Available online at www.sciencedirect.com

ScienceDirect

Computers & Graphics 32 (2008) 25–40

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Special Section: Parallel Graphics and Visualization

Parallel techniques for physically based simulation on multi-core processor architectures

Bernhard Thomaszewski[a,*], Simon Pabst[a], Wolfgang Blochinger[b]

[a] *WSI/GRIS, Universität Tübingen, Germany*
[b] *Symbolic Computation Group, Universität Tübingen, Germany*

Received 31 July 2007; received in revised form 10 November 2007; accepted 27 November 2007

Abstract

As multi-core processor systems become more and more widespread, the demand for efficient parallel algorithms also propagates into the field of computer graphics. This is especially true for physically based simulation, which is notorious for expensive numerical methods. In this work, we explore possibilities for accelerating physically based simulation algorithms on multi-core architectures. Two components of physically based simulation represent a great potential for bottlenecks in parallelisation: implicit time integration and collision handling. From the parallelisation point of view these two components are substantially different. Implicit time integration can be treated efficiently using static problem decomposition. The linear system arising in this context is solved using a data-parallel preconditioned conjugate gradient algorithm. The collision handling stage, however, requires a different approach, due to its dynamic structure. This stage is handled using multi-threaded programming with fully dynamic task decomposition. In particular, we propose a new task splitting approach based on a reasonable estimation of work, which analyses previous simulation steps. Altogether, the combination of different parallelisation techniques leads to a concise and yet versatile framework for highly efficient physical simulation.
© 2008 Elsevier Ltd. All rights reserved.

Keywords: Physically-based simulation; Parallel collision detection; Parallel conjugate gradients; Multi-core processors

1. Introduction

Physically based simulation is an important component of many applications in current research areas of computer graphics. The most prominent examples are fluid, soft body, and cloth simulation. All of these applications utilise computationally intensive methods and runtimes for realistic scenarios are often excessive. Obviously, increasing the realism of the simulation by using more accurate methods, like finite elements, further aggravates the problem. In this paper we investigate on parallel techniques for improving the performance of physically based simulation codes on multi-core architectures.

Generally, most of the computation time is spent on two stages, time integration and collision handling. In the following, we will therefore consider these two major bottlenecks, which are present in almost every physically based simulation. Although we focus on cloth simulation in this work, the techniques proposed herein transfer to many other applications, like e.g. thin-shell and three-dimensional soft body simulation.

1.1. Implicit time integration

Often, the physical model at the centre of a specific simulator gives rise to stiff differential equations with respect to time. For stability reasons implicit schemes are widely accepted as the method of choice for numerical time

*Corresponding author. Fax: +49 7071 295466.
E-mail address: b.thomaszewski@gris.uni-tuebingen.de (B. Thomaszewski).
URL: http://www.gris.uni-tuebingen.de/~thomasze/index.html (B. Thomaszewski).

integration (cf. [1]). Implicit schemes require the solution of a (non-)linear system of equations at each time step. As a result of the spatial discretisation, the matrix of this system is usually very sparse. There are essentially two alternatives for the numerical solution of the system. One is to use an iterative method such as the popular conjugate gradients (cg) algorithm [2]. Another is to use direct sparse solvers, which are usually based on fill-reducing reordering and factorisation. The cg-method is favoured in computer graphics as it offers much simpler user interaction, alleviates the integration of arbitrary boundary conditions and allows balancing accuracy against speed. We will therefore focus on the cg-method in this work.

1.2. Collision handling

Most practical applications for deformable objects include collision and contact situations. Maintaining an intersection-free state at every instant is of utmost importance in this context and involves the detection of proximities (collision detection) and the reaction necessary to prevent interpenetrations (collision response). In the remainder, we refer to these two components collectively as *collision handling*. We usually distinguish between external collisions (with other objects in the scene) and self-collisions. Both of these types require specifically tailored algorithms for an efficient treatment. Even with common acceleration structures (see Section 3.4) these algorithms are still computationally expensive. For complex scenarios with complicated self-collisions, the collision handling can easily make up more than half of the overall computation time. It is therefore a second bottleneck for the physical simulation and hence deserves special attention.

1.3. Overview and contributions

In our previous work towards distributed memory architecture we developed basic parallelisation strategies for the two components of physical simulation. For the time integration stage, which exhibits a very fine granularity, we proposed a static data-parallel approach. For the highly irregular collision handling stage we proposed a dynamic task-parallel approach. In [3] the reader will find more details on these design decisions. The purpose of this work is to design efficient implementations of these state-of-the-art parallel techniques for physical simulation on shared-memory based multi-processor systems. We focus on the computationally most expensive components, which are numerical time integration and collision handling.

Implicit time integration leads to the solution of linear systems, for which we propose a parallel preconditioned cg algorithm. We developed an efficient implementation of this method and provide a detailed explanation of preconditioner application and matrix-vector multiplication [4]. In contrast to this, our parallel numerics code for distributed memory architectures employs the message passing-based programming model provided by the PETSc toolkit [5]. Although this code could theoretically be ported to shared-memory platforms, our aim here is to explicitly take advantage of the multi-core architecture, which enables a considerably different programming approach. As an example, inter-task communication can be implemented far more efficiently in the shared-memory setting by simply sharing data structures among threads. Furthermore, our shared-memory numerics code is entirely based on OpenMP directives. Therefore, it is much easier to integrate into existing sequential simulator code. It would require time-intensive redesign when adapting the code to the *single program multiple data*-paradigm of distributed memory architectures.

The performance of our numerical algorithms can be further increased using single precision arithmetic. We discuss this aspect in detail and explain how to implement the required modifications. Additionally, we investigate the performance of the parallel numerics code when applied to large input data.

For parallel collision handling we discuss and evaluate a novel task decomposition scheme based on temporal coherence data. In particular, we take advantage of the tight coupling of multi-core processors to derive work estimates for tasks with very low overhead. Moreover, we show how the resulting highly dynamic task-parallel execution process can be efficiently mapped to shared-memory architectures by using lock-free synchronisation mechanisms for task management. We describe the implementation of this technique based on specific atomic processor instructions and experimentally assess the resulting performance gain.

Finally, we present extensive experimental studies for all of the presented methods on three recent multi-core systems, showing that our approach scales well on different platforms.

2. Related work

Parallel numerics: The parallel solution of large sparse linear systems is a well explored but still active field in high performance computing. Most of the work from this field focuses on problem sizes that are considerably larger than the ones dealt with in computer graphics. Therefore, standard techniques do not necessarily translate directly to our application area. In general, good overviews on parallel numerical algebra can be found in the textbook by Saad [6] and the report compiled by Demmel et al. [7]. Parallel implementation of sparse numerical kernels like the ones used in this work, has already been investigated by O'Hallaron [8]. Oliker et al. [9] explored node ordering strategies and programming paradigms for sparse matrix computations. However, they did not consider parallel preconditioning.

Parallel cloth simulation: Previous research on parallel cloth simulation addressed shared address-space [10–12] as well as message passing-based architectures [13–16]. Since