficiëntie van gedistribueerde experimenten is de hoge responstijd van intercontinentale netwerken. Dit maakt een wereldwijd netwerk vooral geschikt voor simulaties met relatief Weinig communicatiestappen, waarin relatief grote data buffers uitgewisseld worden. De technologie is reeds aanwezig om langdurige productie simulaties over meerdere supercomputers uit te voeren. Er zullen echter aanvullende organisatorische en politieke inspanningen nodig zijn om een wereldwijd gedistribueerde supercomputer klaar te stomen voor productie.
When Simon offered me a PhD position in 2006, I immediately figured that I wanted to continue in science and yet involve myself with the refreshingly different topic of astronomy. Now, four years later, I look back to an unusual but also very fun and defining period of my life. Simon, thank you for making this period possible to begin with, for helping me throughout my PhD, and having time for discussions even when you got busier, busier and busier yourself. Your intricate mix of direct criticism, solid support, sharp advices and empathy has helped me greatly to evolve as a scientific researcher, and a more balanced person in general. I also thank my promotores and the members of the promotiecommissie for reviewing my thesis, and providing valuable feedback.

Having nice colleagues in the workplace has helped me quite a bit, and I am very grateful to all of them. Alessia, thanks for taking the time to introduce me to the world of N-body systems when the deadline banshee was still looming over your own PhD thesis. Evghenii, thanks for the countless discussions and extraordinary insights on topics ranging from low-level optimization of N-body codes to unconventional beverage choices during lunch time. Stefan, thanks for all the valuable discussions laced with a sauce of quality black humor, and helping to introduce football as a perfectly valid conversation topic at work. Steven, thank you very much for helping me to tackle that behemoth of a project called CosmoGrid and introducing me to the concept of massively parallel living. Jeroen, I’m grateful for all the technical support and social amusement you have given. Atakan, thanks for your wisdom on profound and obscure matters. Bernadetta, thanks for inviting me over for several different parties and introducing me to the concept "card game injury".

I also would like to thank all the other members in the computational astrophysics group in Leiden. Thank you all for keeping a brilliant atmosphere, even though the group was rapidly expanding to a scientific horde. I would like to give a special mention to Arjan, Atakan, Bernadetta, Inti, Marcell, Nathan, and Bernadetta, Steven and
Jeroen (again) as they have proof-read my thesis. Thanks to their help I was able to fish out more than a few textual and structural bloopers. I have met many nice colleagues outside of our group here in Leiden, and I am grateful for all the fun they brought during the later years of my PhD. Marcel, whether it was at the NAC with free drinks or in Utrecht with free money, I’ve had a great time all over the board, and I wish you all the best in the States. Chael and Jan-Pieter, thanks for livening up the lunches, especially during the time I was still an alien to Leiden. Anita, Ignas, Jeanette, Jeanne and Merijn, thanks for the fun times at the Soccom and the nice parties we organized. And everyone else at the Sterrewacht, thanks for all your kindness and support.

I also had a great time with the heaps of colleagues at the UvA. Alfredo Tirado Ramos supervised me way back when I was a Bachelor student, and we now share a long history dotted with nice events. Alfredo, thanks for all the good times and your academic, beer-related and unconventional wardrobe suggestions. Alfons, thanks for all the advice and support during (and even after) the days of our QosCosGrid lives. Peter, thanks for all the valuable scientific discussions and insights you have provided over the years. I owe my thanks to the many colleagues at the UvA Computer Science department. Alfonso, Anton, Breaundán, Cees, Dick, Elena, Erik Hitipeuw, Erik Lorentz, Gohar, Gökhan, Ilkay, Jaap, Jordi, Lilit, Max, Michael Scarpa, Paola, Rick, Roman, Rudolph, Robert, Syed, Thomas, Yves and all the others at SCS, thank you for making my stay at the UvA so enjoyable. I’m also grateful to the colleagues from ‘upstairs’, Alessandro, Arjan, Diego, Dominik, Hendrik, Jason, Kostas, Lucinda, Lianne, Maciej, Martin, Nathalie, Peter, Valeriu and everyone else at the API for the good times we’ve had during the NACs, borrels, football tournament and other events.

The many colleagues I met abroad have been largely responsible for making my trips both very educational and a lot of fun. Jun, thank you very much for your hospitality during my trips to Tokyo, and all the guidance on CosmoGrid. Keigo, thank you for all the technical help, edged humor and fun trips to Akihabara. Michiko and Masaki, thank you for your helpfulness and good luck in Leiden! Piet, thank you for your hospitality in Kyoto and the enlightening and highly diverse discussions. Steve, thank you for all your advice and the educational late-night conversations. Tomo, it has been great to work with you for nearly three years on CosmoGrid. Thank you for all the excellent times we’ve had on and off work. I also thank the other QosCosGrid colleagues for the help they provided and fun we’ve had at various meetings, especially Camille, Chris, David, Gabor, Laszlo, Mariusz, Martin, Thomas and Valentin. Camille, thanks for all the insightful conversations we’ve had. I also wish to thank Clayton Breland for initiating a deep scientific and psychological debate after I accidentally poured a full bottle of beer into his pelvic region.

There is more in life than research alone, and there are many people outside of science to whom I’d like to show my appreciation.

My friends have been especially valuable throughout these four years. aTijs, Dirk, eTijs, Marcel, Marco, Martijn and Oliver, thanks for all the godsmacking jam sessions, movie nights and roleplaying sessions over the past years (and Oliver, thank you for designing the cover of this thesis!). Very few distractions are more fun than drenching oneself in self-made music or bizarre fantasy settings. I also thank all of them as well
as Aurelie, Bas, Brenda, Dagmar, Karianne, Kat, Kelly, Paul and René for all the incredible parties, festivals, meetups and discussions (whether deep, insane or both) we’ve had. In addition I’d like to express my gratitude to the Hoofddorp Zondag 4, the football team which provided some great fun/distraction the later years of my PhD. And Sanne, thank you for all the support you’ve given during the years we’ve been together. I definitely cherish the good times and I wish you all the best.

En uiteraard wil ik mijn familie graag bedanken. Maaike, bedankt voor het altijd luisterende oor, de scherpe suggesties en de cruciale hulp die je gegeven hebt, ook op momenten dat het jou niet uit kwam. Taco, bedankt voor alle nuttige discussies en je verhelderende blik op alledaagse zaken. Dick, bedankt voor alle scherpe discussies, cruciale praktische hulp en je rust. Mam, bedankt dat je altijd met me meedenkt en achter me staat. Pap, bedankt dat je er altijd voor me bent.

This research is supported by the Netherlands organization for Scientific research (NWO) grant #639.073.803, #643.200.503 and #643.000.803, the European Commission grant for the QosCosGrid project (grant number: FP6-2005-IST-5 033883), the Stichting National Computerfaciliteiten (NCF project #SH-095-08), SURFNet with the GigaPort project, the International Information Science Foundation (IISF), NAOJ, the Netherlands Advanced School for Astronomy (NOVA) and the Leids Kerkhoven-Boschcha fonds (LKBF). We also thank the network facilities of SURFnet, DEISA, IEEAF, WIDE, JGN2plus (Project No. JGN2P-A20077), SINET3, Northwest Gigapop, the Global Lambda Integrated Facility (GLIF) GOLE of TransLight Cisco on National LambdaRail, TransLight, StarLight, NetherLight, T-LEX, Pacific and Atlantic Wave. We thank the DEISA Consortium (www.deisa.eu), co-funded through the EU FP6 project RI-031513 and the FP7 project RI-222919, for support within the DEISA Extreme Computing Initiative (GBBP project).

Finally, I am also grateful to Hans Blom, Maxine Brown, Andreas Burkert, Halden Cohn, Jonathan Cole, Tom Defanti, Jan Eveleth, Juha Fagerholm, David Groep, Katsuyuki Hasabe, Douglas Heggie, Kei Hiraki, Wouter Huisman, Igor Idziejczak, Mary Inaba, Akira Kato, Walter Lioen, Hashim Mohammed, Kees Neggers, Petri Nikunen, Masafumi Oe, Hanno Pet, Ronald van der Pol, Gavin Pringle, Joaching Stadel, Huub Stoffers, Yoshino Takeshi, Thomas Tam, Jin Tanaka, Peter Tavenier, Mark van de Sanden, Peter Tavenier, Alan Verlo, Joni Virtanen, Lucas Weteling and Seiichi Yamamoto for their contributions to this work.
List of Publications

Journal Papers


- D. Groen, S. Rieder, P. Grosso, C. de Laat and S. Portegies Zwart: A Light-Weight Communication Library for Distributed Computing, Computational Sci-

Journal Papers in Preparation


Peer-reviewed Conference Papers


• J. Gomes; M. David; J. Martins; L. Bernardo; A. Garca; M. Hardt; H. Kornmayer; J. Marco; R. Marco; D. Rodriguez; I. Diaz; D. Cano; J. Salt; S. Gonzalez; J. Sanchez; F. Fassi; V. Lara; P. Nyczyk; P. Lason; A. Ozieblo; P. Wolniewicz; M. Bluj; K. Nawrocki; A. Padee; W. Wislicki; C. Fernandez; J. Fontan; Y. Cotronis; E. Floros; G. Tsouloupas; W. Xing; M.D. Dikaiakos; J. Astalos; B. Coghlan; E. Heymann; M. Senar; C. Kanellopolous; A. Tirado Ramos and D. Groen: Experience with the International Testbed in the CrossGrid Project, in P.M.A. Sloot; A.G. Hoekstra; T. Priol; A. Reinefeld and M. Bubak, editors, Advances in Grid Computing EGC 2005, in series Lecture Notes in Computer Science, vol. 3470, pp. 98-110. Springer, Berlin, Heidelberg, February 14-16 2005.


Book Chapters


Other Publications


• J. Gomes; M. David; J. Martins; L. Bernardo; A. Garcia; M. Hardt; H. Kornmayer; J. Marco; R. Marco; D. Rodriguez; I. Diaz; D. Cano; J. Salt; S. Gonzalez; J. Sanchez; F. Fassi; V. Lara; P. Nyczyk; P. Lason; A. Ozieblo; P. Wolniewicz;


