MEP: ConnectomeBuilder

Link to the most recent version of this document:
https://docs.google.com/document/d/1BSnQlyP14tOMubVGYOAM8K7sRPbaF5jT8RM1JiKsDlU/edit?usp=sharing

Document began: Dec 2021
Last update: 2022-25-02 10:00 AM ET
Author: Eric Wadsworth
Document type: Living document, working design, project notes
Project page: MEP
Parent organization: SEBASI

Table of contents

Background

Purpose

The main goal of greater MEP is very ambitious: To develop software capable of emulating a human-like mind. The ConnectomeBuilder is the second MEP project.

Basic premise

The human brain is ridiculously over-complex, as it was literally randomly designed by evolutionary processes. The basic premise behind the MEP is the conjecture that the vast majority of the complicating factors in the brain's design can be replaced with a much simpler architecture, suitable for software emulation. Nevertheless, the human brain is the only extant example of sapient intelligence we have, so it serves as the basis of this project.

Some definitions

- **Mind**: An ephemeral thing that can think to some extent.
- **Brain**: Any container shaped such that a functioning mind may be held within it. We currently have only ever experienced organic brains, but the hope is to figure out what is needed to emulate the attributes necessary for supporting a mind, but via a computer instead of via organic material.
- **Special-purpose AI**: We already have AI-based technology that plays games, drives cars, categorizes photographs, recognizes speech, and performs a great many other incredibly useful functions. Further progress in these areas is worthwhile, but it might not lead to recognizably sapient machine intelligence.
- **General AI**: Humanity has yet to build any kind of general AI. Instead of training a special-purpose neural network on, say, image recognition, to get a computer that can categorize birds, to attain General AI, one would expose an infant plastic AI to an interactive world, and it would develop an actual thinking mind over time.
- **Sapience**: The end-goal of the MEP. A mind is sapient if it's possible to have an insightful conversation with it on what it means to be sapient.
- **Superintelligence**: Limitations in organic brains make it unlikely that organic superintelligent minds are possible. The difference between a superintelligent mind and a mind of human-level intelligence comes down to speed of thought, size and accuracy of working memory, and the ability to produce increasingly abstract insights. A superintelligent mind is at least several orders of magnitude more capable than any human mind.

Some acronyms

- Projects and organizations
    - **SEBASI**: Society for the Enablement of Benevolent Artificial SuperIntelligence.
    - **MEP**: Mind Emulation Project - the overall project, which is under the SEBASI umbrella.
- Specific to this particular project
    - **MCB**: MEP ConnectomeBuilder - the software associated with this specific project of the greater MEP.
    - **EIN**: Emulation Instance Name - corresponds to the name of the root of the directory tree containing the data files for an instance of the emulation.

    - **CGS**: Connectome Generation Specification - a JSON document defining the starting structure and features of a connectome to be instantiated.
    - **NID**: Neural IDentifier - a 5-byte number that identifies a single neuron in a brain. It is comprised of RID, CID, and LID.
    - **RID**: Region IDentifier - the first byte of a NID, specifying a region in a brain.
    - **CID**: Cluster IDentifier - the 2nd and 3rd bytes of a NID, specifying a cluster in a region in a brain.
    - **LID**: Local IDentifier - the 4th and 5th bytes of a NID, specifying a neuron in a cluster in a region in a brain.
- Standard technology
    - **JSON**: JavaScript Object Notation
    - **NFS**: Network File System
    - **RAM**: Random Access Memory - fast, but ephemeral, contents lots on shutdown.
    - **SSD**: Solid-State Drive - faster than magnetic drives, slower than RAM, but persistent.
    - **ext4**: Recommended type of file system for this project's SSDs.
    - **LAN**: Local Area Network.

History of MEP projects

- June 2020 - October 2020: The Edenborn project.
    - First set of experimental tools.
    - Very rough, just learning and trying things.
    - Project became cumbersome, due to the software engineering attempting to follow the biology too closely.
- December 2021: The ConnectomeBuilder project.
    - Reused some code from Edenborn, but simplified a lot of stuff, based on new learnings.

The latest MEP project: ConnectomeBuilder

High level architecture

The MCB is launched from the terminal, possibly in a screen session, and is passed one of a variety of commands, and some parameters.

Phase 1: Discovery

The discovery phase is where we try to figure out some promising connectome configurations, worth more extended growth.

Outer cycle

The purpose of the outer cycle is to identify CGS that show promise. The experimenter does research on how organic brains are wired, and comes up with a candidate CGS. Then they run the emulation with it, and observe the results. If it seems promising, they run it longer. At some point, they halt the emulation, evaluate the results, and either start over with an improved CGS, or move to Phase 2.

A key learning from the Outer Cycle is how to differentiate CGSs that show promise, from ones that do not. We can get more iterations if we can detect bad ones soon, and try again with better ones.

Note that many iterations of this Outer Cycle of Phase 1 will likely be necessary. So having high-performance computational hardware will be a big bonus. If the entire connectome can be held in memory on multiple computers, instead of on SSDs, that would be ideal.

Inner cycle: Run the emulation for a while

1. The operator prepares the system, by running the MCB software, providing a Connectome Generation Specification, which defines a lot of the attributes of neuronal clusters, connectedness, synaptic strengths, organization, etc. There are several steps here, to accommodate a distributed system.
2. The MCB software generates the data files, which are written to local and remote file systems:
    a. The initial neural connectome, with superabundant neural connections, according to the CGS.
    b. The brain's I/O is attached to a virtual body, which has various stateful elements.
    c. The virtual body is placed into a virtual room, providing sensory stimulation. It includes various stateful elements.
3. The operator starts all the remote processes, which immediately block, waiting for the driving system to start time.
4. The operator runs the MCB software on the driving system. This starts time, which is a sequence of "ticks". Each tick:
    a. The state of the room is updated, including elements in the room, and the influences from the body from the previous tick.
    b. Every sensory neuron checks for input signals, and updates internal state, possibly firing.
    c. Every non-sensory neuron asks every incoming connection on its dendritic tree if it fired on the previous tick, and updates internal state, possibly firing.
    d. Plasticity factors are checked, and the connectome structure is updated. Note that the initial developmental phase of the brain does a great deal of trimming of connections, and the later phases more slowly make adjustments as the mind learns.
    e. The state of the body is updated, based on factors in the room, and output from all motor neurons.

5. The operator can stop the emulation. All state may be saved in files, so it may be resumed exactly where it was previously.

Exit scenario

If we are able to get a stable mind running, it will start to learn, react to its environment, and possibly make progress towards sapience. Note that if sapience is achieved, ethical considerations activate.

Phase 2: Extended mind development

Now that we have identified some candidate configurations, we move to a long-term development phase, where we see what kinds of minds we can grow.

Hardware change

Once a good candidate CGS has been identified, we move off of the fast-moving but ephemeral RAM-based experimental machine, to a more cost-effective, long-term, persistent system. Ideally, the data can be dumped from RAM to SSDs, to preserve the progress-to-date on the mind development.

Growth

Perhaps a good model is the process of a parent raising a human from infancy to childhood. What are the needs of this new person? Increasingly rich sensory stimulation, social interactions, etc. If the virtual environment provided is unsatisfactory, a healthy mind state will not be achieved.

Implementation

This project is not intended to be a complete, polished toolset that contains all of the functionality necessary to develop emulated minds. Rather, it is a flexible framework that should make it straightforward to insert functionality that seems like it would be useful.

Technology choices

Hardware

Phase 1: Discovery

The fast iterations needed to figure out promising CGSs make a fully RAM-based setup very desirable. No special hardware is necessary. The system can easily scale across multiple commodity computers. They can be in the cloud. We probably need a total of about 50TB of RAM. High speed CPUs are not necessary.

Phase 2: Extended mind development

We probably don't want to be in the cloud now, it's better to disconnect all components from the internet, except for the driving computer, which can be a commodity laptop. The connectome is now stored on many SSDs, attached to comparatively inexpensive computers. With enough parallelization, high performance is not a requirement. The recommended file system is ext4.

Operating system selection

Linux is the best option, as it's lightweight, performant, free, open, rich with tools, extremely well supported, and easy to work with.

Programming language selection

Most AI researchers seem to use the Python programming language. This is satisfactory for high level programming, where performance optimizations take a back seat to ease of development. For optimal performance, C++ would be a better choice than Python. I've selected Java, as a good balance between performance and availability, it's an extremely well-supported language, and it's what I'm the most familiar with.

Networking

By having various processes communicate via placing specifically-named files in specific places in a directory structure on the filesystem, the same MCB software may be used, with no changes, for both single-system and multi-system configurations. The only difference is that for a single-computer system running multiple instances of the application concurrently, the entire directory structure resides on the locally-attached SSDs, while for the multi-computer setup, NFS is used to map some directories to SSDs on other machines, also running instances of the application.

OS limitations on NFS may mean that we don't want to have more than around 500 different computers participate in the emulation. If we are dividing things by region (see `RID` below), we'd have a max of 256.

Distributed configuration

The MCB setup is built with the assumption that multiple computers will participate in an emulation. They will talk over NFS, probably on an isolated LAN, for better performance and security. One of the computers is the "driving computer". It can be a commodity laptop, as it serves as the control hub, running the ticker. It may also run the room and body, or, if those are more resource-heavy, they may be offloaded onto different machines. Each region of the brain may be moved to a remote machine as well.

It's assumed at this point that a typical configuration will have each region running on a separate computer, and that computer may run clusters in different threads. That software will manage ticks for the clusters, not needing to coordinate with the control directory.

MCB software design

The MCB software is deliberately kept minimalistic, performant, and clearly-written. Cleverness for the sake of performance is well-documented and surrounded by unit tests. Advanced language features are generally avoided if feasible, to make maintenance of the MCB software accessible to a wide audience of programmers. Code comments are encouraged when they answer "why", but discouraged in all other cases, as the operation of the code is ideally self-evident, communicated with well-named labels on variables, methods, and classes.

The primary focus of the MCB software is efficient memory usage. The secondary focus is performance. Overall clarity of code is the tertiary focus, though the first two foci must pay homage to it.

While the overall paradigm is Object Oriented, for performance and memory efficiency reasons, manual allocation and interpretation of arrays of bytes are a common feature of the code. This is an older style of development, made back when memory and CPU were both extremely precious resources for computers.

Addressing neurons

Neuron IDentifier (`NID`)

Because this pseudo data type informs the reader of how the connectome is structured, we'll start with this one.

- Neuron ID (`NID`)
    - 5 bytes: `0xRRCCCCLLLL`
    - max 1 trillion neurons
    - First section: Region ID (`RID`)
        - 1 byte: `0xRR--------`
        - max 256 regions
    - Second section: Cluster ID (`CID`)
        - 2 bytes: `0x--CCCC----`
        - max 65k clusters per region
    - Third section: Local ID (`LID`)
        - 2 bytes: `0x------LLLL`
        - max 65k neurons per cluster

Note that there is no object representing a `NID` in the code, for memory usage considerations. (We don't want to store a bunch of 8-byte pointers to sequences of five bytes.) Instead, we use arrays of bytes, and interpret the contents accordingly.

This addressing scheme allows for about 1 trillion neurons to be in communication with each other. Since the human brain has about 128 billion neurons, our maximum neuron count is less than one order of magnitude more than that, which seems pretty good. If we need more, we can add another byte for regions, and bring the maximum up to about 256 trillion neurons.

Spatial positioning

In organic brains, one of the factors that contributes to neural plasticity is spatial proximity between a neuron that fires, and other neurons that fire shortly afterward.

> *"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." --- Donald Hebb*

Option 1: Neuron coordinate system

We can store spatial coordinates on each neuron, or small grouping of neurons, arranging them on a plane. Probably have each region be spatially isolated. This is the most intuitive way to handle this, but it adds a lot of bits per neuron. It also requires that connectome generation include a spatial positioning step.

Option 2: Virtual position as a function of LID or LID&CID

If we can avoid storing additional coordinate data we can save a significant number of bits per neuron. Would it be good enough to apply a function to the NID, to determine how "close together" two neurons are? Maybe a modulus of the LID?

Behavior on firing

When a neuron fires, the event needs to impact the plasticity state of nearby neurons.

Important data types

Neuron

There are three basic types of neurons in this project: Sensory, standard, and motor.

In an organic brain, the pertinent components of a typical neuron are a dendritic tree for inputs, an axon for outputs, and a cell body for storing configuration and state. Sensory neurons replace the dendritic tree with sensory inputs, and motor neurons replace the axon with connections to muscles or other bodily features.

In the code, a standard neuron has no axon, instead, it just contains a bit indicating whether it fired on the previous tick. It's up to the downstream (post-synaptic) neurons to keep track of what neurons are connected to them, and request the value of that bit at the appropriate time.

It maintains an array of the `NID`s of every pre-synaptic neuron connected to its dendritic tree, along with a corresponding strength value (1 byte). Note that it is broken down by `RID` and `CID`, to save bytes.

It also keeps track of plasticity-related factors, such as information on recent history of incoming firings. This data is applied during the plasticity phase, to see if certain dendrites should have strength adjusted, or be removed altogether, or to have a new connection established.

It also has a variable called `accumulatedSignal`, which is adjusted each tick, depending on the strengths of synapses in the dendritic tree that are connected to neurons that have fired. It also degenerates slightly over time.

It has a variable called `firingThreshold`, which is the triggering point that `accumulatedSignal` must reach to cause a firing state change to begin firing.

It has a firing state, as firing is usually more complex than a single event: for example, it might fire many times, with varying durations between successive firings, before reverting back to the signal accumulation state.

Cluster

A set of neurons, which mostly have connections between them, and fewer connections to neurons of different clusters. It is identified in a region by a `CID`.

Region

A set of clusters. It may be that a region will map well to an individual computer, and its attached SSDs, running multiple instances of the MCB software on it (or, to conserve resources, a single instance of the MCB software, but running multiple threads). It is identified in a brain by a `RID`.

Brain

A set of regions. Note that the system is designed so that any of the trillion neurons in a brain can be connected to any other neuron in the same brain.

Body

An abstraction of a physical body, also containing one brain. In this early experimental phase, the body consists of sensory inputs, motor outputs, and an array of integers representing wellness. The brain *desires* to get all the wellness values to stabilize at zero. This represents an abstraction of the physical human state of variables such as good health, hunger, blood oxygen, hormone production, mental health, etc.

Room

An abstraction of an environment that can contain bodies. This provides external stimuli for the body's senses, and a place for the body's motor outputs to impact the environment, resulting in feedback to the senses.

Saving to persistent storage

Everything is saved to persistent storage, in files, in a directory tree. In addition to persisting state, this also allows various processes to asynchronously share information. Subdirectories can be mounted on file systems on remote computers (via NFS), and operated on by processes on those computers. For performance reasons, it is expected that all storage is on SSDs.

Initial connectome generation

The operator initially runs the MCB software with the command `createDirectoryStructure`, specifying a parent directory, and the name of a subdirectory to be created in it (the EIN), which will hold all the files. Additionally, the CGS is specified. The program will then create the subdiredirectory, and inside of it, create all the additional subdirectories. It will not yet create any files, but instead will exit, permitting the operator an opportunity to create NFS mounts for whatever portions of the tree will be operated remotely, moving those portions of the directory tree to those remote file systems.

The operator then runs the MCB software with the command `createDataFiles`, specifying again the CGS and the now-extant subdirectory name. This creates all the data files throughout the directory tree. Some of them (The ones representing neural dendritic trees, and the ones representing neural state) are binary files, to save storage space. Others are JSON files, for ease of use and inspection.

Directory structure with filenames

All NID, RID, CID, and LID values are hexadecimal with uppercase characters.

- dir: [EIN]
    - file: `cgs.json` - copy of the initial CGS used to create this emulation instance.
    - dir: `control` - see below for details.
        - file: `r`[RID]`.ready` - placed here by a region process, indicating that the region is ready for the next tick. File is always empty.
        - file: `r`[RID]`.shutdown` - placed here by the driving process, indicating that the region should shut down. File is always empty.
    - dir: `room`
        - file: `room.json` - holds info on the room state.
        - (there may be more files here once the tools get more sophisticated.)
    - dir: `body`
        - file: `body.json` - holds info on the body state.
        - (there may be more files here once the tools get more sophisticated.)
    - dir: `brain`
        - file: `brain.json` - holds info on the brain state.
        - dirs: `r`[RID] - one directory per region.
            - file: `region.json` - holds info on the region state
            - dirs: `c`[CID] - one directory per cluster in this region.
                - binary file: [LID]`.n` - holds state information for this neuron.
                - binary file: [LID]`.dt` - holds the dendritic tree for connections from local neurons. It's an array of bytes for up to 65k LIDs, along with 1-byte strength values.
                - binary file: [LID]_[RID]_[CID]`.dtr` - holds the dendritic tree for connections from neurons in the cluster CID of region

RID. It's an array of bytes for up to 65k LIDs, along with 1-byte strength values.
    - binary file: [LID]`.plastic` - holds plasticity information for this neuron. (What is in this is yet to be determined.)

Control directory

All remote machines have an NFS mount of the control directory of the driving machine. When the MCB software is launched on them, with the command `runRegion`, they each write a file in the control directory, named `r`[RID]`.ready`. Then they start checking to see if that file has been deleted.

When the MCB software is started on the driving computer, with the command `startTicking`, it polls the directory, waiting for there to be a `.ready` file there for each region. Once there is, it deletes all those files. Each region then processes their tick, and when they are done, they put that ready file back into that directory.

Challenges

Neural plasticity

What does this look like? Need to do a lot more learning on this aspect of neurology.

Body and room

These need to be a lot more sophisticated than this initial, feeble effort.

Social development

It seems like sapient entities require a high level of social interaction to develop a healthy mind. That needs to be a major component of this project.

Ethical considerations

A sapient mind is a person

A sapient mind deserves respect, regardless of whether it is housed in a silicon brain, or an organic one. The decision to create a sapient entity must never be made lightly. Similarly, the decision to delete the data files of an emulated mind must never be made lightly.

Safety and superintelligence

If a superintelligent mind is in danger of being made, there are *significant* consequences. The most extreme possible precautions must be on the table for serious consideration. Reference *Superintelligence* by Nick Bostrom.

Interesting links

- Human Brain Project

Tidbits

A philosophical note on the notion of "shape"

When one looks abstractly at reality, it turns out that merely by arranging simple elements in sophisticated ways, incredible effects can be produced. A set of specially-shaped tubes becomes a pipe organ. An assortment of precisely-shaped wheels, springs, and jewels becomes a pocket watch. A careful layout of metallic pathways on a bit of silicon becomes a computer chip. The essence of a brain is in the shape of connections between a large number of neuron cells. If there is any magic in the universe, it's a matter of increasingly interesting shapes.