Neural Networks 47 (2013) 103–111

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2013 Special Issue

Realtime cerebellum: A large-scale spiking network model of the cerebellum that runs in realtime using a graphics processing unit

Tadashi Yamazaki [a,*], Jun Igarashi [b]

[a] RIKEN Brain Science Institute, 2-1 Hirosawa, Wako, Saitama 351-0198, Japan

[b] Computational Science Research Program, RIKEN, 2-1 Hirosawa, Wako, Saitama 351-0198, Japan

ARTICLE INFO

Keywords:
Graphics processing unit
Realtime simulation
Spiking network model
Cerebellar microcomplex
Reservoir computing
Robotics

ABSTRACT

The cerebellum plays an essential role in adaptive motor control. Once we are able to build a cerebellar model that runs in realtime, which means that a computer simulation of 1 s in the simulated world completes within 1 s in the real world, the cerebellar model could be used as a realtime adaptive neural controller for physical hardware such as humanoid robots. In this paper, we introduce "Realtime Cerebellum (RC)", a new implementation of our large-scale spiking network model of the cerebellum, which was originally built to study cerebellar mechanisms for simultaneous gain and timing control and acted as a general-purpose supervised learning machine of spatiotemporal information known as reservoir computing, on a graphics processing unit (GPU). Owing to the massive parallel computing capability of a GPU, RC runs in realtime, while reproducing qualitatively the same simulation results of the Pavlovian delay eyeblink conditioning with the previous version. RC is adopted as a realtime adaptive controller of a humanoid robot, which is instructed to learn a proper timing to swing a bat to hit a flying ball online. These results suggest that RC provides a means to apply the computational power of the cerebellum as a versatile supervised learning machine towards engineering applications.

1. Introduction

The cerebellum plays an essential role in motor learning and control. In an engineering term, the cerebellum is thought as an "adaptive control device", which observes the status of body parts by sensors continuously and calibrates the movement online to achieve fast and smooth motor control. The ability of adaptive control is unique to living organisms, and is expected its engineering applications such as humanoid robot control for flexible movements. Several attempts have been made to date. Feedback-error learning is a system-level approach to adopt the adaptive control ability for humanoid robots (Kawato & Gomi, 1992; Miyamoto, Kawato, Setoyama, & Suzuki, 1988; Shibata & Shaal, 2001). In those studies, a three-layer perceptron with rate-coding neurons was employed as a model of the cerebellum. Other studies have built spiking network models, and adopted to control a navigation robot that avoids hitting at the wall (Hofstötter, Mintz, & Verschure, 2002) and a robot arm with 2 joints to perform reaching tasks (Carrillo, Ros, Boucheny, & Coenen, 2008). Those models, however, focus primarily on engineering applications. It remains unknown whether they reproduce experimental results of, for example, Pavlovian delay eyeblink conditioning and gain adaptation of vestibulo-ocular reflex. Moreover, for the sake of computational time, the network size of those models is relatively smaller than the other cerebellar models which aim to reproduce experimental results.

On the other hand, we have built a large-scale spiking network model of the cerebellum, which is composed of more than 100,000 spiking neuron units with realistic parameters. The model has been demonstrated to reproduce experimental results of Pavlovian delay eyeblink conditioning (Yamazaki & Tanaka, 2007b) and gain adaptation of optokinetic response eye movements (Yamazaki & Nagao, 2012), suggesting that our cerebellar model can learn and control gain and timing information adaptively. Our cerebellar model could be adopted to such real-world applications as well, if the computer simulation is made in real time.

A graphics processing unit (GPU) is hardware designed and optimized for graphics, video, and visual computing in 2D and 3D (Patterson & Hennessy, 2011). The architecture consists of two components, one for graphics and the other for numerical calculation, which turns a GPU into a programmable graphics processor as well as a scalable parallel computational platform. CUDA (Compute Unified Device Architecture) (NVIDIA, 2011), a unified software development environment for GPUs, allows us to use a GPU as a highly-parallel, multi-threaded multiprocessor. GPUs have been already employed in the field of computational

* Correspondence to: Graduate School of Informatics and Engineering, The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan. Tel.: +81 42 443 5000.
E-mail address: nn12@neuralgorithm.org (T. Yamazaki).

0893-6080
doi:10.1016/j.neunet.2013.01.019

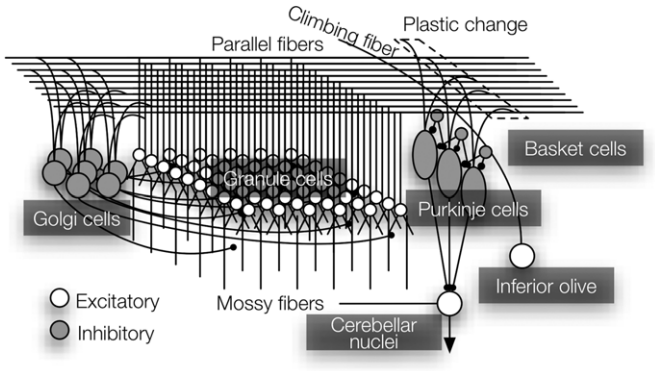

Fig. 1. Network structure of our cerebellar model.

neuroscience (Goodman & Brette, 2009; Igarashi, Shouno, Fukai, & Tsujino, 2011; Miikkulainen, Bednar, Choe, & Sirosh, 2005; Nageswaran, Dutt, Krichmar, Nicolau, & Veidenbaum, 2009). A study demonstrates realtime simulation of a detailed basal ganglia model for decision making (Igarashi et al., 2011).

In this study, we re-implement our cerebellar model on a GPU, so as to carry out the computer simulation in real time. Using some techniques on implementation, the new model, which we call "Realtime Cerebellum (RC)" can run in real time: a simulation of 1 s in the simulated world completes within 1 s in the real world. We carry out computer simulation of Pavlovian delay eyeblink conditioning, and confirm that RC reproduces qualitatively the same results with the previous model. We also adopt RC to hardware control to demonstrate the power of realtime computing and delay compensation in sensorimotor loop. We set up a robot experiment, in which a small humanoid robot is instructed to hit a flying ball thrown by a pitching machine by swinging a bat at hand. The robot gradually learns the correct timing by repetition of practice and finally succeeds to hit the ball.

2. Materials and methods

2.1. Overview of our cerebellar model

RC, the cerebellar model we implemented on a GPU in this study, is based on our previous models (Yamazaki & Nagao, 2012; Yamazaki & Tanaka, 2007b). Briefly, RC is composed of 102,400 granule cells, 1024 Golgi cells, 16 Purkinje cells, 16 basket cells, 1 inferior olive and 1 neuron in the cerebellar nucleus (Fig. 1). External inputs are fed by mossy fibers to granule cells and the nucleus. Granule cells excite Golgi cells, Purkinje cells and basket cells. In turn, Golgi cells and basket cells inhibit granule cells and Purkinje cells, respectively. All the 16 Purkinje cells inhibit the nuclear cell. Other external inputs are fed by climbing fibers to Purkinje cells. The final output of the network is generated by the nucleus. Granule cell–Purkinje cell synapses undergo plastic change (long-term depression and potentiation).

Neurons are modeled as conductance-based, leaky integrate-and-fire units.

$$C\frac{dV(t)}{dt} = -g_{\text{leak}}(V(t)-E_{\text{leak}}) - g_{\text{ex:AMPA}}(t)(V(t)-E_{\text{ex}}) - g_{\text{ex:NMDA}}(t)(V(t)-E_{\text{ex}}) - g_{\text{inh}}(t)(V(t)-E_{\text{inh}}) - g_{\text{ahp}}(t-\hat{t})(V(t)-E_{\text{ahp}}), \quad (1)$$

where $V(t)$ and C are the membrane potential at time t and the capacitance, respectively. The membrane potential is determined by five types of currents specified by the right-hand side of Eq. (1), namely, leak, alpha-amino-3-hydroxy-5-methyl-4-isoxazolepropionic acid receptor (AMPAR)-mediated, NMDAR-mediated and gamma-aminobutyric acid type A receptor ($GABA_AR$)-mediated currents, and the current for emulation of the after-hyperpolarization. For each type $c \in \{\text{leak, ex:AMPA, ex:NMDA, inh, ahp}\}$, the current at a given time is calculated with the conductance g_c and reversal potential E_c. The conductance is calculated by the convolution of the alpha function $\alpha(t)$ and the spike event $\delta_j(t)$ of presynaptic neuron j at time t as follows:

$$g_c(t) = \sum_j w_j \int_{-\infty}^{t} \bar{g}_c \alpha(t-s)\delta_j(s)ds, \quad (2)$$

where $\bar{g}_c$ represents the maximum conductance and w_j the synaptic weight from the presynaptic neuron j. The alpha functions are defined for each current and each neuron type with different time constants. When the membrane potential of a neuron exceeds the threshold θ, the neuron is supposed to elicit a spike, followed by the after-hyperpolarization that determines a refractory period. The conductance for the after-hyperpolarization is given by

$$g_{\text{ahp}}(t-\hat{t}) = \exp(-(t-\hat{t})/\tau_{\text{ahp}}), \quad (3)$$

where τ_{ahp} represents the time constant of the after-hyperpolarization and $\hat{t}$ is the last firing time of the neuron. Detailed parameters are described in our previous papers (Yamazaki & Nagao, 2012; Yamazaki & Tanaka, 2007b).

For each connection between two neurons, a constant called a synaptic weight is assigned; detailed values are shown in our previous paper (Yamazaki & Nagao, 2012). These synaptic weights do not change during the whole computer simulation, except those between parallel fibers (granule cell axons) and Purkinje cells. When a parallel fiber is solely activated, the weight increases slightly, whereas when the activation is paired with that of a climbing fiber, the weight decreases slightly. This bidirectional change models long-term potentiation (LTP) (Coesmans, Weber, De Zeeuw, & Hansel, 2004; Lev-Ram, Mehta, Kleinfeld, & Tsien, 2003) and long-term depression (LTD) (Ito, 2001, 2002), respectively. The equation for LTP/LTD is shown in our previous paper (Yamazaki & Nagao, 2012; Yamazaki & Tanaka, 2007b).

We have hypothesized that, the recurrent inhibitory network composed of granule and Golgi cells generates various temporally-fluctuating spike patterns among granule cells in response to mossy fiber signals (Yamazaki & Tanaka, 2005). Because different granule cells exhibit different temporal patterns, the population of active granule cells changes gradually in time, indicating that there is a one-to-one correspondence between a granule-cell population and a time step from the onset of mossy fiber signals. Therefore, the temporal evolution of active granule-cell populations can represent the passage of time from the mossy fiber signal onset. To study how the spike patterns of granule cells evolve over time, we define two indices. Let $z_i(t)$ be the population average activity of a granule-cell cluster i, which is defined by a set of nearby granule cells sharing the same inhibitory inputs from Golgi cells via glomeruli (see Yamazaki & Nagao, 2012; Yamazaki & Tanaka, 2007b for details):

$$z_i(t) = \frac{1}{\tau_{\text{PKJ}}} \sum_{s=0}^{t} \exp\left(-(t-s)/\tau_{\text{PKJ}}\right) \left(\frac{1}{N_{\text{granule per cluster}}} \sum_j \delta_j(s) \right), \quad (4)$$

where $\delta_j(s)$ represents the spike elicited by model granule cell j in the cluster, $N_{\text{granule per cluster}}$ is the number of granule cells in a cluster (namely, 100), and τ_{PKJ} is a decay time constant of AMPAR-mediated EPSPs at Purkinje cells, which was set at 8.3 ms. We define the autocorrelation of the activity pattern at times t and $t+\tau$ as follows:

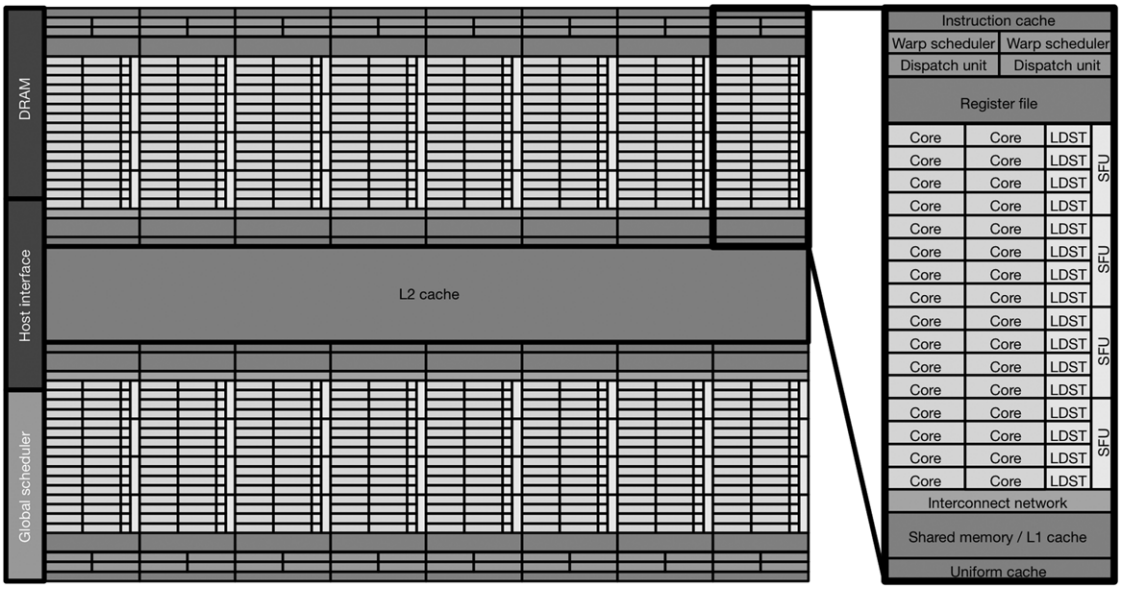

Fig. 2. Architecture of GTX580. 16 SMs are positioned around a common L2 cache. Each SM is a vertical rectangular strip that contains scheduler and dispatch, execution units, and register file and L1 cache. Abbreviations: LDST, load/store unit; SFU, special function unit.

$$C(t, t+\tau) = \frac{\sum_i z_i(t) z_i(t+\tau)}{\sqrt{\sum_i z_i^2(t)}\sqrt{\sum_i z_i^2(t+\tau)}}. \quad (5)$$

The numerator represents the inner product of activity pattern vectors at time t and $t+\tau$, and the denominator normalizes the vector lengths. Because $z_i(t)$ takes only positive values, the correlation takes a value between 0 and 1. It would be 1 if the activity pattern vectors at time t and $t+\tau$ are identical, and it would be 0 if the vectors are orthogonal, indicating that the activity patterns have no overlap. We now define the similarity index $S(\tau)$ as follows:

$$S(\tau) = \frac{1}{T}\sum_{t=0}^{T} C(t, t+\tau), \quad (6)$$

where T represents the duration of the mossy fiber signals. This is the average of Eq. (5) with respect to t. This index represents how two activity pattern vectors separated by $|\tau|$ are correlated, on average. If the similarity index decreased as $|\tau|$ increased, it indicates that an activity pattern is evolved with time into uncorrelated patterns. We also define the reproducibility index $R(t)$ as follows:

$$R(t) = \frac{\sum_i z_i^{(1)}(t) z_i^{(2)}(t)}{\sqrt{\sum_i z_i^{(1)2}(t)}\sqrt{\sum_i z_i^{(2)2}(t)}}. \quad (7)$$

$z_i^{(1)}(t)$ and $z_i^{(2)}(t)$ are activity patterns of granule-cell cluster i at time t for two successive trials. The reproducibility index represents how two temporal activity patterns for different trials differ from each other as time elapses.

2.2. Architecture of a GPU

The GPU we used in this study is GeForce GTX 580 (NVIDIA). Fig. 2 illustrates the architecture (NVIDIA, 2009). GTX 580 has 16 Streaming Multiprocessors (SMs), each is composed of 32 execution units called "cores", 16 load/store units, 4 special function units, dual warp schedulers and dispatch units, 4 kB registers, and 64 kB configurable shared memory and L1 cache. Each core has a pipelined integer arithmetic logic unit and floating point unit. GTX 580 has 64 kB L2 cache, 1.5 GB GDDR5 DRAM and a global scheduler as well, and is connected to a host PC via PCI-Express. In other words, GTX 580 is composed of 512 processors and very fast 1.5GB memory. Floating-point arithmetic is performed in single precision.

2.3. Parallel implementation on a GPU

We used CUDA, a unified software development environment for GPU programming provided by NVIDIA (NVIDIA, 2011), for the implementation.

We illustrate how to translate a C code for a CPU to the identical one for a GPU using CUDA. A standard program of a neural network model would be like the one as in Fig. 3(A). The code involves two for-loops with respect to time (`t`) up to `T` and neurons (`i`) up to `N`. For each time step and for each neuron, the neuron state is updated by function `compute(t, i)`. The GPU version is shown in Fig. 3(B). The inner loop with respect to `i` is replaced by a function call `kernel<<N/K, K>>(t)`. That is, for each time step, function `kernel` is invoked. The notation "`<<a, b>>`" is a directive of CUDA, which commands to assign `a` × `b` threads to execute `kernel` in parallel, where a thread is a computing entity on a GPU, `a` and `b` denote the numbers of "grids" and "thread blocks", respectively. Grids and thread blocks are the strategy of CUDA, which groups all invoked threads into multiple sets so that multiple threads are executed concurrently and efficiently on multiple SMs. In the example code, `N/K` × `K` = `N` threads are involved for parallel execution of `kernel`. Function `kernel` is declared with the prefix `__global__`, which is a directive of CUDA showing that the function is executed on a GPU. In the function, the value of variable `i` is set using two parameters `blockIdx.x` and `threadIdx.x`, which are unique indices assigned for each thread. In this example, it holds $0 \le$ `blockIdx.x` $< N/K$, and $0 \le$ `threadIdx.x` $< K$, so that the value of `i` varies between 0 and `N` − 1. Hence, `N` threads calculate function `compute(t, i)` using unique indices `i` in parallel.

By this translation, the total number of loops decreases from `T` × `N` to `T`, suggesting maximally `N`-times speed up. This is effective when a model contains a large number of neurons. RC, composed of more than 0.1 million neurons, is such a case.

A
```
int main(void)
{
  int t, i;
  :
  for(t = 0; t < T; t++){
    for(i = 0; i < N; i++){
      compute(t, i);
    }
  }
  :
  return 0;
}
```

B
```
__global__ void kernel(int t)
{
  int i = blockIdx.x*K + threadIdx.x;
  compute(t, i);
}
int main(void)
{
  int t, i;
  :
  for(t = 0; t < T; t++){
    kernel<<N/K, K>>(t);
  }
  :
  return 0;
}
```

Fig. 3. Example code of a neural network model in C. (A) CPU version. (B) GPU version. `__global__`, `BlockIdx.x`, `threadIdx.x` and `<<x, y>>` are directives of CUDA.

A
```
for(i = 0; i < N_post; i++){
  for(j = 0; j < N_pre; j++){
    g[i] += w[i][j]*conv[j];
  }
}
```

B
```
__global__ void kernel(void)
{
  // i: postsyn. neuron
  // j: presyn. neuron
  int j = BlockIdx.x*K + ThreadIdx.x;
  __shared__ float shared_g[N_post];

  for(i = 0; i < N_post; i++){
    atomicAdd(&shared_g[i],
              w[i][j]*conv[j]);
  }
  __syncthreads();

  if (threadIdx.x == 0){
    for(i = 0; i < N_post; i++){
      atomicAdd(&g[i], shared_g[i]);
    }
  }
}
```

Fig. 4. Example code of calculating conductances. (A) CPU version. (B) GPU version. `__global__`, `__shared__`, `atomicAdd(x, y)` and `__syncthreads()` are directives of CUDA.

We also illustrate how to calculate conductances efficiently on a GPU. Fig. 4(A) is the code snippet for a CPU. The outer and inner loops enumerate the postsynaptic and presynaptic neurons `i`, `j` up to `N_post` and `N_pre`, respectively. We assume that $j \gg i$, namely, a large number of presynaptic neurons are converged to one postsynaptic neuron. The conductance `g[i]` of postsynaptic neuron `i` is the weighted sum of the value of convolution in Eq. (2) denoted by `conv[j]` with synaptic weights `w[i][j]`. Here, we assume that the convolution is calculated in advance. The GPU version is shown in Fig. 4(B). We split the inner loop with respect to `j` to `N_pre` threads, and each thread enumerates the loop with respect to `i` individually. First, a set of threads in the same thread block allocates an array `shared_g[N_post]` on shared memory. Shared memory is a special memory area, so that `shared_g` becomes accessible from these threads. Threads in a thread block calculate `w[i][j]*conv[j]` and add to `shared_g[i]`. Here, because all threads race the same memory bank to write simultaneously, the addition must be made exclusively. We use `atomicAdd(&a, b)`, a directive of CUDA for atomic add operation (i.e., $a+=b$). Next, these threads invoke `__syncthreads()`, which ensures that all threads in the same thread block completes the above atomic operation. Then, the representative thread, who has the thread index 0 in each thread block, adds the value of `shared_g[i]` to `g[i]` using the same atomic add operation. By this translation, the total number of loops decreases from `N_post` $\times$ `N_pre` to $2\times$ `N_post`, suggesting maximally `N_pre`/2-times speed up. This is effective when a large number of presynaptic neurons innervate to the same postsynaptic neuron. Again, RC is such a case, because about 30,000 granule cells contact to the same Purkinje cell in the model.

We note that, these techniques do not directly contribute for `N`-times or `N_pre`/2-times speed up of overall simulations. There are several reasons. First, the number of cores on GTX 580 is limited to 512. Second, the number of threads running in parallel is limited to `K`. If `N` or `N_pre`/2 are larger than these numbers, the speed up is regulated. Third, the most time consuming operations are not arithmetic operations but memory operations. In particular, memory transfer between CPUs and GPUs can easily slow down the overall simulation.

2.4. Simulation of Pavlovian delay eyeblink conditioning

We adopt RC to simulation of Pavlovian delay eyeblink conditioning (Christian & Thompson, 2003; Mauk & Donegan, 1997), to demonstrate the ability of timing learning.

In Pavlovian delay eyeblink conditioning (Fig. 5), an animal receives repeated paired presentations of a tone (conditioned stimulus; CS) and an airpuff (unconditioned stimulus; US). The animal becomes conditioned to close eyes slightly earlier than the onset of the airpuff (conditioned response; CR) in response to the tone, indicating that the animal learns the proper timing to close eyes to protect from the airpuff. Lesion and pharmacological studies have demonstrated that the timing information is acquired by the cerebellar cortex (Garcia & Mauk, 1998; Perrett, Ruiz, & Mauk, 1993). The tone and airpuff information is fed to the cerebellar cortex via mossy fibers and climbing fibers, respectively, whereas motor command for eyeblink is exerted by neurons in the cerebellar nucleus. Timing is expressed by the pause of Purkinje cells (Berthier & Moore, 1986; Jirenhed, Bengtsson, & Hesslow, 2007).

To simulate eyeblink conditioning experiments, we feed Poisson spikes of 30 spikes/s and a single spike to mossy fibers and climbing fibers as a CS and a US, respectively. The CS is sustained for 0.65 s, whereas the US is fed when the CS is terminated. Therefore, the interstimulus interval is set at 0.65 s. On the other hand, the intertrial interval is set at 4 s. Differential equations are solved by the 2nd-order Runge–Kutta method with the fixed time step

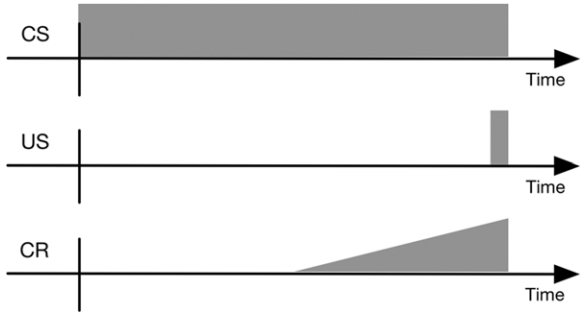

Fig. 5. Schematic of Pavlovian delay eyeblink conditioning. Abbreviations: CS, conditioned stimulus; US, unconditioned stimulus; CR, conditioned response; ISI, interstimulus interval.

of 1 ms. All floating-point arithmetic operations are performed in single-precision format. A linear congruential method is used as a pseudo random number generator.

2.5. Robot experiments

We developed a batting robot system to demonstrate that RC can learn timing online, which is analogous to Pavlovian delay eyeblink conditioning. The system is composed of a toy pitching machine (Toss-Mac, UNIX, Fig. 6(A)), a hobby—use small humanoid robot (KHR-3HV, Kondo-Kagaku, Fig. 6(B)), and a fence behind the robot (Fig. 6(B)). The pitching machine throws a small (diameter 4 cm) plastic ball to a robot (CS) one at a time. The robot has a bat at hand which is a plastic round fan, and tries to hit the flying ball (Fig. 6(D)). If the robot fails, the ball hits the fence behind the robot (US). The task of the robot is to learn the correct timing to swing a bat to hit a flying ball (CR). The distance between the pitching machine and the fence is about 0.35 m (Fig. 6(C)), and the ball thrown takes about 0.65 s to hit the fence, so that the ball speed is estimated as approximately 0.54 m/s. The time interval between pitching is approximately 4 s.

Two accelerometers (RAS-2, Kondo-Kagaku) are attached at the outlet of the pitching machine and the fence, respectively, so that they can detect the onset of ball throwing and the failure of ball hitting by the robot. Thus, these timing information are mechanically detected. No visual or auditory information are taken into account. A PC receives signals from those accelerometers via a physical computing device (Arduino Uno) and sends motor command to a humanoid robot to swing a bat. The PC also sends the event information of ball throwing and ball hitting on the fence as a CS and a US to another PC with a GPU. The latter PC carries out computer simulation of RC and emits the signal for motor command to the former PC that triggers a CR (i.e., batting). By the trigger of a CR, the robot simply plays a preprogrammed, static swing motion. Dynamic characteristics of the robot are not taken into account. The swing motion is induced only by the trigger of a CR. The US signal is used only for learning. The entire control program is written in Ruby.

Because of the toy-grade hardware, signal transmission on slow serial connections and programming issues, there is a delay in the sensorimotor loop, which is roughly estimated about less than 0.1 s in total with a large trial-by-trial variability.

3. Results

3.1. Realtime simulation of Pavlovian delay eyeblink conditioning

We carried out computer simulation of Pavlovian delay eyeblink conditioning to demonstrate that RC generates qualitatively the same results with our previous version, and that the simulation is carried out in realtime.

3.2. Granule cells exhibit temporally-fluctuating spike patterns for passage of time representation

First, we investigated the spike patterns of granule and Golgi cells. Fig. 7(A) and (B) respectively show the spike patterns of 1000 out of 102,400 granule cells and all 1000 out of 1024 Golgi

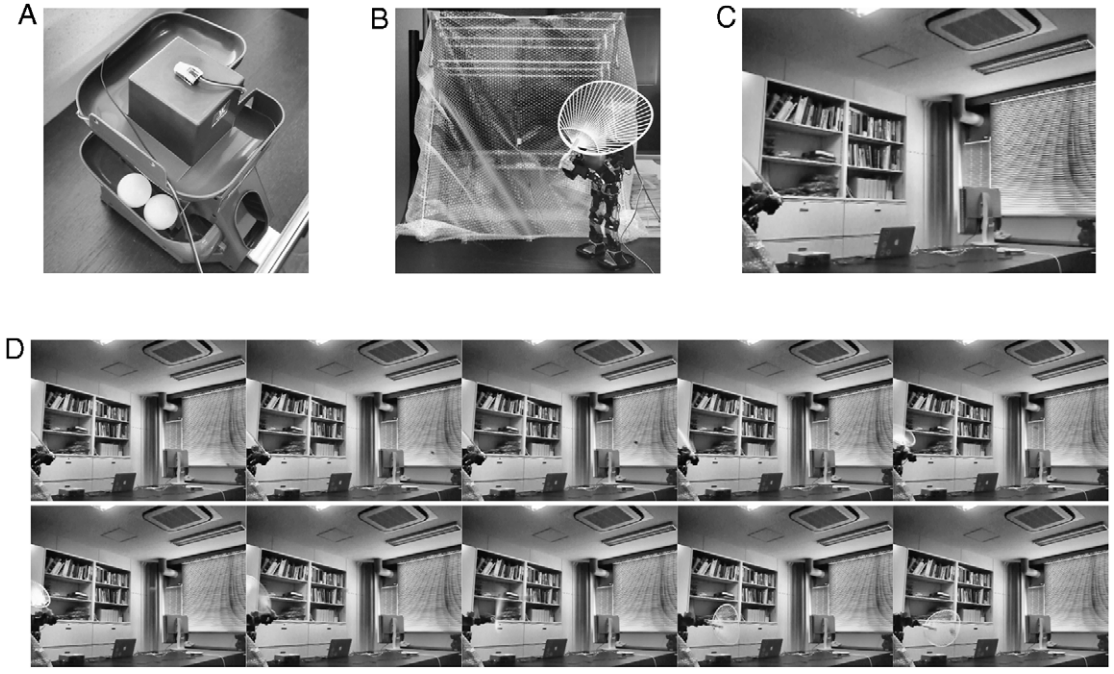

Fig. 6. Robot experiments of ball batting. (A–C) Facilities used in the experiments: (A) pitching machine, (B) humanoid robot and fence behind it, (C) setup overview. On the pitching machine and the fence, an accelerometer is attached so that the onset of ball throwing and the failure of batting are detected. (D) Robot motion for successful ball batting.

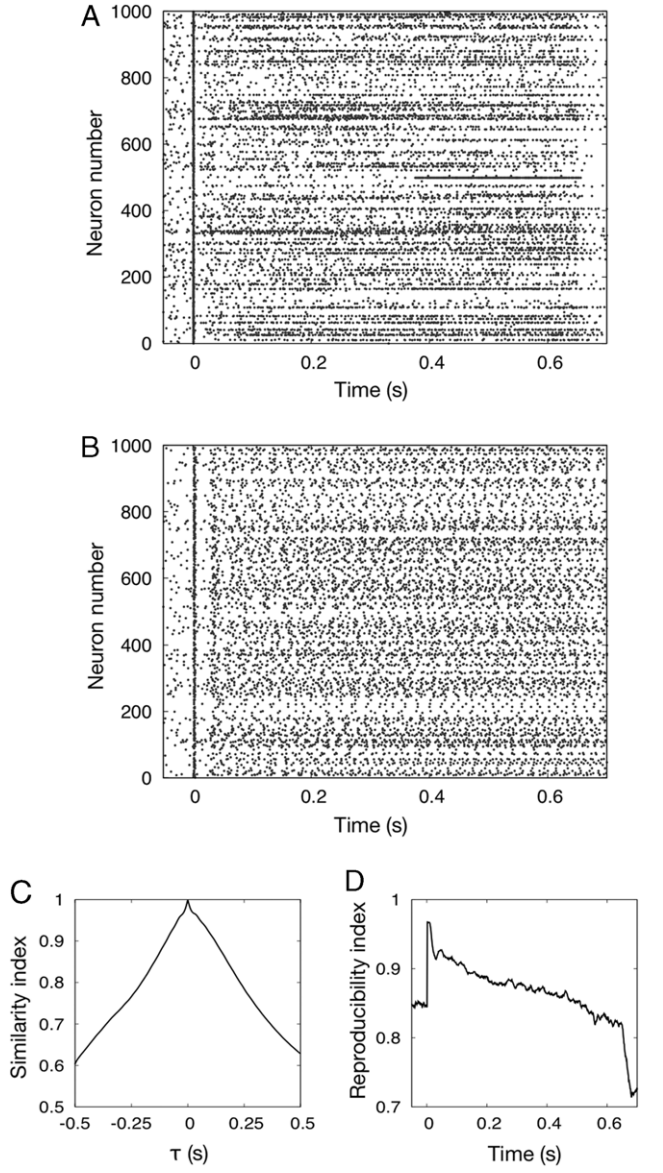

Fig. 7. Spike patterns of granule cells and Golgi cells in response to the CS. (A) Granule cells underwent random alternation between burst and silent modes. Different granule cells exhibit different temporal profiles of their spike patterns. (B) Golgi cells also underwent similar repetitions of spike bursts, although they elicit spikes more regularly than granule cells. Abscissa and ordinate represent time from the CS onset and neuron number, respectively. Only 1000 neurons were plotted. (C) Similarity index $S(\tau)$ for the spike patterns of granule cells. The index takes the value of 1 at $\tau = 0$ and decreases monotonically as $|\tau|$ increases, suggesting that the population of active granule cells changes gradually in time. (D) Reproducibility index $R(t)$ for the spike patterns of granule cells. The index takes the largest value at $t = 0$ and decreases almost linearly as t increases, suggesting that sequences of the active granule-cell populations for subsequent trials are initially quite similar but gradually diverge as time elapses.

cells in response to the simulated CS that sustains for 0.65 s. The granule cells were chosen according to the following criteria. In the present model, 102,400 granule cells are arranged on a two-dimensional square grid. The 320×320 granule cells are functionally rearranged into 32×32 granule-cell "clusters" by grouping nearby 10×10 granule cells, which are assumed to receive inputs from the same mossy fibers and Golgi cells (Yamazaki & Tanaka, 2007b). Because granule cells in the same cluster exhibit similar temporal spike patterns, we could pick up 1 cell arbitrary as the representative cell from each cluster. In Fig. 7(A), we depicted the raster plot of 1000 out of 1024 representative cells. Granule cells undergo random alternation between burst and silent modes. Different granule cells exhibit different temporal patterns, so that the population of active granule cells gradually changes in time. Golgi cells also undergo similar random alternation, although they elicit spikes more regularly than granule cells. This random temporal patterns emerge from the dynamics of the random recurrent inhibitory network composed of granule and Golgi cells, not from input noise (Honda, Yamazaki, Tanaka, Nagao, & Nishino, 2011; Yamazaki & Tanaka, 2007b). To study the recurrence of the sequence of active granule-cell populations, we calculated the similarity index $S(\tau)$ for the spike patterns of granule cells according to Eq. (6), and plotted in Fig. 7(C). The value of $S(\tau)$ is 1 at $\tau = 0$ because of the trivial identity. The value monotonically decreases as $|\tau|$ increases, indicating that the sequence of active granule-cell populations is non-recurrent. The non-recurrent sequence represents the passage of time from the CS onset, by assigning one population to one time step (Yamazaki & Tanaka, 2009). We also calculated the reproducibility index $R(t)$ as in Eq. (7) to study whether the same sequence is reproducible across trials and plotted in Fig. 7(D). The value of $R(t)$ is the largest at $t = 0.008$ s and almost linearly decreases in time until the offset of the CS (0.65 s) with small fluctuations, suggesting that two sequences of active granule-cell populations for two successive trials are almost the same initially, and gradually diverges in time. These results are qualitatively the same with those reported previously by the same group (Yamazaki & Tanaka, 2007b), suggesting that the simulation results were reproduced by the new model.

3.3. Purkinje cells learn to pause for timing expression

We then investigated the spike patterns of Purkinje cells and a nuclear cell. Fig. 8(A) and (B) respectively represents spike patterns of a Purkinje cell and a nuclear cell during the simulated training of eyeblink conditioning. At the 1st trial, the Purkinje cell elicits spikes tonically throughout the CS presentation, so that the nuclear cell is inhibited strongly. At the 5th trial, the Purkinje cell shows a short "pauses" (Berthier & Moore, 1986; Jirenhed et al., 2007) around the US onset. The nuclear cell is disinhibited and elicits spikes mainly after the US onset. At the 10th trial, the Purkinje cell's pause starts earlier. The nuclear cell is timely released from the inhibition and starts to elicit spikes 0.45 s in advance of the US onset, which causes anticipatory CR that starts slightly earlier than the US. Again, these results are qualitatively the same with those reported previously (Yamazaki & Tanaka, 2007b), suggesting that the new model was able to reproduce the simulation results.

The present model contains 16 Purkinje cells, which converge to a single nuclear cell. The activity of the nuclear cell (Fig. 8(B)) is not controlled solely by the Purkinje cell shown in Fig. 8(A), but by the population average of these 16 Purkinje cells. Therefore, the nuclear cell's activity does not necessarily become mirror symmetric with the Purkinje cell's activity.

3.4. Comparison of computational time

We measured the computational time spent by the cerebellar model (either the previous CPU version or the present GPU version) during the simulation of eyeblink conditioning. We extended the duration of a CS to 1 s for clarity. The CPU version spent 93 s, on average, for the simulation of 1 s, whereas the GPU version 0.98 s (Fig. 9), indicating that we could achieve approximately 100-times speed up and realtime computer simulation.

3.5. Robot experiments for realtime control and online learning

Finally, we adopted RC to real hardware control. We developed a batting robot system analogous to the Pavlovian delay eyeblink

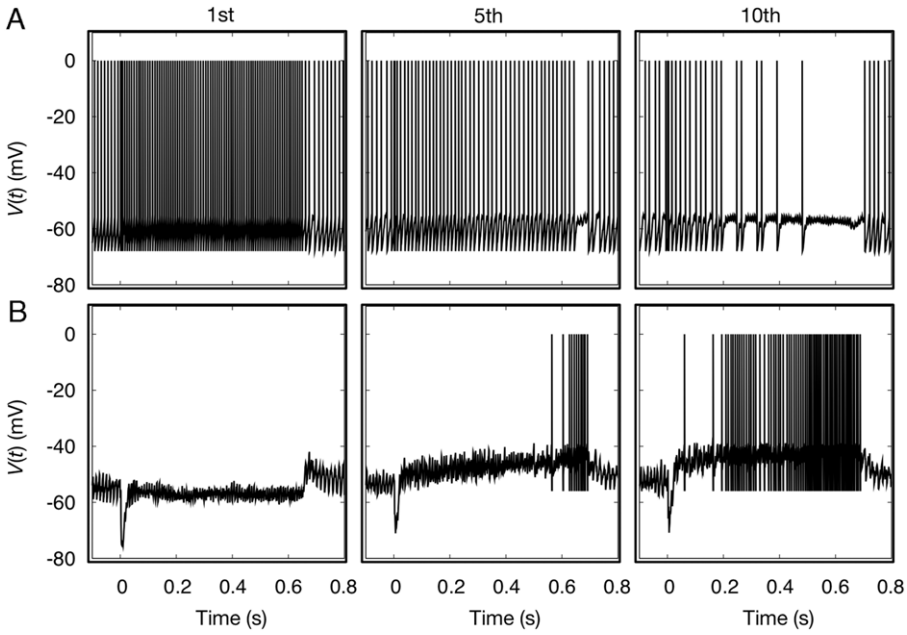

Fig. 8. Spike patterns of a Purkinje cell and a nuclear cell during the simulation of eyeblink conditioning. (A) The Purkinje cell gradually learns to pause in advance of the US onset (= 0.65 s). (B) The nuclear cell is timely released from the inhibition exerted by the Purkinje cells, and elicits spikes slightly earlier than the US onset, suggesting the occurrence of anticipatory CR. Abscissa and ordinate represent time from the CS onset and membrane potential, respectively.

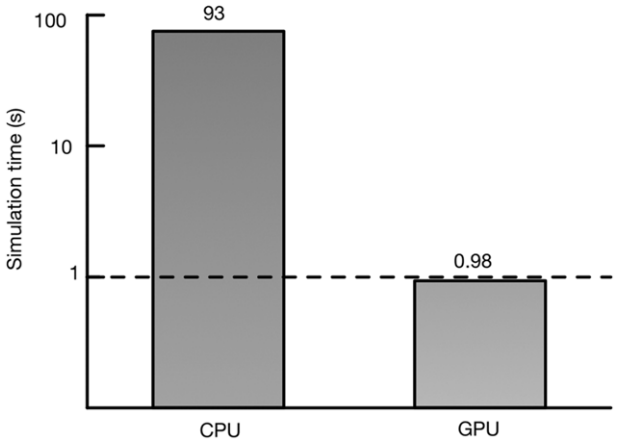

Fig. 9. Comparison of simulation time spent by the cerebellar models implemented on either a CPU or a GPU. Vertical axis plots the time in log scale.

conditioning, in which a humanoid robot learns the correct timing to swing a bat at hand to hit a flying ball thrown by a pitching machine. There is a slight difference between the simulated Pavlovian delay eyeblink conditioning and the robot experiments. In the simulation, information on CS and US are given instantly without delay to RC, which in turn issues CR instantly as well. On the other hand, in the robot experiment, the information of ball throwing detected by accelerometers is transmitted to RC with a certain delay, and servomotors start to move with a certain delay after RC emits the motor command. In this way, there is an inevitable delay within the hardware sensorimotor loop. The robot has to compensate the internal delay in the experiments.

In typical experiments, the robot swings the bat for the first time after about 5 trials. This first swing, however, delays significantly. The robot consumes 5–10 more trials to decrease the delay. This learning curve is almost the same with the simulation of eyeblink conditioning. After that, the robot succeeds to hit a ball for the first time (Fig. 6(D), supplementary video). This result suggests that, RC can be used as a realtime neural controller of a humanoid robot.

We also conducted experiments for extinction of the learned timing. After the success of ball hitting, we remove balls from the pitching machine and continue the training. Thus, the robot receives only the CS information detected by an accelerometer. We found that, the robot kept to swing a bat in response to the CS for 3 successive trials. After that, the robot stopped to swing the bat, suggesting extinction of the learned timing.

4. Discussion

In this study, we presented RC, a new implementation of our large-scale spiking network model of the cerebellum on a GPU, and demonstrated its realtime computer simulation and application to adaptive control of a humanoid robot.

4.1. Parallel neural network simulation on a GPU

In a typical neural network model, neuronal units are sparsely connected and their internal states (e.g. membrane potentials, conductances, and so on) are updated independently. This is an ideal case for parallel computing using a GPU. By rewriting a program of a neural network model to use a GPU, which can be made quite straightforward as described in Materials and Methods, we can instantly achieve 2–100-times speed up. Moreover, GPUs are affordable (300–2000 USD). These suggest that GPUs accelerate computer simulation of a neural network model as well as the advance of computational neuroscience.

There are several different ways of parallel computing. MPI (Message Passing Interface) is a library specification for message passing across multiple CPU cores on computer clusters (Pacheco, 1996). OpenMP is a set of APIs (application programming interfaces) that automatically parallelize a program across multiple CPU cores on a single computer (Chandra et al., 2000). These support parallel programming in a multicore environment. A consumer-grade multicore environment would be composed of tens of

workstations and contains hundreds of CPU cores in total, whereas a GPU, for example GTX 580, contains more than 500 cores on a single board. Regarding on the number of cores, space and power consumption, GPUs seem superior than multicore clusters. On the other hand, CPU cores are much more powerful and flexible than GPU cores. Using OpenMP, we do not need to write a parallel program explicitly. On these issues, multicore environments seem superior than GPUs. A best practice would be to use both systems simultaneously.

4.2. Scaling up computer simulation using a GPU

Realtime simulation enables us to conduct computer simulation for a very long period. For example, cerebellar motor learning involves two time courses: short-term learning for a few hours and long-term learning for weeks and months (Okamoto, Endo, Shirao, & Nagao, 2011; Shutoh, Ohki, Kitazawa, Itohara, & Nagao, 2006). To date, it was impossible to simulate the process of such long-term memory formation which lasts for 1 week completely, because there was no way to simulate the whole 168 h with temporal resolution of milliseconds. Now, we are ready to conduct the computer simulation to explore the detailed process of the learning and memory formation.

CUDA supports simultaneous use of multiple GPUs. This allows us to instantly scale up a model to include more synapses, compartments and neurons while keeping realtime of computer simulation. For example, cerebellar granule cells are the largest population among all neurons in the brain. The cell density is about $1,000,000/mm^3$, which is 10 times more than the present model. Now, it is no longer impossible to simulate the dynamics of the whole cerebellar granular layer of 1 mm^3 in a reasonable simulation time. Such high-performance neural network simulation may open a new era in the field of computational neuroscience.

4.3. Differences between our previous models and the present model

The present model is slightly different from our previous models (Yamazaki & Nagao, 2012; Yamazaki & Tanaka, 2007b) in the following 5 respects, which makes quantitative comparison difficult.

First, Yamazaki and Tanaka (2007b) lacks basket cells, which are molecular layer interneurons that receive excitatory inputs from parallel fibers and issue inhibitory inputs to Purkinje cells. This shortcoming was overcome by our latest model (Yamazaki & Nagao, 2012) as well as the present model. Second, the previous models employ the 4th-order Runge–Kutta method for numerical integration, whereas the present model uses the 2nd-order one to reduce the number of arithmetic operations by 50%. This is necessary to achieve realtime simulation. Third, the previous models use Mersenne Twister (Matsumoto & Nishimura, 1998) as a pseudo random number generator, whereas the present model relies on a linear congruential generator. The former requires a relatively large memory array per thread, which could use up local memory on CUDA cores and slow down the simulation, whereas the latter consumes only one integer per thread. Fourth, and most important, all double-precision (`double`) floating-point operations in the previous models were replaced with single-precision (`float`) operations in the present study. This could degrade the accuracy of numerical integration to solve differential equations. NVIDIA GeForce series including GTX 580 used in the present study focuses primarily on the performance on integer and single-precision floating-point operations for faster graphics in games. Double-precision operations are 2–3 times slower than the single-precision ones. Therefore, there was a trade off between the accuracy and the speed, and we chose the speed. Finally, compilers are different from the previous and the present models (`gcc` versus `nvcc`).

The most important issue on numerical simulations is to guarantee the accuracy of numerical calculation. It is always desirable to use double-precision or higher-order precision floating points. For example, to replicate precise spike timing of a biological neuron by a spatial model neuron in noise-free conditions with a fine temporal resolution, accurate numerical integration of differential equations are definitely required. On the other hand, the present model is driven by noisy Poisson spike trains as mossy fiber inputs and multiple neurons are involved to cancel the noise out by averaging (Yamazaki & Tanaka, 2007b). Thus, the population activity of the neurons are more important than the activity of individual neurons, implying that the present model as a whole system is, by chance, insensitive to the accuracy of numerical calculation for individual neurons. Nevertheless, we still need higher accuracy for further development. To improve the accuracy while retaining the realtime simulation capability, we could use a high-end GPU in the category of NVIDIA Tesla series, which provides faster double-precision operations.

4.4. Potential applications of RC

We adopted RC to timing control of a humanoid robot to demonstrate the power of realtime computing. RC can learn not just timing information, but also gain information, which are two important quantities for precise motor control (Yamazaki & Nagao, 2012). In general, RC is a member of reservoir computing, which is a class of general-purpose supervised learning machine of spatiotemporal information (Yamazaki & Tanaka, 2007a). Therefore, RC could, in principle, be used to learn and represent any types of spatiotemporal information in a supervised learning manner. Furthermore, reservoir computing models have been demonstrating their versatile computational power for engineering applications including robot navigation (Antonelo & Schrauwen, 2010), speech recognition (Skowronski & Harris, 2007), and time-series prediction (NN3, 2007), and so on (for other literature, e.g., Jaeger, Maass, & Principe, 2007). Although in this study we only demonstrated timing learning by RC, we expect that RC can demonstrate such versatile computational power as well.

This versatile computational power is particularly important for a model of the cerebellum. The cerebellum is thought to acquire internal models of physical and mental objects by supervised learning and manipulate them for predictive control (Ito, 2011). Internal models should be acquired online through repeated practice, so that the ability to acquire any kind of spatiotemporal signals online is inevitable. RC is a potential candidate for this, because this is a general-purpose supervised learning machine of spatiotemporal information and performs realtime information processing. In sum, RC provides a means of adaptive control for physical hardware and a model of the cerebellum to explore the computational principles for learning and memory in the cerebellum.

Acknowledgments

We would like to thank Mr. Mamoru Kubota at The University of Electro-Communications for creating the swing motion of a robot. This study was supported by KAKENHI (20700301, 23300055).

Appendix. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.neunet.2013.01.019.

References

Antonelo, E., & Schrauwen, B. (2010). Supervised learning of internal models for autonomous goal-oriented robot navigation using reservoir computing. In *Robotics and automation (ICRA), 2010 IEEE international conference on* (pp. 2959–2964).

Berthier, N. E., & Moore, J. W. (1986). Cerebellar Purkinje cell activity related to the classically conditio ned nictitating membrane response. *Experimental Brain Research*, *63*, 341–350.

Carrillo, R. R., Ros, E., Boucheny, C., & Coenen, O. J.-M. (2008). A real-time spiking cerebellum model for learing robot control. *BioSystems*, *94*, 18–27.

Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., & McDonald, J. (2000). *Parallel Programming in OpenMP*. Morgan Kaufmann.

Christian, K. M., & Thompson, R. F. (2003). Neural substrates of eyeblink conditioning: acquisition and retention. *Learning & Memory*, *10*, 427–455.

Coesmans, M., Weber, J. T., De Zeeuw, C. I., & Hansel, C. (2004). Bidirectional parallel fiber plasticity in the cerebellum under climbing fiber control. *Neuron*, *44*, 691–700.

Garcia, K., & Mauk, M. (1998). Pharmacological analysis of cerebellar contributions to the timing and expression of conditioned eyelid responses. *Neuropharmacology*, *37*, 471–480.

Goodman, D. F. M., & Brette, R. (2009). The Brian simulator. *Frontiers in Neuroscience*, *3*, 192–197.

Hofstötter, C., Mintz, M., & Verschure, P. F. M. J. (2002). The cerebellum in action: a simulation and robotics study. *European Journal of Neuroscience*, *16*, 1361–1376.

Honda, T., Yamazaki, T., Tanaka, S., Nagao, S., & Nishino, T. (2011). Stimulus-dependent state transition between synchronized oscillation and randomly repetitive burst in a model cerebellar granular layer. *PLoS Computational Biology*, *7*, e1002087.

Igarashi, J., Shouno, O., Fukai, T., & Tsujino, H. (2011). Real-time simulation of a spiking neural network model of the basal ganglia circuitry using general purpose computing on graphics processing units. *Neural Networks*, *24*, 950–960.

Ito, M. (2001). Cerebellar long-term depression: characterization, signal transduction, and functional roles. *Physiological Reviews*, *81*, 1143–1195.

Ito, M. (2002). The molecular organization of cerebellar long-term depression. *Nature Reviews Neuroscience*, *3*, 896–902.

Ito, M. (2011). *The cerebellum: brain for an implicit self*. FT Press.

Jaeger, H., Maass, W., & Principe, J. (2007). Echo state networks and liquid state machines.

Jirenhed, D.-A., Bengtsson, F., & Hesslow, G. (2007). Acquisition, extinction, and reacquisition of a cerebellar cortical mem ory trace. *Journal of Neuroscience*, *27*, 2493–2502.

Kawato, M., & Gomi, H. (1992). A computational model of four regions of the cerebellum based on feedback-error learning. *Biological Cybernetics*, *68*, 95–103.

Lev-Ram, V., Mehta, S. B., Kleinfeld, D., & Tsien, R. Y. (2003). Reversing cerebellar long-term depression. *Proceedings of the National Academy of Sciences of the United States of America*, *100*, 15989–15993.

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM:Transactions on Modeling and Computer Simulation*, *8*, 3–30.

Mauk, M. D., & Donegan, N. H. (1997). A model of Pavlovian eyelid conditioning based on the synaptic organization of the cerebellum. *Learning & Memory*, *3*, 130–158.

Miikkulainen, R., Bednar, J. A., Choe, Y., & Sirosh, J. (2005). *Computational maps in the visual cortex*. Springer.

Miyamoto, H., Kawato, M., Setoyama, T., & Suzuki, R. (1988). Feedback-error-learning neural network for trajectory control of a robotic manipulator. *Neural Networks*, *1*, 251–265.

Nageswaran, J., Dutt, N., Krichmar, J., Nicolau, A., & Veidenbaum, A. (2009). A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Networks*, *22*, 791–800.

NN3 (2007). Neural Forecasting Competition 3. http://www.neural-forecasting-competition.com/NN3/.

NVIDIA (2009). NVIDIA's Next Generation CUDA Compute Architecture: Fermi.

NVIDIA (2011). CUDA Programming Guide 4.0.

Okamoto, T., Endo, S., Shirao, T., & Nagao, S. (2011). Role of cerebellar cortical protein synthesis in transfer of memory trace of cerebellum-dependent motor learning. *Journal of Neuroscience*, *31*, 8958–8966.

Pacheco, P. (1996). *Parallel programming with MPI*. Morgan Kaufmann.

Patterson, D. A., & Hennessy, J. L. (2011). *Computer organization and design: the hardware/software interface* (4th ed.). Morgan Kaufmann.

Perrett, S., Ruiz, B., & Mauk, M. (1993). Cerebellar cortex lesions disrupt learning-dependent timing of conditioned eyelid responses. *Journal of Neuroscience*, *13*, 1708–1718.

Shibata, T., & Shaal, S. (2001). Biomimetic gaze stabilization based on feedback-error-learning with nonparametric regression networks. *Neural. Netw.*, *14*, 201–216.

Shutoh, F., Ohki, M., Kitazawa, H., Itohara, S., & Nagao, S. (2006). Memory trace of motor learning shifts transsynaptically from cerebellar cortex to nuclei for consolidation. *Neuroscience*, *139*, 767–777.

Skowronski, M., & Harris, J. (2007). Automatic speech recognition using a predictive echo state network classifier. *Neural Networks*, *20*, 414–423.

Yamazaki, T., & Nagao, S. (2012). A computational mechanism for unified gain and timing control in the cerebellum. *PLoS ONE*, *7*, e33319.

Yamazaki, T., & Tanaka, S. (2005). Neural modeling of an internal clock. *Neural Computation*, *17*, 1032–1058.

Yamazaki, T., & Tanaka, S. (2007a). The cerebellum as a liquid state machine. *Neural Networks*, *20*, 290–297.

Yamazaki, T., & Tanaka, S. (2007b). A spiking network model for passage-of-time representation in the cerebellum. *European Journal of Neuroscience*, *26*, 2279–2292.

Yamazaki, T., & Tanaka, S. (2009). Computational models of timing mechanisms in the cerebellar granular layer. *Cerebellum*, *8*, 423–432.