Techniques for Automatic Parallelization and Optimization of Biological Simulations from *insilicoIDE*

Eric Heien [1], Yoshiyuki Asai [3], Taishin Nomura [2,3], Kenichi Hagihara [1,3]

1: Graduate School of Information Science and Technology, Osaka University
2: Graduate School of Engineering Science, Osaka University
3: The Center for Advanced Medical Engineering and Informatics, Osaka University

in Silico IDE

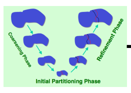
As part of the *in silico* medicine initiative, the *insilicoIDE* program is being developed. This program allows scientists to create models of biological systems and perform model based simulations. The long term goal is to database and simulate large scale biological models.

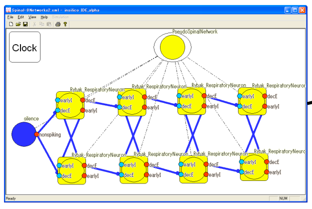
These models may be on the order of thousands or millions of components (modules), which can require hours to simulate. The focus of this research is investigating methods for decreasing execution time through parallel execution. The intended environment is an MPI enabled cluster of networked computers or multi-core processor.

In this research, we used a model of a spinal motor neural network based on the brain stem respiratory neural network model by Rybak.

Simulation Generation

METIS Graph Partitioner





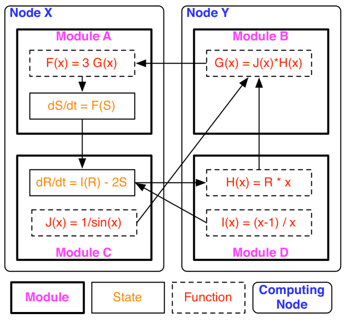
insilicoIDE with Spinal-8 Model

Message Passing Interface

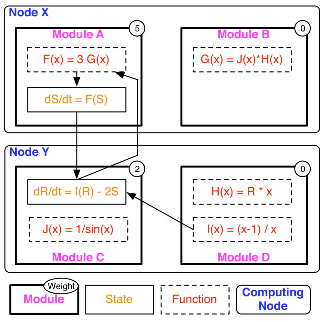
```
....
//! High-threshold calcium channel
class CCa_L_Channel_16 : public CModule {
public:
		///// module states /////
		//! channel dynamics of activation.  Unit : ratio
		double m;
		///// module parameters /////
		//! maximal conductance of the channel.  Unit : uS
		double g_;
		///// module functions /////
		//! stady state value of m  Unit : ratio
		double m_inf(const double &V) { return (1.6/(1 + exp(-0.072*(V - 5))))/(1.6/(1 + exp(-0.072*(V - 5))) + 0.02*(V - 1.31)/
(exp((V - 1.31)/5.36) -1)); }
		//! time constant of m  Unit : ms
		double t_m(const double &V) {return 1/(1.6/(1+exp(-0.072*(V - 5)))+0.02*(V-1.31)/(exp((V-1.31)/5.36)-1)); }
		//! Conductance of the Channel  Unit : uS
		double g(const double &g_ , const double &m) { return g_ * (m*m); }
		/////////////////////////////
		CCa_L_Channel_16( const double &m_)  : g_(0.0015) { m = m_; }
		~CCa_L_Channel_16(){};
		void integration(vector<double*> &vec){ m += *vec[11]; }
		void updateStates(const double &time,vector<double*> &vec,vector<double*> &results) {
				*results[11] = dt * (((*functionVec[2180]) - (*vec[11])) / (*functionVec[2175])); }
		void immediateStates(const double &time,vector<double*> &vec) { }
};
```

Automatically Generated C++ Code

Model Parallelization Techniques



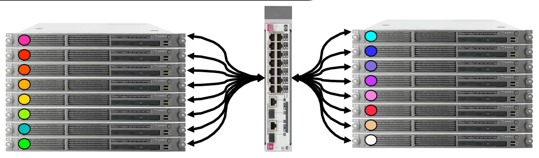
Initial Model Graph



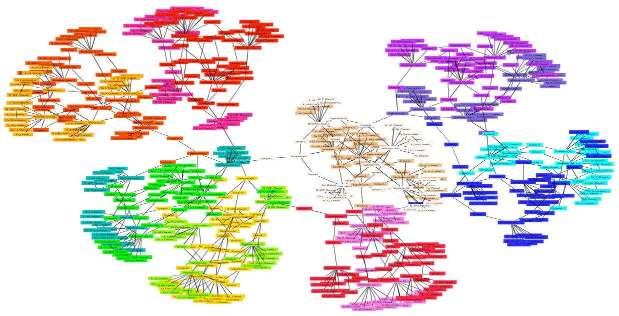
Simplified Model Graph

Models may be highly complex with many dependencies between modules. In the example above, the initial model has states and functions which depend on each other for calculation. However, because of this complexity the resulting parallel simulation can require multiple communication steps between compute nodes (i.e. the result of J must be sent from node X to Y, then the result of G must be sent back from Y to X). To reduce communication, the model is simplified as shown in the above right. Only state dependencies are retained, and function dependencies are implicitly recorded as calculation weights in modules. Combined with redundant function calculation, this allows fast parallel simulations with single communication phases. This also allows a clean break between computation and communication, which improves overall speed.

Automatic Parallelization

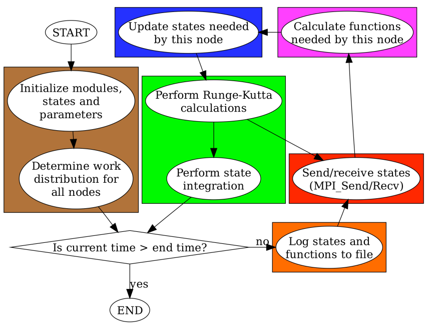


Test Cluster Setup
HP ProLiant G2 with 2.8 GHz Xeon x 32

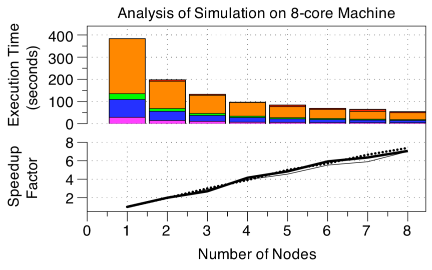


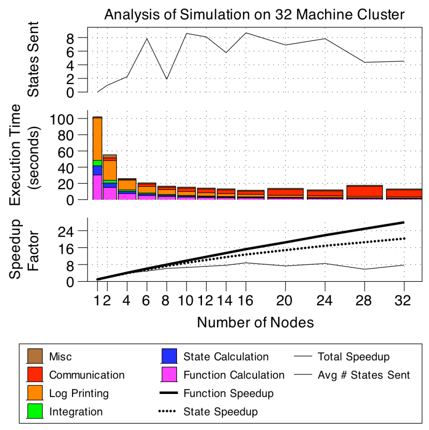
Graph showing the relations between modules for the Spinal-8 network and the mapping for a 16 node cluster. Colors indicate which node a module is mapped to, and edges represent state dependencies between modules. Creation of the graph and division among cluster nodes was accomplished in less than 0.02 seconds using the METIS serial graph partitioning library function **METIS_PartGraphKWay().**

Parallel Execution Results


START
Update states needed by this node
Calculate functions needed by this node
Initialize modules, states and parameters
Perform Runge-Kutta calculations
Determine work distribution for all nodes
Perform state integration
Send/receive states (MPI_Send/Recv)
Is current time > end time?
no
Log states and functions to file
yes
END

The program flow for a parallel insilicoIDE simulation using the Runge-Kutta approximation to solve ordinary differential equations.





Graphs showing the breakdown of time spent in each part of the simulation. Coloring corresponds to the program flow graph (left). As node count increases, communication grows. Also, since this model is of 8 neurons with few interconnections, node counts of 4, 8 and 16 perform well.