Automated Code Generation for Deploying NESTML Neuron Models on SpiNNaker2

M. Weih[(1)] and C. Linssen[(2,3)] and B. Vogginger[(1)] and A. Morrison[(2,3)] and C. Mayr[(1)]

[(1)] TUD Dresden University of Technology [(2)] Simulation and Data Laboratory Neuroscience, Jülich Supercomputer Centre, Institute for Advanced Simulation, Jülich-Aachen Research Alliance, Forschungszentrum Jülich GmbH [(3)] Software Engineering, Department of Computer Science 3, RWTH Aachen University, Germany

Neuromorphic computing platforms promise scalable and energy-efficient simulation of large spiking neural networks, but deploying neuron models on specialized hardware often requires manual low-level implementation. This creates a barrier between high-level neuron model description languages and hardware-specific deployment. To surmount this barrier, the domain-specific modeling language NESTML [1] enables concise specification of neuron models in a user-friendly syntax, combined with automated code generation for specific simulation platforms, thereby maintaining high performance. In this work, we extend the NESTML code generation pipeline to support the neuromorphic SpiNNaker2 [2] platform, enabling automatic translation of neuron models into model-specific C-code implementations and Python-code interface classes. The approach introduces an intermediate transformation stage that eliminates physical unit annotations and converts expressions into dimensionless numerical form suitable for embedded execution. In addition to analytic integration, two numerical integration schemes are implemented to evaluate solver trade-offs on the platform: We provide a Forward Euler method solver with low computational cost and a fourth-order Runge-Kutta (RK4) solver providing improved numerical accuracy. Generated implementations are validated against reference simulations in NEST Simulator [3] using leaky integrate-and-fire and Izhikevich neuron models.
The results show that generated implementations preserve expected firing statistics and qualitative network dynamics when appropriate numerical parameters are chosen, based on membrane potential trajectories, spike timing accuracy, and numerical stability across different integration step sizes. Performance measurements indicate that automatically generated code is fractionally slower than hand-optimized implementations, but significantly reduces development effort and enables rapid deployment on SpiNNaker2. In conclusion, SpiNNaker2 support for NESTML is a key enabler in the transition towards power-efficient neuromorphic computing.

Figure 1: Workflow for formalising and simulating neuron models with NESTML and PySpiNNaker2: A mathematical model describing neuronal dynamics is formulated in the NESTML syntax by the user, after which automatic translation to C- and Python-code makes the model available for deploying SNN simulations on SpiNNaker2.

[1] C. Linssen et al., Front. Neuroinform. Volume 19:1544143, 2025

[2] H. Gonzalez et al., Machine Learning with New Compute Paradigms Conference, 2023

[3] M. Diesmann and M.-O. Gewaltig, Forschung und wisschenschaftliches Rechnen, Beiträge zum Heinz-Billing-Preis, Volume 58, Pages 43-70, 2001