Validation and Optimisation of Cardiac Cellular Models

Software Engineering Centre
Randall Owen

Outline

- Introduction and motivation
- Units Checking
- Optimisation
  - Adaptivity
  - Partial Evaluation
  - Lookup Tables
- Future work

Computer Science perspective

- CellML looks like a (domain specific) programming language
- Interesting dynamic evaluation
- Static (i.e. compile-time) checks
- Provably correct optimisations

Why Validate?

- To prevent incorrect results
- e.g.NASA's Mars Climate Orbiter, Sept. 23, 1999.

"People sometimes make errors. The problem here was not the error, it was the failure of NASA's systems engineering, and the checks and balances in our processes to detect the error. That's why we lost the spacecraft." — Edward Weiler, NASA's Associate Administrator for Space Science

Levels of Validation

- Well-formed XML
  - XML Parser
- Grammatical correctness
  - RELAX NG
- Name references match defined names
  - Schematron
- Equation dimension checking
  - Python
- ‘Curation’

Units Checking

- Analogous to type-checking a programming language

$$\forall R \in \mathit{Relops}\,\frac{\mathrm{e}_1 :: u \quad \mathrm{e}_2 :: u}{\mathrm{e}_1\ R\ \mathrm{e}_2 :: \texttt{cellml:boolean}}$$

$$\frac{\mathrm{e}_1 :: u \quad \mathrm{e}_2 :: v}{\mathrm{e}_1/\mathrm{e}_2 :: u \otimes v^{-1}}$$

$$\frac{\mathrm{e}_1 :: u \quad \mathrm{e}_2 :: \texttt{dimensionless}}{\mathrm{e}_1^{\mathrm{e}_2} :: u^{\mathrm{e}_2}}$$

- Can also do automatic units conversion

Automatic Optimisation

- We simulate by translating CellML to code and compiling
- Compiler optimisation is helpful
- But compilers only perform general optimisations
- There are also domain-specific optimisations
  - Adaptive solution of the bidomain equations
  - Staging work in an ODE solver
  - Lookup tables
- Performing these by hand is tedious and non-scalable

Adaptive algorithm

- Adaptive solution scheme for the bidomain equations developed by Jonathan Whiteley
  - "An efficient numerical technique for the solution of the monodomain and bidomain equations", TBME 53(11)
- At each time step n
  1. Solve the PDEs for V_m^n using a semi-implicit method
  2. Use V_m^n to solve the ODEs using backward Euler, and Newton's method to solve the nonlinear system

Analysis for Adaptivity

- For efficiency, this technique requires modifications to the cell models
  - Decouple, separating linear and nonlinear ODEs
  - Rearrange linear ODEs so the backward Euler update is explicit
  - Compute a symbolic Jacobian for the nonlinear system
- We have automated these transformations
- Adaptivity requires collaboration of the simulation framework—work in progress

Experimental results

- The results of the above analyses were used to generate cell model code for Oxford Integrator (Chaste). The Luo-Rudy I and Noble '98 models were used as initial tests.
- Using the decoupled cell model allows a larger ODE timestep to be used, resulting in a 30% speedup for Luo-Rudy I in a simple 3D simulation.
- Note that no adaptivity was used in generating these results, as Chaste does not yet support this.

Staging by Partial Evaluation

- An ODE solver is essentially a loop over time
- Some computations are the same at every time-step and depend only on information available within the model
  - So perform them once only
- The context is too complicated for a compiler to do this for us so use a partial evaluator
- A partial evaluator is an automatic tool that pre-computes parts of a program known at compile time

Lookup Tables

- Many expressions depend only on the transmembrane potential V
  e.g. $\beta_h = \frac{1}{e^{-(V+45)/10}+1}$
- Usually V takes values in the range [-100, 50]mV
- We can thus:
  - Tabulate expression values prior to simulation
  - Use linear interpolation to look up a value for the expression given V
  - This is faster than computing an exponential

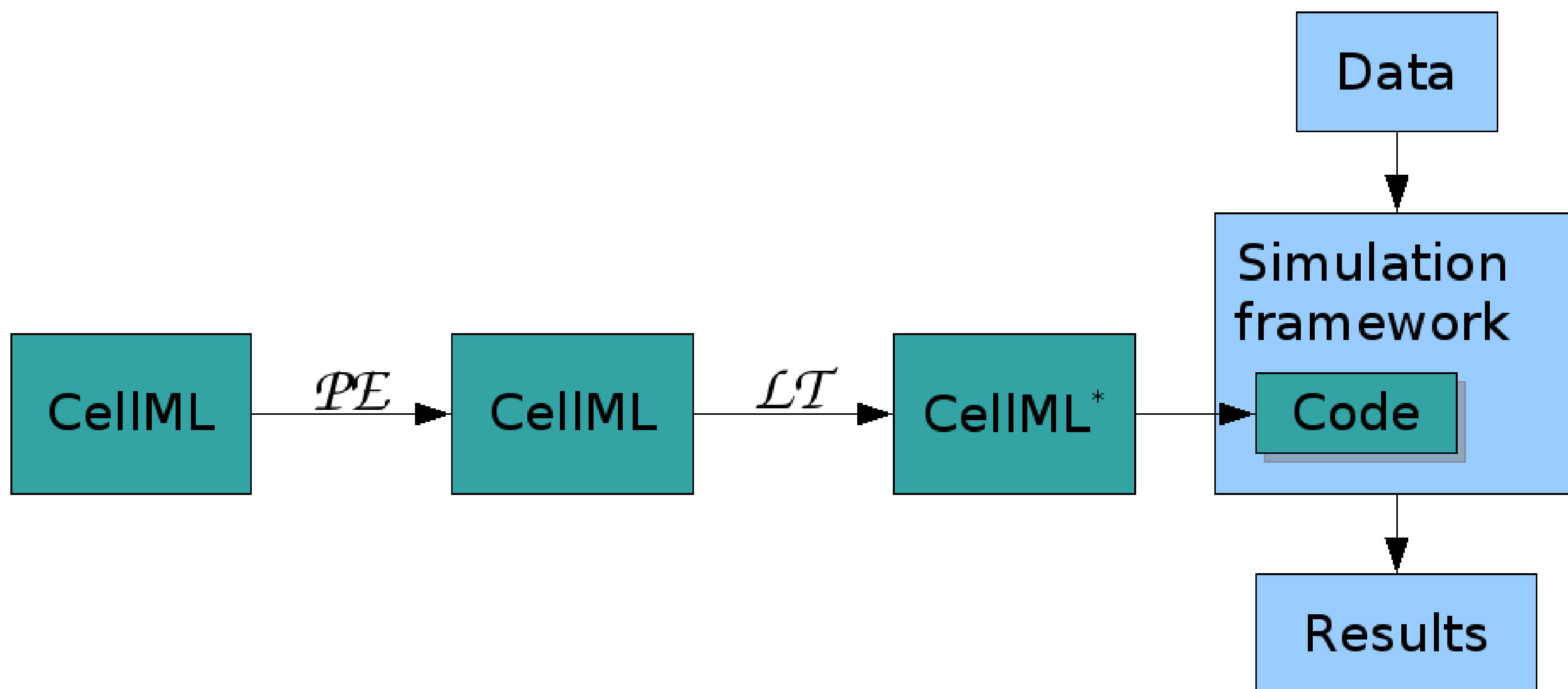
Automatic Lookup Tables

- Analysis of when to use tables can be automated
  - Check variables used to compute the expression
  - Check for occurrence of expensive functions

Automatic Lookup Tables

- We can allow variables other than V to occur in the expression
  - Constants, and variables whose values depend only on constants, are OK
  - Key point: is the value known when the table is generated?
  - This kind of analysis is done by partial evaluation
  - So do partial evaluation then lookup tables

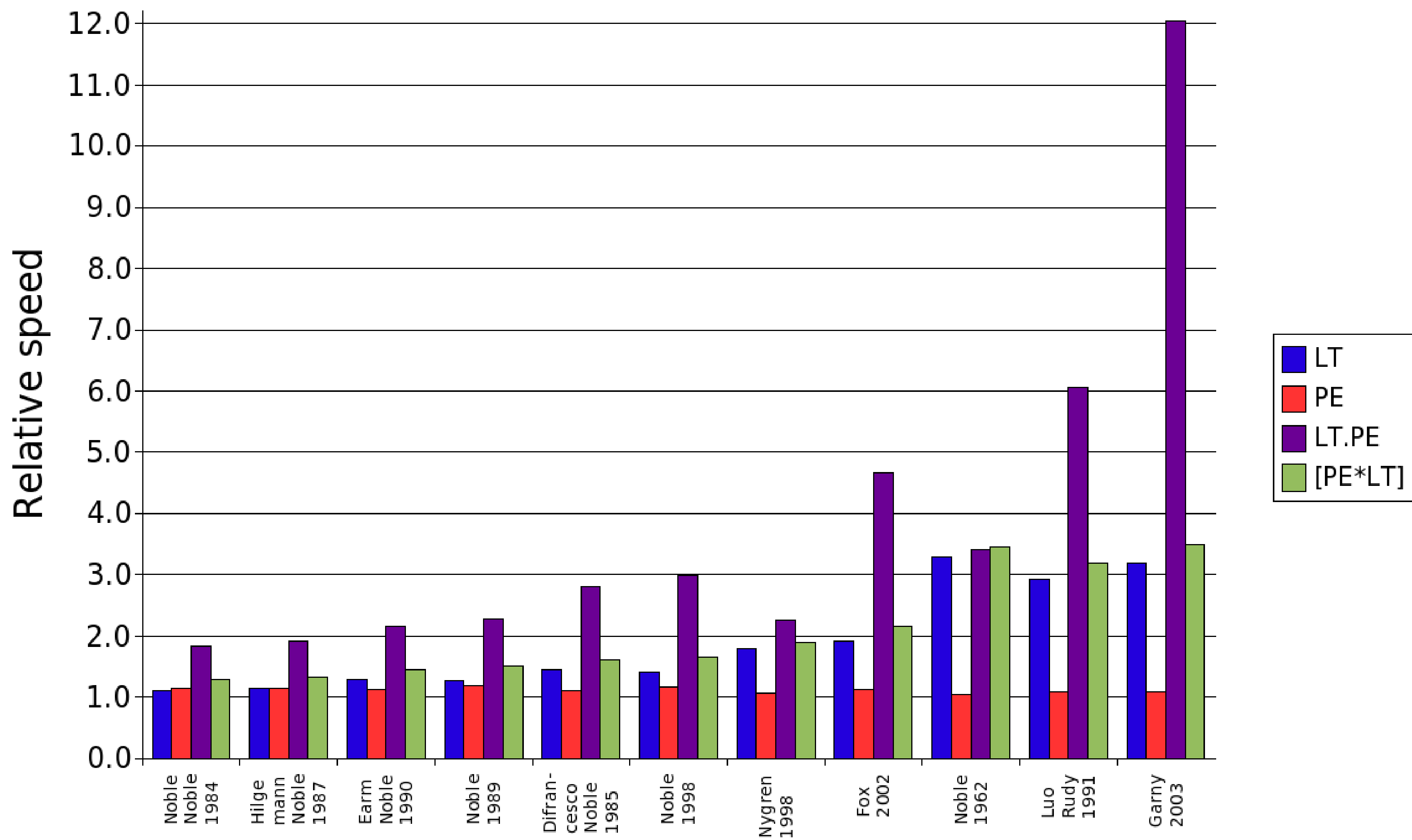
Optimisation Framework



$$\forall\, i \in inputs,\;\; c \in CellMLmodels$$

$$interp_{[ode]}(c, i) \approx interp_{[ode]}(\mathcal{LT} \cdot \mathcal{PE}\; c, i)$$

Experimental Results



Current and Future Work

- Developing a formal semantics for CellML, in order to prove correctness
- Applications to other types of (cell) model
- Extend Chaste to use adaptive techniques
- Compute a symbolic Jacobian without decoupling, for single cell use?

Model Correctness and Test Case Generation

How can we formally verify that the code implements the mathematical model?

- Studying the asymptotic behaviour of several cardiac cell models
- Developing formal specifications, including test cases, for each model
- Developing explicit logical and mathematical relationships between the models
- Refining the specifications to executable code

Overall goal is to improve model design, implementation and verification.

IntBioSim

The overall aim of this project is to explore and establish an integrated approach to computational systems biology spanning from the chemical to the subcellular level of simulations.

- Designed BioSimML to enable standard input to atomistic, course-grained atomistic and molecular simulations
- Developed a distributed environment that enables simulations to be initiated at a course-grained level, run until an equilibration state has been reached, spawn an atomistic simulation on a Grid
- Extending work to deal with data transformation between Molecular Dynamics and QM/MM

Proof Outline

- In order to complete the full proof of our methodology we need to break it down:
  - Partial Evaluation Correctness
  - Argument Complexity
  - Lookup Table Validity

CellML formal semantics

- An interpreter written in Haskell
- A lazy functional language style semantics
- Various simplifying assumptions, e.g.
  - All equations are explicit
  - The model is a system of first order ODEs
- The meaning of a model is given by evaluating the RHS of the ODE system in a suitable environment

Lookup Table Validity

$\forall\ i \in inputs,\ c \in CellMLmodels$

- $interp_{[ode]}(c, i) \approx interp_{[ode]}(\mathcal{LT}\ c, i)$

- Truncation error analysis isn't tight enough
- A posteriori error analysis is better
- ...but the stiff nature of cardiac cell models still raises issues

Argument Complexity

- In order to reason about how our transformations modify the evaluation of CellML models we introduce the argument complexity, $\mathcal{AC}$, function. This provides a rough estimate as to how long it takes to compute all operations in a tree.

  $\forall\ c \in CellMLmodels$

  $\mathcal{AC}\ c \geq \mathcal{AC}(\mathcal{PE}\ c) \quad \mathcal{AC}\ c \geq \mathcal{AC}(\mathcal{LT}\ c)$

- *Proof*: By induction on the rewrite rules of the partial evaluator and that of the lookup table generator. Each rule either preserves the tree or reduces its complexity.

Resultant Complexity

- We still haven't explained how $\mathcal{LT} \cdot \mathcal{PE}$ enables the dramatic improvement seen in our testing.
- We have:

  $\forall\, c \in \mathit{CellMLmodels} \quad \mathcal{AC}\ (c) \geq \mathcal{AC}(\mathcal{PE}\ c)$
- Depending on the model, the RHS:
  - will have the same replaceable subtrees as in the LHS
  - can have larger or more subtrees than the LHS

  therefore a larger proportion of subtrees will be removed and replaced by lookups.