SPINET: A Parallel Computing Approach to Spine Simulations

PETER G. KROPF[1], EDGAR F. A. LEDERER[2], THOMAS STEFFEN[3], KARL GUGGISBERG[4], JEAN-GUY SCHNEIDER[4], AND PETER SCHWAB[4]

[1]*Department of Computer Science, Laval University, Quebec City, Canada*
[2]*Institute of Computer Science, University of Basel, Switzerland*
[3]*Orthopaedic Research Laboratory, Division of Orthopaedic Surgery, McGill University, Montreal, Canada*
[4]*Institute of Computer Science and Applied Mathematics, University of Berne, Switzerland*

ABSTRACT

Research in scientific programming enables us to realize more and more complex applications, and on the other hand, application-driven demands on computing methods and power are continuously growing. Therefore, interdisciplinary approaches become more widely used. The interdisciplinary SPINET project presented in this article applies modern scientific computing tools to biomechanical simulations: parallel computing and symbolic and modern functional programming. The target application is the human spine. Simulations of the spine help us to investigate and better understand the mechanisms of back pain and spinal injury. Two approaches have been used: the first uses the finite element method for high-performance simulations of static biomechanical models, and the second generates a simulation development tool for experimenting with different dynamic models. A finite element program for static analysis has been parallelized for the MUSIC machine. To solve the sparse system of linear equations, a conjugate gradient solver (iterative method) and a frontal solver (direct method) have been implemented. The preprocessor required for the frontal solver is written in the modern functional programming language SML, the solver itself in C, thus exploiting the characteristic advantages of both functional and imperative programming. The speedup analysis of both solvers show very satisfactory results for this irregular problem. A mixed symbolic–numeric environment for rigid body system simulations is presented. It automatically generates C code from a problem specification expressed by the Lagrange formalism using Maple.

1 INTRODUCTION

The interdisciplinary SPINET project applied the possibilities of parallel computing to biomechanical simulations of the human spine. These are used to study the mechanisms for spinal disease leading to low back pain and/or loss of function. Due to the complex nature of such simulations, the very different issues of the various disciplines have to be considered and coordinated. A major effort has been made for a work-up of the interdisciplinary background of the SPINET project. Researchers working in different fields were defining the project goals in a joint effort. Special care was taken to understand and apply the correct terminology used in the different fields, and researchers were work-

Received November 1994
Revised September 1995

Scientific Programming, Vol. 5, pp. 15–24 (1996)
CCC 1058-9244/96/010015-10

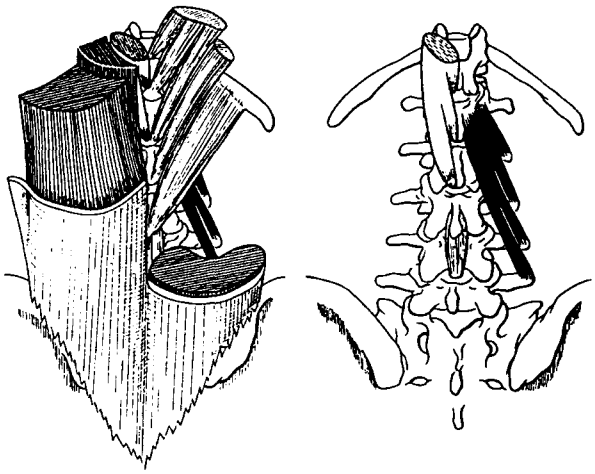

FIGURE 1 Lumbar spine with the large superficial muscles (left) and the small muscles of the deep layers (right). (From Kapandji [15], p. 26, with permission.)

ing in a close collaboration as the project was advancing.

It was the aim of the SPINET project to first apply the state-of-the-art possibilities of parallel computing to the mechanical simulation of the human spine, thereby evaluating parallel processing techniques. The principal target parallel architecture used throughout this project is the single-program multiple data (SPMD) programmable MUSIC [11], a distributed memory array of up to 63 digital signal processors.

In particular, the work presented here includes:

1. A parallel implementation of a finite element simulation. The target application is a production system for the calculation of the spine's mechanical behavior. It is well known, and could also be confirmed by our profiling, that the main time for a finite element simulation is needed (1) for the computation of the element stiffness matrices of the finite elements, and (2) for the solution of the system of linear equations resulting from the assembly of these element stiffness matrices to a global matrix. Thus, these two parts of the program are the main targets for parallelization. We have investigated two possibilities of organizing these computations: one is based on a conjugate gradient solver and the other on a frontal solver.
2. Realization of a mixed symbolic–numeric environment with automatic code generation for a rigid body system used to simulate mechanical behavior in a forward dynamic spine model. The target application is a development system for the interactive modeling of the spine's dynamic stabilization controlled by the various muscles and passive structures. The dynamics of the mechanical system is expressed by the Lagrange formalism using Maple [5]. Such a model description is subsequently transformed and combined with numeric algorithms to form the simulation system.

This article is organized as follows: Section 2 explains the target application and summarizes the relevant biomechanical and medical issues of the project. In Section 3 we present the finite element program under consideration and the results of two approaches for solving the associated linear systems of equations in parallel. Section 4 describes the symbolic–numeric environment used for automatic code generation of rigid body systems. We conclude with the final remarks in Section 5.

2 BIOMECHANICAL AND MEDICAL ISSUES

2.1 The Human Spine

The human spine is a complex structural unit of multiple functional elements with largely different material properties: rigid vertebrae, shock-absorbing intervertebral discs, controlling muscles, and stabilizing ligaments. It has to fulfill several tasks: mechanical support for loads, maintaining stability, permitting movements, and protecting the neural structures. Changes in this functional unit usually result in spinal dysfunction combined with low back pain and/or impairment.

A motion segment consists of two adjacent vertebrae and all interlining structures such as the intervertebral disc, posterior facet joints, muscles exerting active forces, and passive ligamentous structures limiting the range of motion. Twenty-five individual motion segments are concatenated to form the human spine. Movements of the trunk and the head are achieved through fine graded movements in the individual motion segments. Global spinal stability is maintained by large cross-sectional superficial muscles, whereas fine graded movements are coordinated by the shorter intersegmental muscles with a short lever arm of physical activity (Fig. 1).

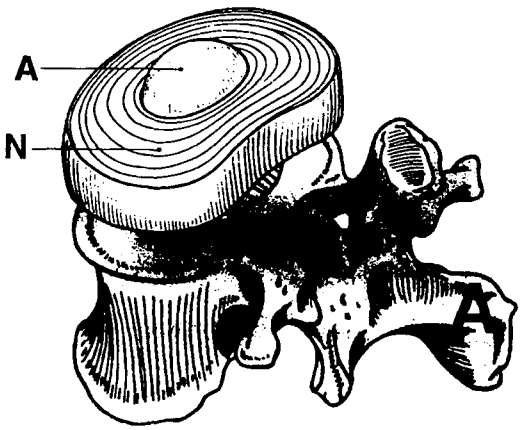

FIGURE 2 Vertebra and intervertebral disc. A, annulus fibrosus; N, nucleus pulposus. (From Kapandji [15], p. 91, with permission.)

2.2 Modeling the Spine

Computer modeling and simulation have become increasingly important in the field of bioengineering because mathematical modeling supports researchers in testing clinically relevant hypothesis [6, 9, 24]. Detailed simulations typically require enormous computer resources and can therefore be tackled only by the application of parallel computers. For instance, computation of the static load distribution in the spinal system for one single situation may take up to several hours on standard workstations when using a complex finite element model. However, bioengineers and physicians would like to perform dynamic simulations, therefore requiring much more computer performance.

To simulate spinal issues, integral models are needed including valid representations of the mechanical system, geometry, and the various tissue properties. The literature describes many approaches for the mechanical simulation of the spinal system (see [17] for an overview).

For the work presented in Section 3, a poroelastic finite element model was used. Figure 2 shows a lumbar vertebra with the intervertebral disc, and Figure 3 a corresponding finite element mesh used to model the mechanical properties of a motion segment. The model we used was developed by our biomechanical collaborators of the Warsaw University of Technology, Poland [6, 7], who had also implemented a corresponding simulation program. This original sequential implementation—subsequently called the Warsaw program—has served as a basis for the parallelization described in the next section. The main objective of this model is to examine changes in the intervertebral disc deformation, internal fluid pressure, and stresses over time when the disc is subjected to various loads. The disc tissue is simulated with porous materials composed of a solid porous skeleton and a fluid phase completely saturating the solid phase. The large deformations involved imply nonlinear analysis. Under axial load, fluid is redistributed within the intervertebral disc [4, 16, 23]. To understand the change in the disc's mechanical response over time, many simulations over long periods are required, which are very compute intensive.

A controlled rigid body model (Section 4) has been built to serve as a basis for investigations of dynamic phenomena. Muscles are represented by springs and dampers. The finite element model simulation is mainly related to models of the intervertebral disc, neglecting muscles and control activities. The simulation of rigid body systems is, in contrast, designed to consider the whole skeletal subsystem for possibly providing a plant on which the mechanisms of the muscle control subsystem and its behavior could be further studied.

2.3 Experimental Validation Procedures

Various experiments on human subjects and on cadaveric spine specimens have been conducted by the medical partners of the SPINET project. They serve to adjust input parameters and offer validation procedures for the predictions of the different models: stress–strain relations of cadaveric motion segments subjected to external loads [26], internal pressure distributions within the loaded cadaveric intervertebral disc [25], relative

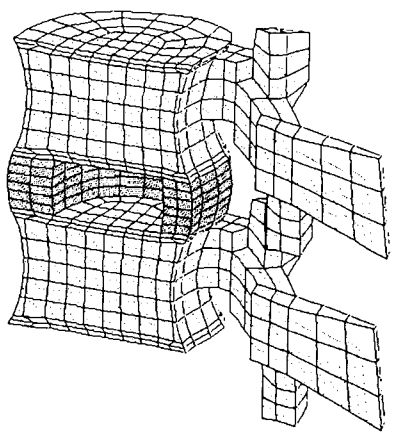

FIGURE 3 Finite element mesh of the geometry of a lumbar motion segment. (Provided by M. Matyjawski.)

displacement between adjacent vertebral bodies measured on human subjects [27], and muscle forces predicted from electromyographical readings [14].

3 STATIC SIMULATION

To enable the compute intensive simulations of the finite element model, we have developed a parallel version of the original sequential Warsaw program written in Fortran. This included a major reverse engineering to identify the most time-consuming parts by profiling and analyzing the algorithms, and to encapsulate those parts of the program which have to be parallelized or rewritten.

The following steps were needed to parallelize the Warsaw program: (1) Developing parallel solvers for the MUSIC, (2) porting the Fortran Warsaw program to the MUSIC, and (3) integrating the parallel solvers with the remaining Warsaw program. The porting of the Fortran program to the MUSIC platform was done using an `f2c` program generating C code from Fortran code; no Fortran compiler is available for the MUSIC. The following two sections describe these steps in more detail.

3.1 Parallel Finite Element Simulation

As outlined in the Introduction, we have investigated two approaches for the parallelization of the two most compute-intensive parts of a finite element simulation: (1) the computation of the element stiffness matrices and (2) the solution of the associated system of linear equations. The two approaches are based on a conjugate gradient solver and a frontal solver.

Conjugate Gradient Solver

All element stiffness matrices are computed and assembled to form one single large stiffness matrix. After this, the resulting system of linear equations is solved, either by a direct method such as a GAUSSian elimination or an iterative method such as a conjugate gradient method. In any case, the sparseness of the matrix is exploited by means of appropriate data structures. We have chosen a conjugate gradient method which is a standard solution method for finite element simulations [21, 29]. It utilizes the symmetry and positive definiteness of the whole matrix and usually uses less iterations than the number of equations.

Frontal Solver

For each finite element, the following two steps are carried out sequentially [13, 21]: (1) The element stiffness matrix of this element is computed and assembled with the (actual) so-called "frontal matrix" to form a new (actual) frontal matrix. (2) Those equations of the new frontal matrix which are independent of the stiffness matrices of all following finite elements [21] are eliminated. In this way, the computation of element stiffness matrices and the solution of the system of equations are performed in an interleaved way. The whole stiffness matrix will never be constructed. The frontal matrix grows by incorporating a new element, and shrinks by eliminating the corresponding equations. Usually the frontal matrix is much smaller than the complete matrix. A GAUSSian elimination method is used to eliminate the equations.

3.2 Parallel Conjugate Gradient Solver

The iterative conjugate gradient method is motivated by the potential convergence speedup of stationary iterative methods [1] in the case of symmetric and positive definite systems of linear equations [3, 20, 22].

When using the conjugate gradient method, a well-chosen preconditioning of the linear system of equations is crucial to achieve fewer iteration steps. However, as a first step, we were not concentrating on preconditioning methods, but rather on the parallel implementation itself.

The conjugate gradient solver implemented on the MUSIC system was realized by first giving a specification in terms of recursive definitions in Maple, then deriving a sequential code in C, and finally constructing a parallel program in C for the MUSIC.

Profiling of the sequential implementation identified the calculation of dot-products as one of the most time-consuming steps. Using a straightforward parallel implementation, numerical problems can occur: depending on the number of processors, the parentheses settings vary and therefore the sum in the dot-product is calculated differently. Because real arithmetic is not associative, the settings of parentheses affect the results, thus influencing the number of iterations needed for a given accuracy. Table 1 shows the resulting number of iterations needed to calculate a finite element model of a linear beam when using different floating-point precisions. As expected, the varia-

Table 1. Number of Iterations for a Linear Beam with 10 Elements*

Number of Processors	Linear Beam with 10 Elements Dimension = 324			
	32 Bit	44 Bit	64 Bit	128 Bit
1	194	123	112	102
3	381	123	113	102
5	167	123	111	102
10	163	124	113	102
15	298	123	112	102
20	171	123	112	102
25	183	123	112	103
30	237	124	112	102

*The results are obtained from the MUSIC system (32 and 44 bit [32 bit mantissa, 12 bit exponent; 9 decimal digits]), a SUN workstation (64 bit), and a DEC Alpha station (128 bit).

tions of the number of iterations due to this effect are much bigger for lower than for higher floating-point precisions.

The systems of linear equations of the finite element simulations of linear beams were used to investigate the speedup of the implementation on the MUSIC system. Because the number of iterations depends on the number of processors (refer to Table 1), the time consumed for one iteration step was used to determine the speedup. Due to memory restrictions of the MUSIC system (2.5 MByte per node), it was not possible to run all calculations on one processor only. Therefore, the speedup shown in Figure 4 is normalized to 1.0 for five processors, i.e., $S_5(P) = T(5)/T(P)$, in order to compare speedup values for systems of linear equations of different size.

For the finite element simulation of linear beams consisting of a homogeneous material, this solver yielded satisfactory results. As expected, the number of iterations is much smaller than the number of equations (e.g., 460 iterations for 2,500 equations). Unfortunately, the situation for finite element simulations of the human spine is quite different: The number of iterations is considerably higher than the number of equations. This is due to the very different material constants (YOUNG moduli) of the vertebrae and intervertebral discs; they differ by a factor of about 10^5. Therefore, the eigenvalues of the matrix differ strongly resulting in a large condition number of the matrix [12]. While simulating the same spine model with equal material constants for vertebrae and discs (which is not valid from a biomechanical point of view), the solver yielded again satisfactory results [19]. However, the investigation of better preconditioning methods for spine simulations might reduce the number of iterations.

Our results show that an increase of the floating-point precision decreases the number of iterations, more in badly than in well-conditioned equation systems, but does not necessarily lead to a better numerical behavior. Moreover, a floating-point precision of 44 bit requires less than 10% more iterations than 64 bit (refer to Table 1).

3.3 Parallel Frontal Solver

The original Warsaw program contains a frontal solver. The main reason for using a frontal solver in this program is the efficient usage of storage space on small machines—it has been originally developed for a PC with a maximum of 640 KBytes main memory. Its concrete implementation was taken from Beer and Haas [2], which has been especially designed for machines with a small main memory and no virtual memory support by the operating system. It simulates a virtual memory for frontal matrices: Even though frontal matrices are usually much smaller than the whole matrix, they are too big to fit into the main memory of a small computer. For parallelization, this implementation is not well suited because (1) the simulation of virtual memory requires file I/O which is inefficient; instead we wanted to keep all data in the distributed main memory, and (2) a sufficiently detailed documentation was missing so that paral-

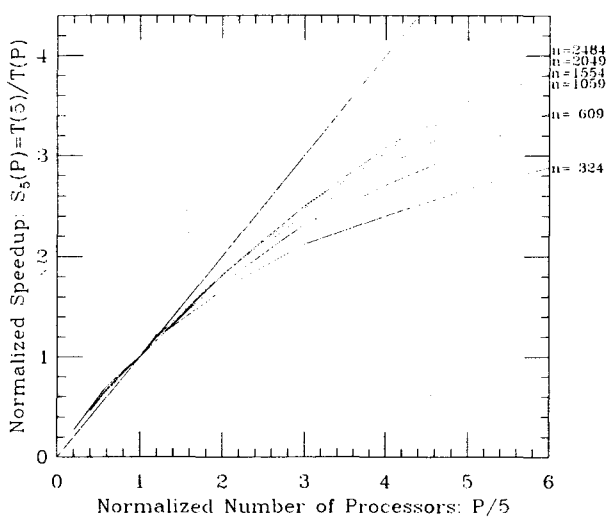

FIGURE 4 Speedup normalized to five processors ($S_5(P) = T(5)/T(P)$) of the conjugate gradient solver on the MUSIC system. n is the dimension of the corresponding system of linear equations.

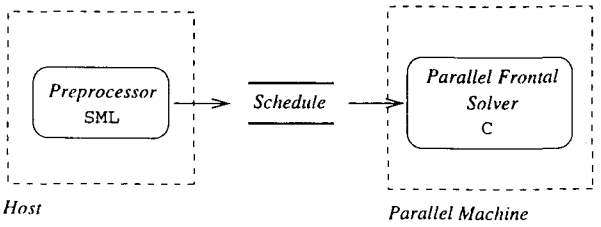

FIGURE 5 Schematic overview of the parallel frontal solver.

lelizing the program would have become very difficult. Therefore, the implementation of a new parallel frontal solver was evident. Moreover, using this method we did not change the numerical behavior of the original Warsaw program, because this program was already using a frontal solver.

The method mainly described in the literature for the parallelization of the frontal method is the multifrontal method [8, 28]. However, we have chosen a simpler scheme handling only one single frontal matrix. Because the frontal matrices of our application are quite large, we expected enough parallelism, which could be confirmed by the measurements presented below.

Parallelism in the frontal solver is exploited as follows: The stiffness matrices of the single finite elements are independent of each other; hence, their parallel computation is trivial. The solution of the system of linear equations is carried out by element-wise application of a GAUSSian elimination to the frontal matrix. Because for our finite element simulations the frontal matrices are quite large, the GAUSSian elimination can be efficiently carried out in parallel. This is realized with a row-wise distribution of the frontal matrix onto the processors. Each row of the frontal matrix corresponds to a row of the (never constructed) whole stiffness matrix. The rows of the frontal matrix are distributed such that this corresponds to a cyclic distribution of the rows of the whole stiffness matrix.

The backward substitution proceeds in a sequential, but distributed way: The entire matrix resulting from the frontal elimination is stored block-wise, and distributed over the whole machine; however, only one processor is active at a time.

From a software–technical point of view, the frontal solver program consists of two parts (Fig. 5): (1) A preprocessor, taking the incidence list of the finite elements as input and generating a schedule as output, and (2) the parallel solver itself, taking this schedule as input. The solver essentially consists of nested and sequentially composed for-loops. The ranges over which these for-loops iterate as well as the sparsity patterns are given in the schedule. During a complete finite element simulation, the parallel solver runs many times, the preprocessor only once. Therefore, the preprocessor is not time critical compared with the solver; however, it is more involved. This suggested the following design: The preprocessor is written in the modern functional programming language SML [18] and sequentially runs on the host, whereas the parallel solver is written in C and runs on the MUSIC. Using this approach, the characteristic advantages of imperative and modern functional programming languages are combined: easy programmability in the functional setting and high efficiency in the imperative setting.

We have measured the speedup of our frontal solver using three-dimensional finite element models of a spinal motion segment with 52, 104, and 208 finite elements. Each finite element has 20 nodes, with three degrees of freedom each. Therefore, each element stiffness matrix is of size 60 × 60. The dimensions of the resulting systems of linear equations are 1,020, 1,884, and 3,327, for the models with 52, 104, and 208 finite elements, respectively. All measurements were performed on a MUSIC system with 28 processors and performed for 1, 2, 3, 4, 6, 8, 10, 12, 16, 20, 24, and 28 processors using 44-bit floating-point precision. In the following, we study the influence of three aspects on the speedup: (1) the backward substitution, (2) the number of finite elements and the front width, and (3) the time for computing the element stiffness matrices.

Backward Substitution

Times were measured for forward (GAUSSian) elimination alone and for forward elimination and backward substitution together, for the model with 52 finite elements (Fig. 6). Because the backward substitution runs sequentially, it flattens the speedup curve.

Number of Finite Elements and Front Width

Times are measured for the models with 52, 104, and 208 finite elements (Fig. 7). Because the entire matrix required for the backward substitution can only be stored with a larger number of processors that provide enough storage, just the forward elimination is carried out and measured in order to determine the speedup. The speedup for 104 ele-

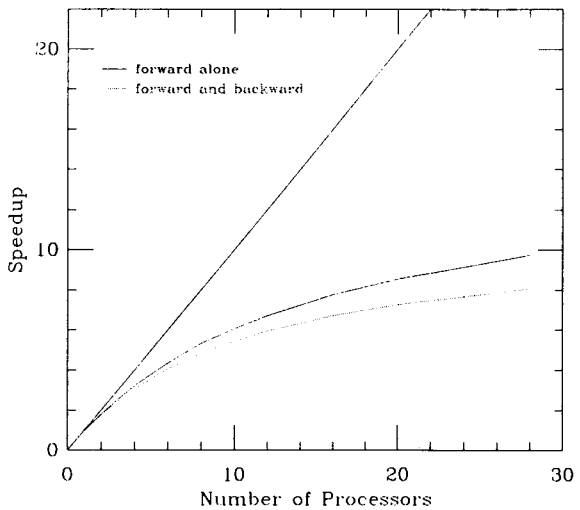

FIGURE 6 Speedup of the frontal solver on the MUSIC system: influence of the backward substitution.

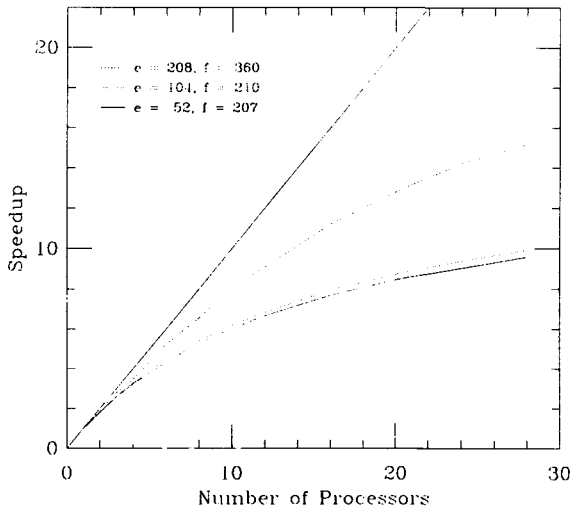

FIGURE 7 Speedup of the frontal solver on the MUSIC system: influence of the number of finite elements e and the maximal front width f.

ments is only slightly better than for 52 elements, whereas the speedup for 208 elements is considerably better. This is because the performance mainly depends on the front width, not on the number of finite elements—although a larger number of finite elements may result in a larger front width. Here, the (maximal) front widths are 207, 210, and 360, for the models with 52, 104, and 208 elements, respectively, which explains the measurements.

Time for Computing the Element Stiffness Matrices

To measure only the speedup of the solver itself, without being influenced by the times for the computation of the element stiffness matrices, all measurements given above take fixed element stiffness matrices, which requires no computation time (Fig. 8). In the following, the speedup of a complete frontal solver application, including the time for the computation of the element stiffness matrices, was determined. We simulate this time by repeating the assignment of the fixed element stiffness matrices either 10, 100, and 1,000 times. These measurements were carried out using the model with 52 elements, including both forward elimination and backward substitution. Because we have an almost ideal parallelization of this computation, the speedup considerably increases when the computation of stiffness matrices requires considerably more time. However, parallelization is only nearly ideal, as can be seen from the rapidly changing slopes in the curves, especially for the larger number of repetitions: The speedup depends on the number of chunks in which the element stiffness matrices are computed. For example, for 20 as well as for 24 processors three chunks of element stiffness matrices are computed: 20 + 20 + 12 = 52 and 24 + 24 + 4 = 52, respectively. This yields the same speedup for the parallelization of the computation of the element stiffness matrices for both 20 and 24 processors.

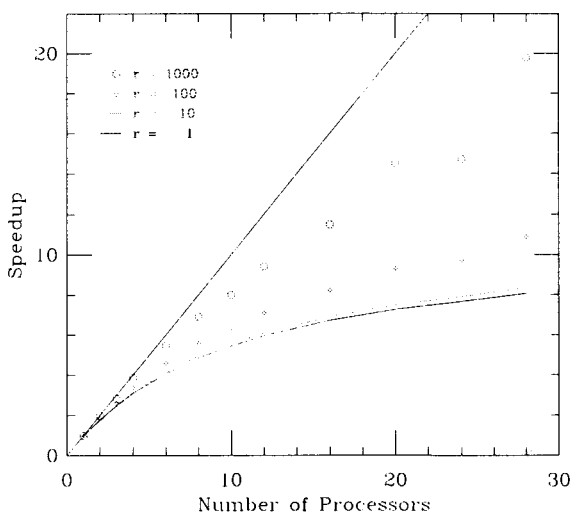

FIGURE 8 Speedup of the frontal solver on the MUSIC system: influence of the time for the computation of the element stiffness matrices, represented by the number of repetitions r of assigning fixed element stiffness matrices.

4 DYNAMIC SIMULATION

In contrast to the finite element simulation discussed above, this section deals with dynamic simulations of the human spine.

We have developed a mixed symbolic–numeric environment for the simulations of the rigid body models of the human spine. The need for such a system stems mainly from two reasons: (1) It eases the interdisciplinary work between biomechanical experts and computer scientists, and (2) it supports the evolutionary development of rigid body models. A dynamical skeletal model of the spine is a necessary first step, because neither a real human spine nor a physical model (or phantom) of a spine may be used as a control plant.

Our mixed symbolic–numeric environment consists of (1) a symbolic part implemented in Maple generating the problem-specific code (in C) for the simulation of the model and (2) a numerical part consisting of a set of library procedures for the numerical integration of ordinary differential equations that may be combined with the automatically generated code. Both of these subsystems will be explained in the following paragraphs.

4.1 Symbolic Step

The Lagrange formalism is a widely used technique to describe the dynamics of a mechanical system. Given the potential energy V and the kinetic energy T as functions of the generalized coordinates $q_k(t)$ and the corresponding velocities $\dot{q}_k(t)$, the Lagrange function L writes as $L = T - V$.

The unknown functions $q_k(t)$ are subjected to the condition, that the action functional

$$\int_{t_1}^{t_2} L(q_1, \ldots, q_n, \dot{q}_1, \ldots, \dot{q}_n, t)\, dt$$

takes an extremum for the true history $q_k(t)$ between arbitrary t_1 and t_2. The Euler–Lagrange equations, which solve this variational problem, are the well-known Newton equations of motion.

If we additionally consider energy dissipation by damping elements we get the following equations of motion

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_k}\right) - \frac{\partial L}{\partial q_k} + \frac{\partial \mathcal{F}}{\partial \dot{q}_k} = 0, k = 1, \ldots, n. \quad (1)$$

where $\mathcal{F}$ is the Raylaigh dissipation function.

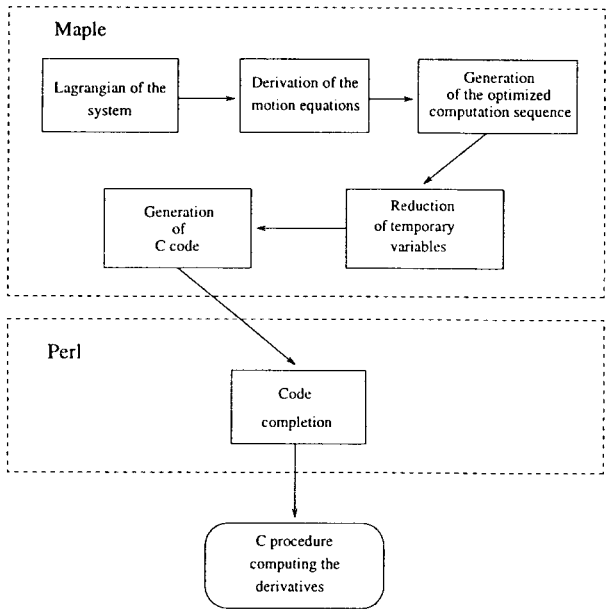

FIGURE 9 Overview of the mixed symbolic–numeric environment.

The Lagrange equations completely describe the dynamic properties of a mechanical system. Given L and $\mathcal{F}$, to obtain the equations of motion, the partial derivatives with respect to q_k and $\dot{q}_k$ as well as the total derivative with respect to t are determined symbolically. The equations of motion are second-order differential equations that can be integrated using well-known numerical algorithms.

We have implemented a Maple package that performs all symbolic transformations and the automatic code generation. Figure 9 outlines the data flow through the different modules of the package. The Maple package expects a description of V, T, and $\mathcal{F}$ as input. It first derives the equations of motion and transforms them, subsequently, into first-order differential equations. Thereafter, a sequence of assignment statements is computed; their right-hand sides consist of algebraic expressions and their left-hand sides are temporary variables. This sequence is time optimized, considering common subexpressions. Finally, a space optimization is applied, reducing the number of temporary variables. Using Maple's built-in procedure C, the package then generates a sequence of C assignment statements equivalent to the sequence of abstract assignment statements. In a last postprocessing step we complete the generated C fragment to obtain a separately compilable unit [10].

4.2 Numerical Step

The skeleton of an explicit numerical algorithm for the integration of ordinary differential equations is independent of a specific problem. For example, using a Runge–Kutta algorithm, the only problem-specific code is concentrated in a routine that computes the derivatives of the differential equations. This code is, according to the preceding section, automatically generated.

We implemented a set of standard numerical algorithms for the solution of ordinary differential equations (Runge–Kutta algorithms of different order), which may be combined as library routines with the automatically generated code to build up a simulation program. Technically speaking, the numerical library routines, the automatically generated code, and optional additional code (e.g., GUI code) are compiled separately and linked to a simulation program.

5 FINAL REMARKS

The principal application goal of this interdisciplinary project was to build suitable simulation tools, which allow for biomechanical simulations of the human spine. The scientific network built up for this project helped much in attacking this complex task.

The two approaches which we used for the simulations pursue different goals: (1) the finite element simulation of complex biomechanical models allows for fast execution times on parallel computers, and (2) the simulation development tool provides a symbolic–numeric environment with automatic code generation allowing for rapid experimenting with different dynamic models.

We have investigated linear equation solvers based on an iterative (conjugate gradient) and on a direct (frontal) method. The first method is often used in the context of finite element simulations. We have parallelized and implemented such an algorithm, however, the very different material properties of the present application require further investigations on the preconditioning. The frontal solver, already used in the original sequential implementation, served as a basis for the development of a new parallel frontal solver.

For the implementation of this parallel frontal solver we have used modern functional and imperative programming techniques: The preprocessor was realized in SML and the solver itself in C. This has in particular allowed us to exploit the respective typical advantages: the easy programmability in the functional, and the high efficiency in the imperative setting.

The speedup results achieved for this unstructured and irregular problem are more than satisfactory. The relatively small real arithmetic precision of the MUSIC (44 bits) is not an a priori limiting factor for this application. However, further investigations are necessary. Because of memory limitations of the MUSIC system (2.5 MBytes per node), it is planned to port the program to other parallel systems (currently the Parsytec PowerXplorer and Intel Paragon). This will subsequently allow us to run larger spine simulations as well as other finite element applications.

The automatic code generation tool allows for the evolutionary development of dynamic biomechanical models, this especially in fields where no widely accepted models are available (e.g., for forward dynamic models of the human spine). New models will be implemented quicker because biomechanical experts are relieved from the burden of coding. Rather, they express their models in a mathematical language with which they are familiar. The mixed symbolic–numeric approach shows a maximal flexibility regarding both the (application) problem formulation and the execution of the generated simulation program. Because the modular set-up of the simulation code allows for different enhancements such as the integration with other simulation programs, the addition of graphical user interface code, or the implementation of fast, possibly parallel code to solve the ordinary differential equations.

Finally, the predicted results from the various simulations will be validated by comparing them with experimental data provided by the medical partners.

ACKNOWLEDGMENTS

We would like to thank all collaborators who helped in realizing this project: M. Kientsch and M. Müller, Institute of Computer Science, University of Berne, Switzerland; M. Aebi, Division of Orthopaedic Surgery, McGill University, Montreal, Canada; M. Matyjewski, Warsaw University of Technology, Poland; S. M. McGill, Faculty of Health Science, University of Waterloo, Canada; D. Juker, Swiss Sports Institute, Magglingen, Switzerland; and B. Bäumle, Institute of Electronics, ETH Zurich, Switzerland. The SPINET research project was funded by the Swiss National Science Foundation, grants SPP-IF 5003-034405 and NF-7PLPJ038547.

REFERENCES

[1] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhoui, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Philadelphia: SIAM, 1994.

[2] G. Beer and W. Haas, "A partitioned frontal solver for finite element analysis," *J. Numerical Methods Eng.*, vol. 18, pp. 1623–1654, 1982.

[3] D. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation*. New Jersey: Prentice Hall, 1989.

[4] K. Broberg, "Slow deformation of intervertebral discs," *J. Biomech.*, vol. 26, no. 4-5, pp. 501–512, 1993.

[5] B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, and S. Watt, *Maple V Language Reference Manual*. New York: Springer, 1991.

[6] M. Dietrich, K. Kedzior, A. Wittek, and T. Zagrajek, "Non-linear finite element analysis of formation and treatment of intervention disc herniae," *J. Eng. Med.*, vol. 206, no. 4, pp. 225–231, 1992.

[7] M. Dietrich, K. Kedzior, and T. Zagrajek, "Finite element method analysis of human spine segments," in *Biomechanics XIA*. Amsterdam: Free University Press, 1988, p. 337.

[8] I. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices*. Oxford: Oxford University Press, 1986.

[9] V. Goel, T.-H. Lim, J. Gwon, J.-Y. Chen, J. Winterbottom, J. Park, J. N. Weinstein, and J.-Y. Ahn, "Effects of ridigity of an internal fixation device," *Spine*, vol. 16, no. 3, pp. 155–161, 1991.

[10] K. Guggisberg, "Neuromotoric stability control of the human spine—models, code generation and simulation," Master's Thesis, Institute of Computer Science and Applied Mathematics, University of Berne, Switzerland, 1994 (in German).

[11] A. Gunzinger, U. A. Müller, W. Scott, B. Bäumle, P. Kohler, H. R. v. Mühll, F. Müller-Plathe, W. F. v. Gunsteren, and W. Guggenbühl, "Achieving supercomputer performance with a DSP array processor," presented at *Int. Conf. on Supercomputing*, Minneapolis, Nov. 1992.

[12] T. Hughes, R. Ferencz, and J. Hallquist, "Large-scale vectorized implicit calculations in solid mechanics on a CRAY X-MP/48 utilizing EBE preconditioned conjugate gradients," *Comput. Methods Appl. Mech. Eng.*, vol. 61, pp. 215–248, 1987.

[13] B. Irons, "A frontal solution program for finite element analysis," *J. Numerical Methods Eng.*, vol. 2, pp. 5–32, 1970.

[14] D. Juker, S. McGill, P. Kropf, and T. Steffen, "Quantitative intramuscular myoelectric activity of the lumbar portions of psoas and the abdominal wall during a wide variety of tasks," *Spine*, 1995 (submitted).

[15] I. Kapandji, *The Physiology of the Joints, vol. 3, 2nd ed., The Trunk and the Vertebral Column*. New York: Churchill Livingstone, 1974.

[16] M. Matyjewski, P. Kropf, and M. Dietrich, "Finite element method simulation of flow induced deformations in the intervertebral disc," Presented at *The Lumbar Spine: A Basic Science Approach, First International Symposium*, Brussels, Aug. 1994.

[17] A. Nachemson and M. H. Pope, "Concepts in mathematical modeling," *Spine*, vol. 16, no. 6, pp. 675–676, 1991.

[18] L. C. Paulson, *ML for the Working Programmer*. Cambridge: Cambridge University Press, 1991.

[19] J.-G. Schneider, E. T. A. Lederer, and P. Schwab, "The solution of systems of linear equations using the conjugate gradient method on the parallel MUSIC-system," Institute of Computer Science and Applied Mathematics, University of Berne, Berne, Switzerland, Tech. Rep. IAM-94-014, Nov. 1994.

[20] H. Schwarz, *Numerische Mathematik*. zweite Auflage: B. G. Teubner, 1988.

[21] H. Schwarz, *Methode der Finiten Elemente*. zweite Auflage: B. G. Teubner, 1991.

[22] H. Schwarz, H. Rutishauser, and E. Stiefel, *Numerik symmetrischer Matrizen*. zweite Auflage: B. G. Teubner, 1968.

[23] A. Shirazi-Adl, "Finite-element simulation of changes in the fluid content of human lumbar discs. Mechanical and clinical implications," *Spine*, vol. 17, no. 2, pp. 206–212, 1992.

[24] W. Skalli, S. Robin, L. Lavaste, and J. Dubousset, "A biomechanical analysis of short segment spinal fixation using a three-dimensional geometric and mechanical model," *Spine*, vol. 18, no. 5, pp. 536–545, 1993.

[25] T. Steffen, H. Baramki, R. Rubin, J. Antoniou, Z. Zahid, L. Beckman, and M. Aebi, "Multilocalized pressure measurements within the loaded cadaveric intervertebral disc," Presented at ISSLS Meeting, Helsinki, Jul. 1995.

[26] T. Steffen, R. Rubin, J. Antoniou, M. Wegmueller, and M. Aebi, "Mechanical testing of cadaveric whole lumbar spine specimens using multiaxial external load histories," *CORS Orthop. Trans.* (published in *J. Bone and Joint Surgery [BC]*), 1994.

[27] T. Steffen, R. Rubin, H. Baramki, J. Antoniou, D. Marchesi, and M. Aebi, "A new technique for measuring lumbar segmental motion in vivo: Method, accuracy, and preliminary results," *Spine*, 1995 (submitted).

[28] V. Strumpen, "Lösung von schwach besetzten Gleichungssystemen aus der Schaltkreissimulation auf Parallel- und Vektorrechnern," Forschungszentrum Jülich, Tech. Rep. Jül-Spez-549, Jan. 1990.

[29] O. Zienkiewicz, *Methode der Finiten Elemente*. dritte Auflage: Hanser, 1984.