Distributed multi-scale muscle simulation in a hybrid MPI-CUDA computational environment

Simulation: Transactions of the Society for Modeling and Simulation International
XX(X):1–17

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Miloš Ivanović[1], Boban Stojanović[1], Ana Kaplarević-Mališić[1], Srboljub Mijailovich[2] and Richard Gilbert[2]

Abstract

We present Mexie, an extensible and scalable software solution for distributed multi-scale muscle simulations in a hybrid MPI-CUDA environment. Since muscle contraction relies on the integration of physical and biochemical properties across multiple length and time scales, these models are highly processor and memory intensive. Existing parallelization efforts for accelerating multi-scale muscle simulations imply the usage of expensive large-scale computational resources, which produces overwhelming costs for the everyday practical application of such models. In order to improve the computational speed within a reasonable budget, we introduce the concept of distributed calculations of multi-scale muscle models in a mixed CPU-GPU environment. The concept is applied to a two-scale muscle model, in which a finite element macro model is coupled with the microscopic Huxley kinetics model. Finite element calculations of a continuum macroscopic model take place strictly on the CPU, while numerical solutions of the partial differential equations of Huxley's cross-bridge kinetics are calculated on both CPUs and GPUs. We present a modular architecture of the solution, along with an internal organization and a specific load balancer that is aware of memory boundaries in such a heterogeneous environment. Solution was verified on both benchmark and real-world examples, showing high utilization of involved processing units, ensuring high scalability. Speed-up results show a boost of two orders of magnitude over any previously reported distributed multi-scale muscle models. This major improvement in computational feasibility of multi-scale muscle models paves the way for new discoveries in the field of muscle modeling and future clinical applications.

Keywords

multi-scale, muscle modeling, GPU, heterogeneous computing, scheduling policy

Date received 29 June 2015; reviewed 19 September 2015; accepted 3 November 2015

[1]Faculty of Science, University of Kragujevac, Radoja Domanovića 12, Kragujevac, Serbia
[2]Department of Chemistry and Chemical Biology, Northeastern University, 207 Hurtig Hall, 334 Huntington Avenue, Boston, MA 02115

Corresponding author:
Boban Stojanović, Faculty of Science, University of Kragujevac, Radoja Domanovica 12, 34000 Kragujevac, Serbia, +381 34 335 040
Email: bobi@kg.ac.rs

Prepared using sagej.cls *[Version: 2013/07/26 v1.00]*

Introduction

Investigating musculoskeletal disorders, as well as characterizing and prognosing neuromuscular diseases requires deep understanding of the structural and functional properties of muscles. The mechanical behavior of muscles is derived from the behavior of many individual components, such as cell membrane electrical conductivity and action potential, calcium dynamics, chemical reaction kinetics, and the actomyosin cycle, working together across spatial and temporal scales [20, 14]. The expansion of knowledge regarding the complex mechanisms underlying muscle physiology solely leaning on *in vivo* and *in vitro* experiments is quite slow and limited. *In silico* analysis of detailed muscle models enables less expensive and time consuming hypothesis testing and evaluation, therefore providing a valuable tool in research activities [12, 31]. Comprehensive and efficient computer models also play a key role in development of the future software solutions which may be used in everyday clinical practice.

Generally speaking, existing computational muscle models fall into two classes: (1) biophysical, which investigates the ability of contractile proteins to generate force and movement at the cellular level and (2) phenomenological, which evaluates the performance of the whole muscle. Most biophysical models evolve from the hypothesized cross-bridge kinetic concepts originally formulated by A.F. Huxley [20]. In Huxley-type models [14, 13, 29, 11], state transitions between myosin and actin states define force generation and relative velocity between sliding filaments. Mijailovich et al. reformulated A. F. Huxley's sliding filament theory by combining the partial differential equations (PDEs) approach to calculate the traction between actin and myosin filaments and the finite element method (FEM) to assess the deformation of extensible actin and myosin filaments [25]. Simulations of these kinetic processes, in the context of whole muscle models are tremendously computationally intensive and require simplifications of geometry, composition, and activation [29]. Even with these simplifications, whole muscle models are still computationally demanding, and therefore models of whole muscles generally employ phenomenological concepts. The most widely used phenomenological model, the Hill model only takes into account the relationship between active stress and strain rate, and its use is therefore limited to isometric and steady state contractions [19, 33, 21, 26]. Similar to many other phenomenological models, this model exclusively uses macroscopic paradigms, where a muscle is considered to be a mechanical system with defined material characteristics. Thus, while practically useful, the Hill model is often inadequate for simulations of motor physiology. This is due to the fact that the Hill model is defined by a few macroscopically measured constitutive parameters and thus does not take into account the orders of magnitude of change in muscle stiffness during contraction. This condition therefore results in an inappropriate balance between active, passive and external forces, especially when muscle deformation is unsteady and non-uniform. These deficiencies in phenomenological approaches invite the development of multi-scale models of muscle contraction that employ models of molecular interactions to calculate the instantaneous macroscopic constitutive material characteristics of muscle i.e. a dynamic constitutive relationship, necessary for quantitative models of whole muscle functional behavior.

In the last decade, significant efforts have been made in developing multi-scale muscle models. Considering both the Hill and Huxley models, following the ideas presented in by Bestel [5, 6], Makssound integrated the behavior of striated muscle ranging from the microscopic to the macroscopic scale [15]. This enabled the computation of the output force and stiffness by including a description of the force-length relationship and the influence of the velocity on cross-bridge breakage at the microscopic scale. Fernandez at al. [16] and Böl [7, 8] calculated the mechanics of 3D skeletal muscle models by implementing electro-physiological principles. Until now, the most comprehensive multi-scale muscle model was proposed in [30] by Röhrle at al. It is a physiologically based, multi-scale skeletal muscle finite element model, where the electro-physiological behavior of single muscle fibers within motor units may be computed and linked to a continuum-mechanical constitutive law. Their model is capable of representing detailed, geometrical descriptions of skeletal muscle fibers and their grouping.

Regarding the fact that multi-scale models are more informative and realistic, their practical implementation and usage in real-world applications is limited by their requirements for computational power. Such model simulations are computationally demanding, requiring significant memory consumption and the use of high performance computing (HPC) environments. Modern HPC environments are typically heterogeneous, including a variety of multiprocessing architectures such as fast multi-core processors, general purpose graphic processing units (GPUs), clusters of highly interconnected CPU's, and grid/cloud infrastructures. Efficiently harnessing the available architectures requires fine-grained levels of parallelism in order to overcome the differences in bandwidth and memory capacity available to each processing unit and to adapt scheduling policies to processing speeds. Moreover, the use of hybrid-programing models is also needed.

A few general frameworks that enable defining multi-scale models and their execution in a parallel manner have been reported. The latest version of the MUSCLE library [4], is a general multi-scale computational framework which is compatible with MPI (Message Passing Interface), OpenMP (Open Multi-Processing) and threading codes. It separates the runtime environment from sub-model API definitions and coupling scripting environment, therefore allowing for the execution of a model on heterogeneous machines where various serialization and communication methods are enabled [9]. The OpenCMISS framework is a computational software library for multi-scale modeling of physiological systems [10]. It resulted from the collaborative VPH/Physiome project and it uses models described by the VPH/Physiome standards for public domain frameworks [28]. The framework has support for distributed execution in a heterogeneous multiprocessing environment. It uses the MPI standard for distributed parallelization and the OpenMP standard for shared memory parallelization.

To date, the only reported implementation of a multi-scale muscle model capable of execution on a distributed computer system was described by Heidlauf and Röhrle [18], who employed the OpenCMISS framework. They conducted a process of distributing calculation units using the MPI paradigm. The speedup they reported looks promising in terms of scalability, but tests were performed on only 4 processors, which is not enough to determine the global trend. Despite potentially good characteristics, it is questionable if the implemented parallelization methodology is sufficient to enable the usage of the model in real-world applications and future medical practice. Obtaining useful results within the time frame appropriate for real-world applications requires very high speed-ups, since sequential executions of such models last for days or even weeks. Lowering total execution time to a few hours or less demands dozens or even hundreds of processor cores. That results with a price-performance ratio high enough to hinder any intensive development and usage of these models. Keeping in mind that many physiological processes within muscles are highly suitable for the GPU implementation, it is realistic to expect that employing a collection of available GPUs in addition to standard CPUs can result in significant speedup and scalability improvements. Thus, the utilization of GPUs can substantially reduce total execution time, with a significantly lower price-performance ratio.

To address this opportunity, we define a methodology for distributed computation of multi-scale muscle models in heterogeneous computing environments, which can employ an arbitrary number of CPU and GPU units. We present extensible and scalable software solution, named Mexie, which is built on top of MPI and Compute Unified Device Architecture (CUDA) programming models [27], and provides distributed and parallel execution of multi-scale muscle model simulations. The concept of parallelization is presented in the context of a model that simulates muscle behavior across two spatial and two temporal scales. We model macroscopic muscle mechanics using a finite element (FE) method, whereas the material attributes of the muscular tissue comply to the Huxley model [20], employed at the microscopic scale. The adapted algorithm for coupling the FE macro model with Huxley micro models enables distributed calculations in a hybrid MPI-CUDA environment. Finite element calculations of the continuum macroscopic model run strictly on the CPUs, while Huxley micro models associated with the integration points execute on both CPUs and GPUs. A specially

adapted load balancing algorithm provides high utilization of all involved processing units, ensuring high scalability. Significant performance boosts opens the door for new discoveries in the field of muscle modeling and clinical application.

Multi-scale muscle model

A multi-scale muscle model provide a method to characterize deformation and force generation based on instantaneous material properties and component geometry. At the molecular scale, strain-dependent transitions between molecular contractile states define the instantaneous capacity of a muscle to generate active force and stiffness. This instantaneous muscle stiffness directly contributes to local constitutive relationships and thus the balance of forces. Instantaneous material attributes of the muscular tissue, derived from local actomyosin (molecular) interactions within muscle fiber and the material characteristics of the surrounding connective tissue, were prescribed in the FE integration points (Figure 1). Using this comprehensive methodology, the configuration at a given time was obtained from the displacement field, which is incrementally calculated in an iterative scheme by the equilibrium of internal forces, originating from muscle contraction and connective tissue elasticity, and external forces external from the boundary and loading conditions. Macroscopic stresses were obtained from the active stress acting in the direction of muscle fibers and the components of elastic tensor representing connective tissue resistance to deformation.

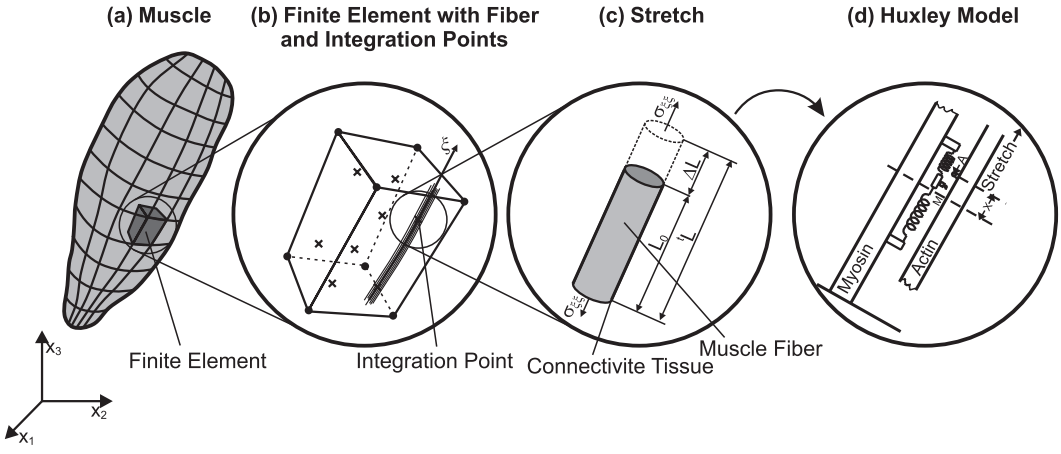

Figure 1. Multi-scale model of muscle contraction. a) Muscle finite element discretization; b) diagram depicting the muscle fibers contained within a characteristic tree-dimensional (3D) FE, including denoted FE integration points and the principal direction muscle fibers, ξ ; c) elongation of an individual muscle fiber, ΔL, at the indicated spatial scale and under stress, $\sigma_{\xi\xi}$; ΔL is calculated from current length, ${}^{t}L$, and slack (relaxed) length L_0; the current time is denoted as t; d) Huxley cross-bridge kinetics model [20]

Macroscopic model

The governing equilibrium equation of an FE structure in deformed configuration at a time step (t) (i) and iteration is formulated as:

$$\left({}^{t+\Delta t}K_{el} + {}^{t+\Delta t}\,K_{mol}\right)^{(i-1)}\Delta U^{(i)} = {}^{t+\Delta t}F_{ext}^{(i-1)} + {}^{t+\Delta t}\,F_{int}^{(i-1)} + {}^{t+\Delta t}\,F_{active}^{(i-1)} \tag{1}$$

where ${}^{t+\Delta t}F_{ext}^{(i-1)}$, ${}^{t+\Delta t}F_{int}^{(i-1)}$, and ${}^{t+\Delta t}F_{active}^{(i-1)}$ are vectors of external physiological loads, internal (structural) nodal forces, and integrated active molecular forces lumped into FE nodal forces, respectively; ${}^{t+\Delta t}K_{el}$ and ${}^{t+\Delta t}K_{mol}$ are stiffness matrices of the passive components of the constitutive FE and of cumulative stiffness of actomyosin bonds, respectively; $\Delta U^{(i)}$ are the increments of nodal displacements at iteration (i); and the left-upper index $t + \Delta t$ indicates that the equilibrium equations correspond to the end of the time step [21]. The key step in a standard FE formulation is the

evaluation of the element nodal internal and active forces:

$$ {}^{t+\Delta t}F_{int}^{(i-1)} + {}^{t+\Delta t}F_{active}^{(i-1)} = \int_{t+\Delta t V^{(i-1)}}^{n+1} {}^{t+\Delta t}B_L^{T(i-1)} {}^{t+\Delta t}\sigma^{(i-1)} dV \tag{2} $$

where ${}^{t+\Delta t}B_L^{T(i-1)}$ is the geometric linear strain-displacement matrix (superscript T means transpose), ${}^{t+\Delta t}\sigma^{(i-1)}$ is the stress tensor within the muscle, and ${}^{t+\Delta t}V(i-1)$ is the volume of an FE. After assembling the element balance in equations 1, the FE equilibrium equations for the entire muscle (Figure 1(a)) are solved, securing the equilibrium of F_{ext},F_{int}, and F_{active} within the prescribed tolerance at the end of each time step, $t + \Delta t$ [3, 22]. The displacement vector $U^{(i)}$ is updated during each iteration by the current increment $\Delta U^{(i)}$, until $\Delta U^{(i)} \approx 0$ at convergence. Active force generation, ${}^{t+\Delta t}F_{active}^{(i-1)}$, and stiffness, ${}^{t+\Delta t}K_{mol}^{(i-1)}$, are directly dependent on the rate of muscle deformation in the principal direction of muscle fibers [21].

Microscopic model

In 1957, Huxley proposed the model for the isometric and isotonic contraction of a striated muscle, suggesting that muscle contraction results as the cross-bridges linking the actin and myosin molecules pull the filaments over one another.[20] Following Huxley, a muscle fiber constitutive unit is represented by interacting actin and myosin filaments via so-called cross-bridges [32]. The cross-bridges form elastic connections between these filaments, and these spring like connections generate, in aggregate, the active muscle force and stiffness (Figure 1(d)). The myosin heads act as thermal ratchets and can attach to the closest actin site. During the binding process, thermal fluctuations provide the energy to extend cross-bridge elements in order to reach the closest actin site. Because the binding is allowed only when the cross-bridge spring is stretched, each freshly attached cross-bridge contributes to the active contractile force. However, over time, depending on boundary conditions the filaments can slide relative to each other so cross-bridges can experience both tension and compression. Following these rules Huxley's sliding filament theory [24] is defined by the following partial differential equation defined over the domain Ω:

$$ \frac{\partial n}{\partial t}(x,t) - v\frac{\partial n}{\partial x} = N(n(x,t),x), \forall x \in \Omega \tag{3} $$

where $n(x,t)$ is the fraction of attached cross-bridges displaced for x from its strain free position at time t; $v = -dx/dt$ is the shortening velocity of the thin filament with respect to the thick filament; $N(n(x,t),x) = [1 - n(x,t)] - n(x,t)g(x)$ is the compounded transition flux between the attached and detached states where $f(x)$ and $g(x)$ are the attachment and detachment rates that strictly depend on the distance x. For solving the first order hyperbolic equation 3 we used the method of characteristics [23, 25] which leads to the following iterative algorithm:

$$ \begin{aligned} {}^{t+\Delta t}x^{(i)} &= {}^{t}x + \frac{1}{2}\left({}^{t+\Delta t}v^{(i-1)} + {}^{t}v\right)\cdot\Delta t \\ {}^{t+\Delta t}n^{(i)} &= {}^{t}n + \frac{1}{2}\left({}^{t+\Delta t}N^{(i-1)} + {}^{t}N\right)\cdot\Delta t \end{aligned} \tag{4} $$

The specific muscle tension arises from the distortion of the cross-bridge represented as a linear spring and can be calculated from:

$$ F(t) = \kappa \cdot \int_{-\infty}^{\infty} x \cdot n(x,t) \tag{5} $$

where κ is the cross-bridge stiffness. Instantaneous specific muscle stiffness is then defined by the integral

$$\mathcal{K}(t) = \kappa \cdot \int_{-\infty}^{\infty} n(x,t) \tag{6}$$

Coupling FEs and Huxley model

The active stress generated within a muscle is calculated as $\sigma_m = \mathcal{F}\frac{\sigma_{iso}}{\mathcal{F}_{iso}}$, where σ_{iso} is a maximal isometric stress and $\mathcal{F}_{iso}$ is a specific maximal force calculated by the Huxley model for isometric contraction $(v = 0)$. The total stress σ is expressed as the contribution of active muscle forces and the contribution of (passive) elasticity of collagenous connective tissue, cell membrane, and muscle noncontractile cytoskeleton in parallel to muscle cells:

$$\sigma = \sigma_m \phi + \sigma^E (1 - \phi) \tag{7}$$

where ϕ is the fraction of muscle fibers in the total muscle volume and σ^E is the stress in the passive part of the muscle. The member of the tangent constitutive matrix in the fiber direction can be calculated as follows:

$$\bar{C}_{11} = \phi \frac{\partial \sigma_m}{\partial e} + \bar{C}^E_{11}(1 - \phi) \tag{8}$$

where $\bar{C}^E_{11}$ is the element of elastic tangent constitutive matrix of connective tissue in the fiber direction, and $\frac{\partial \sigma_m}{\partial e} \sim \mathcal{K}$ is the instantaneous muscle stiffness also in the fiber direction.

Hybrid MPI-CUDA computational environment solution

We devised our software to be executed on heterogeneous high performance computational resources, primarily a computer clusters with GPUs as additional computational devices. In order to support execution on the distributed memory architectures, we used MPI as a language-independent communication protocol, which provides both point-to-point and collective process communication. It enables easy development of portable and scalable parallel applications. To utilize GPUs, our solution employs the CUDA programming model. CUDA is a proprietary library that enables execution of data-parallel and compute-intensive task on NVIDIA GPUs. The task has the form of kernels, which represent the certain sequence of commands that perform calculations over a portion of data. CUDA distributes computations into a grid of GPU thread blocks, where each thread executes a kernel over a certain portion of data. The kernel assumes that its portion of data is located within GPU memory.

We have developed a software platform for parallel and distributed execution of two-scale muscle model simulations within a heterogeneous CPU/GPU environment. The key concept of the system is its parallelization strategy. The starting point in the applied strategy was dividing each iteration, during incremental two-scale model simulation, into two distinct sets of algorithm steps. The first set considers the finite element algorithm calculations such as assembling the element balance equations (1) and solving the FE equilibrium equations for the entire muscle. The second set considers micro-scale models of muscle fibers for each integration point within the FE mesh. Our solution applies parallelization only to the second set, i.e. to micro-scale model calculations, while the FE algorithm executes in a sequential manner. The justification for this approach comes from the performance analysis of the sequentially executed simulations. Micro model computations represent more than 99.9% of the total execution time. The performance of parallel simulation runs in the *Mexie* solution assumes a set of MPI processes, with the fundamental idea being to divide the entire set of micro models into smaller collections called chunks. One of the processes, having the role of a manager, executes the FE calculations.

When the simulation algorithm reaches the point where micro model calculations should begin, each participating process performs simulations over the micro models from its domain of responsibility. Due to this domain decomposition, we reduce process intercommunication to only two operations: the scattering of the deformations and the gathering of the stresses together with stress derivations. According to the proposed *Mexie* design, any of the processes can be delegated to use the GPU device for calculation. We detail the design of *Mexie*, paying special attention to the concept of employing GPU devices, as well as the scheduling strategy defined in compliance with the heterogeneous nature of the computing environment.

Mexie system architecture

The *Mexie* system consists of three functional units: (1) the macroscopic model layer, (2) the intermediate model layer, responsible for coupling and distribution, and (3) the microscopic model layer (Figure 2).

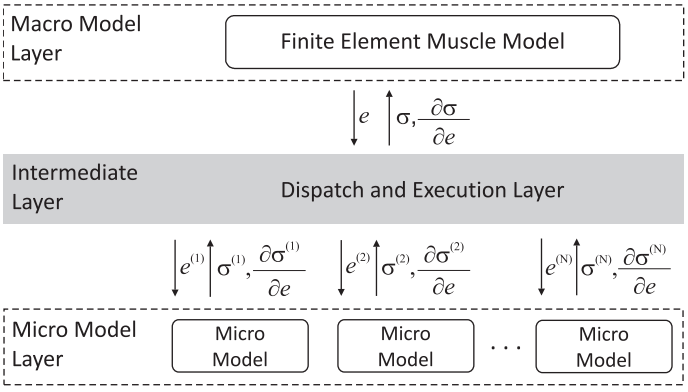

Figure 2. *Mexie* **architecture: High-level design.** The proposed system embodies the interface between a macro-scale layer, including the finite element model and its implementation, a micro-scale layer, including the material model that describes muscle behavior at the points of integration, and an intermediate layer, which acts to mediate the macro and micro-scale functions.

The macro model layer enables setting-up the FE model and running the entire two-scale simulation. In each iteration, the data containing current deformations in all integration points of the FE model are sent to the intermediate layer. Upon successful execution of the micro model steps, the intermediate layer returns stresses and their derivatives with respect to deformation.

The micro model layer includes the material models used to describe muscle behavior within the integration points of the macro model. There is a separate and independent micro model for each integration point, which entails establishing the algorithm that simulates the behavior of elementary building blocks like muscle fiber or sarcomere, and defining model parameters and model state variables. The micro model layer does not carry any information regarding the macro model and communicates exclusively with the intermediate layer. At the request of the intermediate layer, for a given strain, the micro model layer runs a simulation over the corresponding micro model, in order to determine stress and its derivative with respect to strain. In order to utilize various processing units (CPUs and GPUs), we have developed two distinct implementations of the Huxley micro model. The first one is dedicated to execution on CPUs, and the second one employs GPUs and uses the CUDA programming model.

The intermediate layer acts as a mediator between the macro-scale and the micro-scale layers, and its major role is to implement an adopted strategy of parallelization. This layer is also responsible for binding data, such as which micro model is associated with which FE integration point. In order to obtain the required stress values, the macro model layer sends a computation request to the intermediate layer, together with the strain tensors of all integration points.

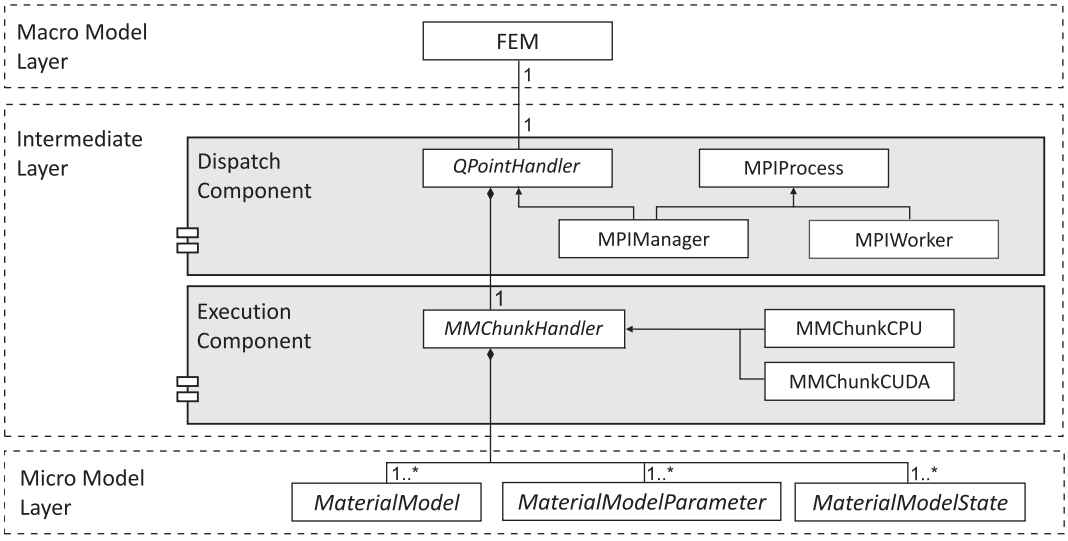

Figure 3. *Mexie* **architecture: Detailed design.** The multilayered computational environment is demonstrated, integrating the individual components of the macro-scale layer, principally the finite element model, the micro-scale layer, consisting of the material model description, and the delineation of material parameters and states, and the intermediate-scale layer, consisting of the dispatch and execution components.

The intermediate layer then decomposes the request and forwards its parts to the appropriate micro models. Once all micro models respond to the requests, the intermediate layer gathers the calculated stresses and their derivatives into a single collection. The entire collection is then returned to the macro model layer. Prior to the simulation run, we have to perform a mapping of the integration points to the MPI processes, which remains unchanged during the entire simulation run. This static mapping in a heterogeneous CPU/GPU environment is also a task that belongs to the intermediate layer. The intermediate layer consists of two distinct components (Figure 3): a dispatching and an execution component. The dispatching component is responsible for communication with the macro-scale model and the distribution of tasks to the micro-scale models, including the initial mapping prior to starting the simulation. The execution component takes care of the micro-scale model references, sending them real computation requests and interpreting their response. The central role within the dispatcher component has an abstract class *QPointHandler*. It represents the interface of the dispatching component toward the macro model layer. Its implementation within *MPIManager* supports the strategy of parallelization. As a supporting component for parallel execution, we implemented class referred to as *MPIWorker*. The basic idea applied within the strategy of parallelization is to divide the entire set of micro-models into chunks and associate them to the available MPI processes. An *MPIManager* instance is responsible for the division into chunks. It provides functionalities like obtaining which material model is associated to which integration point, arranging these material models across chunks, and obtaining which process rank is associated to which chunk. When the request for stresses and stress derivatives arrives from the macro model layer, *MPIManager* translates it into a number of partial requests and forwards them to the distributed *MPIWorker* instances. The requests are then delegated to the execution component. Upon completion of the stress calculation, the *MPIManager* collects all the results obtained by the workers and sends the response (stresses and stress derivatives) back to the macro model (Figure 4).

The execution component is responsible for communication with the micro model layer. It includes various implementations of *MMChunkHandler* whose role is to define an interface to the dispatching component and to communicate with a single chunk of the micro models. More precisely, the execution component breaks up each request of the dispatch component, sends appropriate chunks to the micro models and obtains results from them. Currently, we have two implementations of *MMChunkHandler*: *MMChunkCPU* and *MMChunkCUDA*. Both implementations

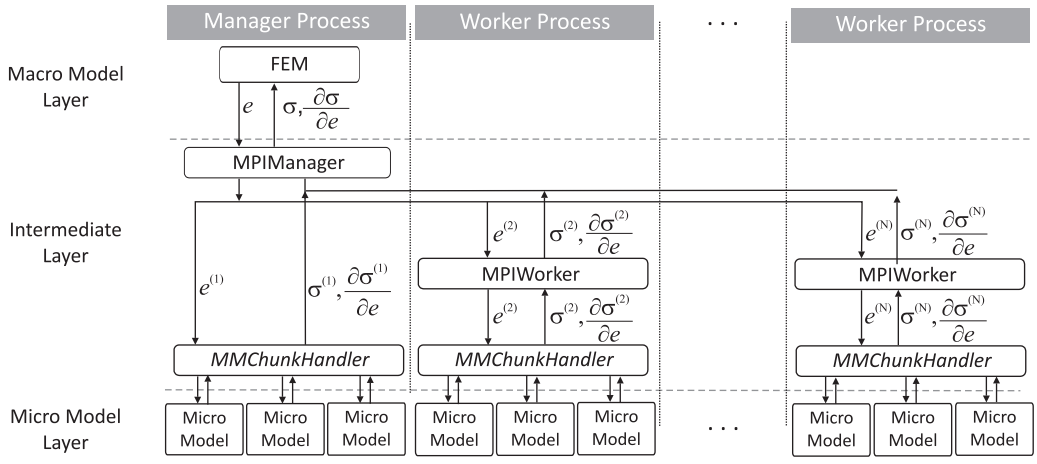

Figure 4. *Mexie* **class instances: Deployment and intercommunication diagram.** *MPIManager*'s responsibility is to manage the entire parallelization process, including division into chunks, translating FE model requests into appropriate partial requests to the instances of the *MPIWorker* and collecting results. A number of *MPIWorker* instances are deployed on various HPC resources, performing real micro-scale model computations.

initialize calculations for each individual micro model by invoking corresponding implementations of the Huxley model; *MMChunkCPU* invokes the the CPU implementation, while *MMChunkCUDA* invokes the CUDA implementation.

Mexie use of GPU resources

We utilize GPUs for the purpose of micro model computations. The CUDA micro model implementation assumes employing GPU in executing computationally intensive parts of the Huxley model simulation. Precisely, the GPU performs vector operations contained in the iterative algorithm formulated in the equation 4, while the CPU performs the rest of the algorithm in sequential manner.

As for the implementation, there is one CUDA-aware MPI driving process associated to each GPU unit. During simulation run, the driving process addresses a CUDA device by initiating the kernel launches. The kernels operate with few vectors, each containing a few thousand of floating-point values. Two of them, x and n represent attached crossbridge state distribution of each contractile unit (equation 4), implying that each micro model maintains its own pair of these vectors at any instance. Because the driving process manages a collection of micro model instances and kernels operate only with the data in GPU device memory, the total data transfer in each macro model iteration is not negligible. Aiming to reduce the device-host data transfer, the states of the micro models are stored in GPU's device memory throughout the entire simulation run. Since Mexie implements a static assignment of chunks to the processes, this rationalization of device-host communication highly benefits the performance. However, since such data organization is limited by the amount of GPU's global memory, we give special attention to the scheduling algorithm in the next section.

Each CUDA-aware driving process includes a single *MMChunkCUDA* instance that plays a role of the execution component, encapsulating all GPU utilizations. The Algortihm 1 describes the activities of *MMChunkCUDA* involving a GPU. Upon receiving request from the dispatch component, *MMChunkCUDA* loops over its micro model chunk. For each model instance, it copies model input data to the GPU global memory and then performs model calculations. New model state is determined upon executing CUDA kernels which compute new values of the vectors x and n. Since the algorithm employs the iterative scheme, we use the additional kernel to compute the error to satisfy convergence criterion. As soon as the micro model simulation completes, *MMChunkCUDA* initiates two kernel launches: one that calculates a value of the specific muscle tension F (equation 5); the other that calculates a value of the specific muscle stiffness

$\mathcal{K}$ (equation 6). Once when these values are transferred from the device memory to the host memory, *MMChunkCUDA* calculates σ and $\partial\sigma/\partial e$, following the equations 7 and 8.

foreach *Micromodel in chunk* **do**
  copy deformation e from the host memory to the GPU device memory;
  call kernel to initialize shortening velocity v;
  repeat
    repeat
      call kernel to calculate x;
      call kernel to calculate n;
      call kernel to calculate *IterationStopCondition*;
    until *IterationStopCondition is true*;
  until *SimulationStep is last*;
  call kernel to calculate F;
  call kernel to calculate $\mathcal{K}$;
  copy F and $\mathcal{K}$ from GPU device memory to host memory;
  calculate total stress σ and muscle stiffness $\partial\sigma/\partial e$;
end

Algorithm 1: Pseudo code describing GPU engagement in the micro model calculations implemented in *MMChunkCUDA* class, where n is the fraction of attached cross-bridges displaced for x from its strain free position, F specific muscle tension, $\mathcal{K}$ specific muscle stiffness.

Static task scheduler for heterogeneous MPI-CUDA environment

Due to the heterogeneous nature of the computing environment employed, task scheduling has a significant impact on *Mexie*'s performance. We developed a static scheduler whose objective is to perform initial partitioning of the set of all material models into chunks and assign those chunks to the participating MPI processes. The scheduling policy takes into account the speed of the participating processes, and the memory capacity of the devices on which the processes take place. The algorithm (shown in Figure 5) assumes the following input parameters:

- **Calculation speeds** of the participating MPI processes, identified as $V_i, i = \overline{1,k}$, where k is the total number of MPI processes (including ones whose purpose is restricted to GPU kernels invocation). We define the calculation speed as the average number of Huxley micro model iterations that the process executes in a unit of time.
- **Memory capacities** of the devices associated to the MPI processes, identified as $M_i, i = \overline{1,k}$. We define the memory capacity as the maximum number of micro models whose state variables can be stored during a simulation run. In the case that the process is associated to a GPU, the size of the "global" memory dictates the capacity value.

The first step in the proposed scheduling scheme is obtaining the value of time T, required to perform simulations over a total of N instances of micro-models, if memory is taken as unlimited:

$$T = \frac{N}{\Sigma_{i=1}^{k} V_i} \tag{9}$$

For each process, we determine the chunk size (number of micro models), that the process with speed V_i is capable to complete in time T:

$$C_i = T \cdot V_i \tag{10}$$

However, in the case that for the i-th process rank, the projected number of integration points exceeds memory capacity $C_i > M_i$, we have to adjust the speed of this process to $V_i = \frac{M_i}{T}$. Upon rewriting the speeds, we have to recalculate the synchronization time T and consequently the size of the chunks. The adjustments of V_i and C_i must be performed in an iterative manner, until all memory capacity requirements are satisfied. As the scheduler tends to distribute more models to faster processes (in this case the ones associated with the GPUs), it is more likely to reach their memory capacity, then reaching the memory capacity of the slower processes associated with the CPUs.

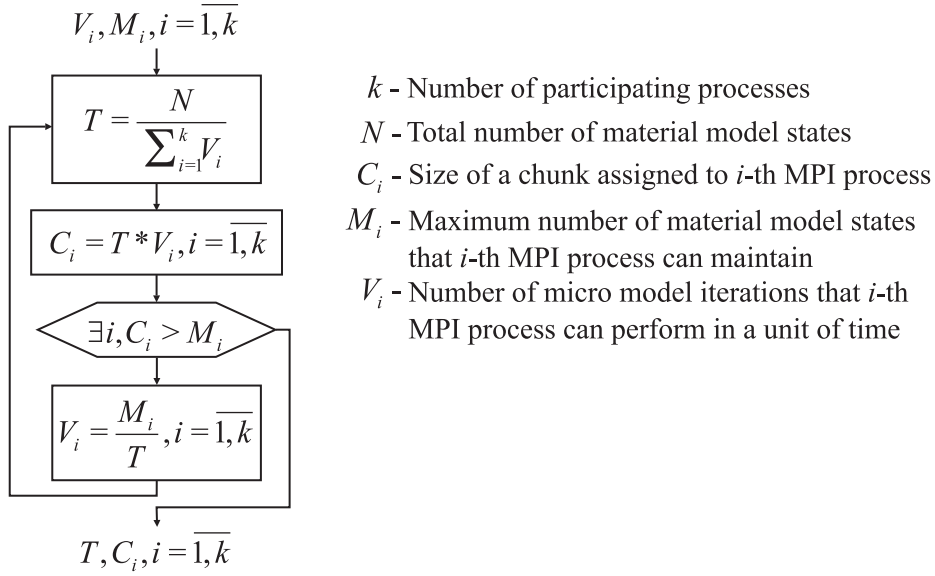

Figure 5. Scheduling algorithm: A representation of the task scheduling algorithm whose aim is to divide the entire micro-scale model domain into chunks and then to assign the integration tasks to the finite element model, taking into account the speed of the component processes and the actual memory capacity of the devices.

Case study: A multi-scale simulation of human tongue deformation during swallowing

We have previously published a method for representing a finite element model of the deforming tongue during swallowing [26]. We summarize in brief the salient aspects of this model for the purposes of the current demonstration. The tongue is an intricately configured muscular organ, which is capable of generating a variations of shapes and stiffness during human speech and swallowing [17]. During normal swallowing, the tongue undergoes a prescribed sequence of deformations, which contain, translate, and propel the bolus retrograde into the pharynx. Our approach involves the use of diffusion weighted MRI to delineate aligned multicellular fiber arrays based on anisotropic variations of water diffusion (representing preferential displacement of protons as a function of macromolecular barriers to diffusion) and the subsequent creation of a finite element mesh co-aligned with the diffusion tractography [2]. The development of an FE model of lingual mechanics provides a method to characterize tissue deformation and force generation based on the muscle's instantaneous material properties and geometry. At the molecular scale, strain-dependent transitions between molecular contractile states define the instantaneous capacity of a muscle to generate active force and stiffness. These molecular transitions can best be described by conservation laws expressed as field equations in a vector form. The solution of these partial differential equations provides a probability density function, from which the active component of stress and muscle stiffness originating in actomyosin bonds can be calculated. This instantaneous muscle stiffness directly contributes to local constitutive relationships and thus the balance of forces. Instantaneous material attributes of the

muscular tissue, derived from local actomyosin (molecular) interactions and the material characteristics of the connective tissue and muscle, can then be prescribed to FE integration points, also known as Gaussian points. The tissue scale myoarchitecture, obtained experimentally from diffusion-weighted MRI images, can be represented as a set of FEs aligned with the principal direction of the myofiber tracts. This anatomically aligned mesh thus provides a structural pattern upon which it becomes possible to carry out calculations of lingual deformation and force generation. The current simulation of local muscle stress employs the material assumptions associated with the Huxley cross-bridge kinetic model (see above). The aggregate outcome of this simulation is a theoretical representation of the deforming tissue, embodying concepts of multi-scale mechanics and MRI methods depicting lingual myoarchitecture to relate the physiological attributes of individual myocytes, coupled myocytes and myocyte regions during swallowing.

Results and discussion

We conducted the performance analysis of the described system on two artificial test cases, and on a real world quasi-3D model of the deforming tongue. As a hardware platform, we used a university cluster consisting of 22 nodes, each of them with dual Intel Xeon E5-2670@2.6GHz 8-core CPU, and 48GB of memory, with InfiniBand QDR interconnect. Four nodes were also equipped with Tesla M2090 accelerators with 6 GB of GDDR5 memory. For the purpose of the artificial test case, we chose a simple 2D rectangular muscle model with various mesh densities. The major aim of these benchmarks was to obtain speed-up values in mixed CPU/GPU setup. The adopted reference unit for all speed-up diagrams is the duration of the sequential run on Xeon E5-2670 CPU.

Prior to the real benchmark, we determined values of calculation speeds from equation (9), in order to provide input for the static scheduler. It turned out that our GPU is capable of processing approximately 22 times more micro model iterations than a single CPU core. In accordance with this ratio, we specified the test configurations, divided into four distinct groups according to their computing power, expressed in equivalent CPU count: $22, 44, 66$ and 88. The configurations are $1G/0C$ and $0G/22C$ for the first group, $2G/0C, 1G/22C, 0G/44C$ for the second group, $3G/0C, 2G/22C, 1G/44C, 0G/66C$ for the third group, and $4G/0C, 3G/22C, 2G/44C, 1G/66C, 0G/88C$ for the last, fourth group. The notation of the form nG/mC indicates that there are nG GPU driving processes and mC processes executing solely on CPU in certain simulation run. Additionally, we measured network bandwidth influence on overall simulation time. In case of the largest model that we used in our tests, employing standard gigabit instead of IB interconnect increased total execution time within range 0.13% to 0.53%. Since lowering bandwidth and latency by two orders of magnitude resulted in only few permillage of deceleration, we omit further experiments on gigabit interconnection. Figure 6 and Figure 7 show the results of the speed-up measurement, where each speed-up value is obtained as an average value taken from ten simulation runs.

Figure 6 shows the speed-ups achieved for the benchmark model with a mesh that consists of 250 quadrilateral finite elements, each of them with 4 integration points, totaling to 1000 integration points. Figure 7 shows the speed-ups for the larger benchmark model, which consists of 1000 quadrilateral finite elements, totaling to 4000 Huxley micro models. As stated in previous section, a GPU unit is capable of keeping only a limited number of micro models' states throughout the simulation process. This limitation influences the distribution of duties among the processes, and consequently the achieved speed-up value. Depending on the size of models and the applied configurations, the impact of the GPU units on the total speed-up can be less than assumed (~ 22 CPUs). Our Tesla M2090 units with 6GB have a capacity of approximately 1000 Huxley micro models. In accordance to the optimistic assumption that a single GPU unit delivers processing speeds equal to that of 22 CPUs, the configurations denoted as $0G/66C$ and $2G/22C$ should give approximately equal speed-up. However, in case of larger model (4000 Huxley micro-models), the GPU is limited to 1000 points, which is approximately 11 times the number of points allocated by CPU units. This means that the influence of

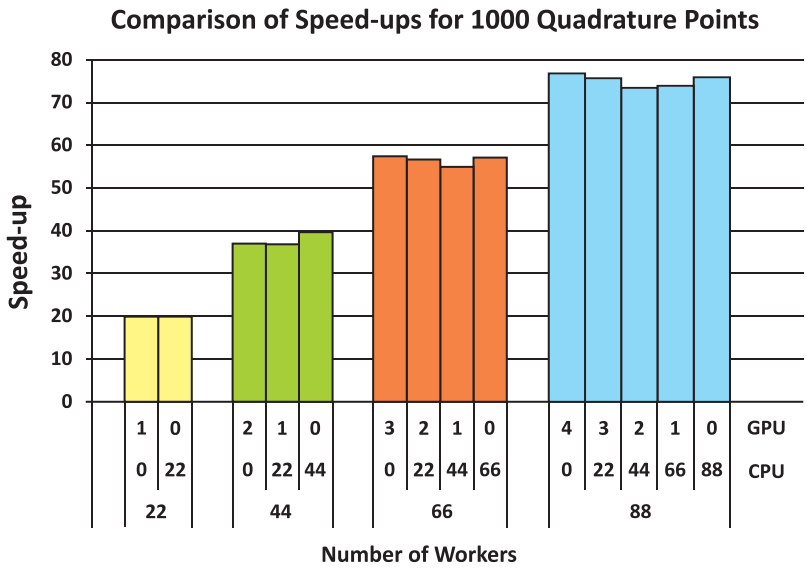

Figure 6. Speed-ups for the benchmark model with 250 FEs, obtained with equivalent computing power of: 22CPUs ($1G/0C$, $0G/22C$), 44CPUs ($2G/0C$, $1G/22C$, $0G/44C$), 66CPUs ($3G/0C$, $2G/22C$, $1G/44C$, $0G/66C$), 88CPUs ($4G/0C$, $3G/22C$, $2G/44C$, $1G/66C$, $0G/88C$). Execution times range from $19.22s$ for $4G/0C$ to $67.9s$ for $1G/0C$ and $0G/22C$. Sequential execution time for this benchmark example is $1340.34s$

.

a single GPU is approximately equivalent to the influence of the 11 CPUs (not the idealistic 22). In fact, in terms of the achieved speed-up, the configuration denoted as $2G/22C$ is approximately equal to the configuration $0G/44C$ (both of them allocate 1000 micro models per GPU), Figure 7.

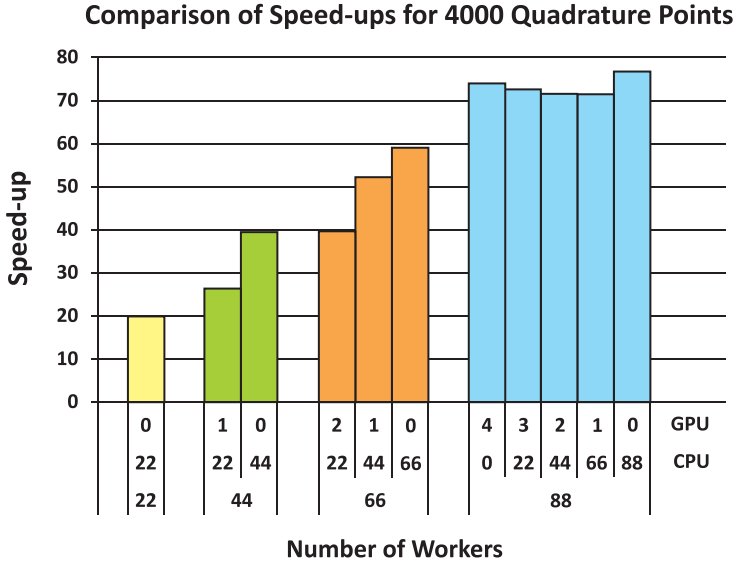

Figure 7. Speed-ups for the benchmark model with 1000 FEs, obtained with equivalent computing power of: 22CPUs ($0G/22C$), 44CPUs ($1G/22C$, $0G/44C$), 66CPUs ($2G/22C$, $1G/44C$, $0G/66C$), 88CPUs ($4G/0C$, $3G/22C$, $2G/44C$, $1G/66C$, $0G/88C$). Execution times range from $88.7s$ for $0G/88C$ to $342.72s$ for $0G/22C$. Sequential execution time for this benchmark is $6804.5s$

.

In addition to these artificial benchmarks, we tested the *Mexie* system on the real-world model of the complex muscle structure of the tongue. The multi-scale tongue model setup is identical to that described by Mijailovich at al. in [26].

The model consists of 873 finite elements, with 3492 integration points. We have simulated the lingual shape changes occurring following lingual tip contact with the hard palate during human swallowing, with a total duration of 0.5 seconds divided into 50 time steps (Figure 8). Sequential execution of the simulation took approximately 96.06 h. Running the same simulation in $4G/124C$ configuration lasted approximately 28.18 minutes which is 204.53 times faster.

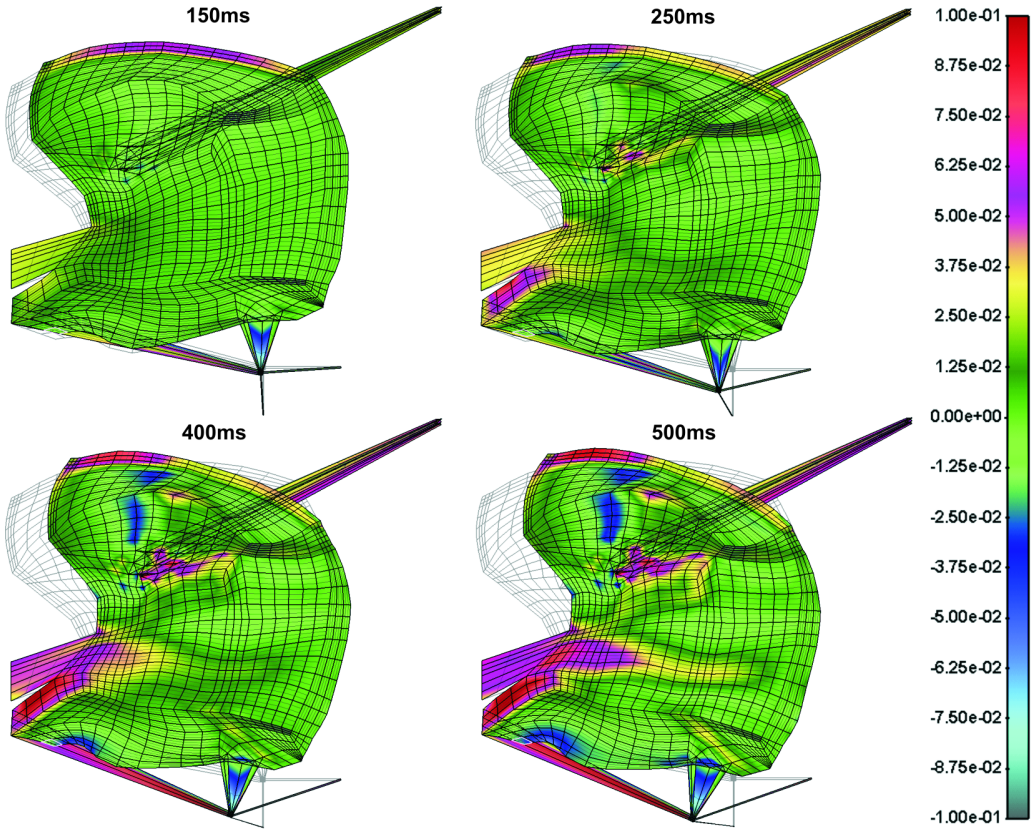

Figure 8. Finite element (FE) simulation of muscle stress for the deforming human tongue during physiological swallowing. The development of this FEM employs diffusion weighted MRI to defined fiber arrays based on anisotropic variations of water diffusion and the generation of an FE mesh co-aligned with the diffusion tractography. The instantaneous material attributes of the deforming tongue were obtained from local actomyosin interactions, elasticity of passive muscle structures and the material characteristics of the connective tissue. These material characteristics were then used for the calculation of the tangent constitutive matrix (Equation (8)) in the FE integration points. The implementation of this model employs the material assumptions of the Huxley model (see text for details). The color code represents the magnitude of muscle tension (MPa) in the direction of muscle fibers at four discrete time points following the initiation of the swallow, $150, 250, 400,$ and 500 ms

.

According to Figure 8 and Figure 7, it is obvious that utilizing CPU/GPU mixed environments provides major speed-ups, leading to excellent price-to-performance ratios. However, while using *Mexie* in production, one should keep in mind the memory limits of GPU devices, which can decrease the total GPU to CPU performance ratio. The maximum ratio for large model runs can be expected only in the case that there is a sufficient number of CPU processes available. Only than, the GPUs can reach exactly V_{GPU}/V_{CPU} times more micro-models (integration points) than the CPU processes, where V_{GPU} and V_{CPU} denote calculation speeds of GPU and CPU, respectively. Tests over real-world models of human tongue deformation have shown very good acceleration. Still, our parallelization strategy applied solely to micro model calculations has a theoretical speed-up limit, which is around 500, according to the Amdahl's law [1]. To summarize, the key features of *Mexie* software solution, giving it advantage over other related solutions are following:

1. the primary novelty of the *Mexie* software solution is the ability to employ GPU accelerators (in addition to standard CPUs) for very efficient parallel execution of the multi-scale muscle models;
2. *Mexie* shows strong scaling, thanks to the hybrid programming model and the optimized static task scheduler;
3. the significant improvements in performance by using GPUs provide very large performance-per-watt ratio and therefore achieves the same speedup at significantly lower cost and power consumption compared to traditional multicore CPUs;
4. the two orders of magnitude acceleration demonstrated by the *Mexie* software solution provides opportunity for development of complex (informative) multi-scale models and their usage in everyday medical practice.

Conclusion and future work

We have presented a novel concept of distributed multi-scale muscle modeling in a heterogeneous CPU/GPU environment. A variant of FEM-Huxley two-scale muscle model is decomposed in a way that enables computationally feasible coupling of information from two scales. Geometric attributes of the muscle are described within the scope of the macro model that runs on the CPU, while physiological behavior is the responsibility of the micro models running in parallel on both CPUs and GPUs. The proposed model is implemented in the *Mexie* software solution with a parallelization strategy based on the static decomposition of the microscopic model domain. The static task scheduler controls load balancing by adjusting workload distribution according to the computational speeds of the available CPUs and GPUs and the memory limitations of the GPU units. We performed tests on both benchmark models and a real-world model, demonstrating scalability of the system and speed-up of approximately two orders of magnitude compared to the sequential CPU run. The achieved performance boost within a relatively low price paves the path for introducing such models into clinical practice with a reasonable price-performance ratio, but also opens the door for new scientific discoveries in the field of biomedicine.

A note regarding the future development of the *Mexie*'s static scheduler, the current task scheduler assumes that all microscopic models assigned to integration points have equal computational complexity, which may not be true in the case of real-world examples. Considering the computational complexity of each microscopic model during domain decomposition will result in better load balance. It is also possible to overcome Amdahl's theoretical speed-up limit by additional parallelization of the macro model. Furthermore, due to Mexie's modular architecture, it should be straightforward to extend our software's functionality to accommodate various kinds of HPC resources, such as FPGA devices. This could offer even a lower the price-performance ratio, and contribute to more rational usage of available computational resources.

Acknowledgements

We like to thank our co-workers Marina Svičević, Djordje Nedić and Darko Antonijević for their help and support in the development of the presented software solution.

This work is supported by the Ministry of Science in Serbia, Grants III41007, OI174028, TR37013, and National Institutes of Health AR048776 and DC011528.

References

[1] Gene M. Amdahl. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, Reprinted from the AFIPS Conference Proceedings, Vol. 30 (Atlantic City, N.J., Apr. 18-20), AFIPS Press, Reston, Va., 1967, pp. 483-485, when Dr. Amdahl was at Inte". In: *IEEE Solid-State Circuits Newsletter* 12.3 (Jan. 2007), pp. 19–20.

[2] P J Basser, J Mattiello, and D LeBihan. “MR diffusion tensor spectroscopy and imaging.” In: *Biophysical journal* 66.1 (Jan. 1994), pp. 259–67.

[3] Klaus Jurgen Bathe. *Finite element procedures in Engineering Analisys*. Englewood Cliffs, NJ: Prentice-Hall, 1982, p. 735.

[4] Mohamed Ben Belgacem et al. “Distributed Multiscale Computations Using the MAPPER Framework”. In: *Procedia Computer Science* 18 (Jan. 2013), pp. 1106–1115.

[5] J Bestel. “Modele differentiel de la contraction musculaire controlee. Application au systeme carduio-vasculaire”. PhD thesis. University Paris IX Dauphine, FR, 2000.

[6] J Bestel and M Sorine. *A differential model of muscle contraction and applications. In: schloessmann Seminar on mathematical models in biology, chemistry and physics.* Bad Lausick, Germany, 2000.

[7] Markus Bol. “Micromechanical modelling of skeletal muscles: from the single fibre to the whole muscle”. In: *Archive of Applied Mechanics* 80.5 (Oct. 2009), pp. 557–567.

[8] Markus Bol, Roman Weikert, and Christine Weichert. “A coupled electromechanical model for the excitation-dependent contraction of skeletal muscle.” In: *Journal of the mechanical behavior of biomedical materials* 4.7 (Oct. 2011), pp. 1299–310.

[9] Joris Borgdorff et al. “Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment”. In: *Journal of Computational Science* 5.5 (Sept. 2014), pp. 719–731.

[10] Chris Bradley et al. “OpenCMISS: a multi-physics & multi-scale computational infrastructure for the VPH/Physiome project.” In: *Progress in biophysics and molecular biology* 107.1 (Oct. 2011), pp. 32–47.

[11] Leslie Chin et al. “Mathematical simulation of muscle cross-bridge cycle and force-velocity relationship.” In: *Biophysical journal* 91.10 (Nov. 2006), pp. 3653–63.

[12] J O Dada and P Mendes. “Multi-scale modelling and simulation in systems biology”. In: *Integrative biology : quantitative biosciences from nano to macro* 3.2 (Feb. 2011), pp. 86–96.

[13] Thomas L. Daniel, Alan C. Trimble, and P. Bryant Chase. “Compliant Realignment of Binding Sites in Muscle: Transient Behavior and Mechanical Tuning”. In: *Biophysical Journal* 74.4 (Apr. 1998), pp. 1611–1621.

[14] E Eisenberg, T L Hill, and Y Chen. “Cross-bridge model of muscle contraction. Quantitative analysis.” In: *Biophysical journal* 29.2 (Feb. 1980), pp. 195–227.

[15] Hassan El Makssoud et al. “Multiscale modeling of skeletal muscle properties and experimental validations in isometric conditions.” In: *Biological cybernetics* 105.2 (Aug. 2011), pp. 121–38.

[16] J W Fernandez et al. “Modelling the passive and nerve activated response of the rectus femoris muscle to a flexion loading: a finite element framework.” In: *Medical engineering & physics* 27.10 (Dec. 2005), pp. 862–70.

[17] Richard J Gilbert et al. “Anatomical basis of lingual hydrostatic deformation.” In: *The Journal of experimental biology* 210.Pt 23 (Dec. 2007), pp. 4069–82.

[18] Thomas Heidlauf and Oliver Rohrle. “Modeling the chemoelectromechanical behavior of skeletal muscle using the parallel open-source software library OpenCMISS.” In: *Computational and mathematical methods in medicine* 2013 (Jan. 2013), p. 517287. ISSN: 1748-6718.

[19] A. V. Hill. “The Heat of Shortening and the Dynamic Constants of Muscle”. In: *Proceedings of the Royal Society B: Biological Sciences* 126.B (1938), pp. 136–195.

[20] A. F. Huxley. “Muscle structure and theories of contraction.” In: *Progress in biophysics and biophysical chemistry* 7 (Jan. 1957), pp. 255–318.

[21] M. Kojic, S. Mijailovic, and N. Zdravkovic. “Modelling of muscle behaviour by the finite element method using Hill’s three-element model”. In: *International Journal for Numerical Methods in Engineering* 43.5 (1998), pp. 941–953.

[22] Milos Kojic and Klaus-Jurgen Bathe. *Inelastic analysis of solids and structures*. Springer, 2005.

[23] Mary Lister. “The numerical solution of hyperbolic partial differential equations by the method of characteristics”. In: *Mathematical methods for digital computers* 1 (1960), pp. 165–179.

[24] Thomas A McMahon. *Muscles, reflexes and locomotion*. Princeton University Press, New Jersey, 1984.

[25] S M Mijailovich, J J Fredberg, and J P Butler. “On the theory of muscle contrction: filament extensibility and the development of isometric force and stiffness.” In: *Biophysical journal* 71.3 (Sept. 1996), pp. 1475–84.

[26] Srboljub M Mijailovich et al. “Derivation of a finite-element model of lingual deformation during swallowing from the mechanics of mesoscale myofiber tracts obtained by MRI.” In: *Journal of applied physiology (Bethesda, Md. : 1985)* 109.5 (Nov. 2010), pp. 1500–14.

[27] NVIDIA. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. Tech. rep. 2007.

[28] Physiome. *Physiome project*. URL: `http://physiomeproject.org/`.

[29] M V Razumova, A E Bukatina, and K B Campbell. “Stiffness-distortion sarcomere model for muscle simulation.” In: *Journal of applied physiology (Bethesda, Md. : 1985)* 87.5 (Nov. 1999), pp. 1861–76.

[30] Oliver Rohrle, J B Davidson, and Andrew J. Pullan. “A physiologically based, multi-scale model of skeletal muscle structure and function.” In: *Frontiers in physiology* 3.September (Jan. 2012), p. 358.

[31] Peter M Sloot and Alfons G Hoekstra. “Multi-scale modelling in computational biomedicine.” In: *Briefings in bioinformatics* 11.1 (Jan. 2010), pp. 142–52.

[32] A. Torelli. “Study of a mathematical model for muscle contraction with deformable elements”. In: *Rend. Sem. Mat. Univ. Politec. Torino* 55 (1997), pp. 241–271.

[33] F E Zajac. *Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control.* Vol. 17. 4. Jan. 1989, pp. 359–411.

Biographies

Miloš Ivanović is an assistant professor at the Department of Mathematics and Informatics, Faculty of Science, University of Kragujevac, Serbia.

Boban Stojanović is an associate professor at the Department of Mathematics and Informatics, Faculty of Science, University of Kragujevac, Serbia.

Ana Kaplarević-Mališić is a teaching assistant at the Department of Mathematics and Informatics, Faculty of Science, University of Kragujevac, Serbia.

Richard J. Gilbert is a research professor in the Department of Chemistry and Chemical Biology at Northeastern University, Boston, MA, USA.

Srboljub M. Mijailovich is a research professor at the Department of Chemistry and Chemical Biology at Northeastern University, Boston, MA, USA.