(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 September 2024 (19.09.2024)

WIPO | PCT

(10) International Publication Number
WO 2024/189144 A1

(51) International Patent Classification:
*G06F 9/50* (2006.01) *G06N 3/092* (2023.01)
*G06N 3/02* (2006.01)

(21) International Application Number:
PCT/EP2024/056811

(22) International Filing Date:
14 March 2024 (14.03.2024)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
63/452,143 14 March 2023 (14.03.2023) US

(71) Applicant: **DEEPMIND TECHNOLOGIES LIMITED** [GB/GB]; 5 New Street Square, London EC4A 3TW (GB).

(72) Inventors: **WANG, Pengming**; 6 Pancras Square, London N1C 4AG (GB). **SAZANOVICH, Mikita**; 6 Pancras Square, London N1C 4AG (GB). **ILBEYI, Berkin**; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **PHOTHILIMTHANA, Phitchaya Mangpo**; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **PUROHIT, Manish Deepak**; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **TAY, Han Yang**; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US).

(74) Agent: **FISH & RICHARDSON P.C.**; Highlight Business Towers, Mies-van-der-Rohe-Str. 8, 80807 Munich (DE).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE,

(54) Title: OPTIMIZING MEMORY ALLOCATION USING REPRESENTATION NEURAL NETWORKS

(57) Abstract: Methods, systems, and apparatus, including computer programs encoded on computer storage media, for optimizing a memory allocation of a target program using a state representation neural network.

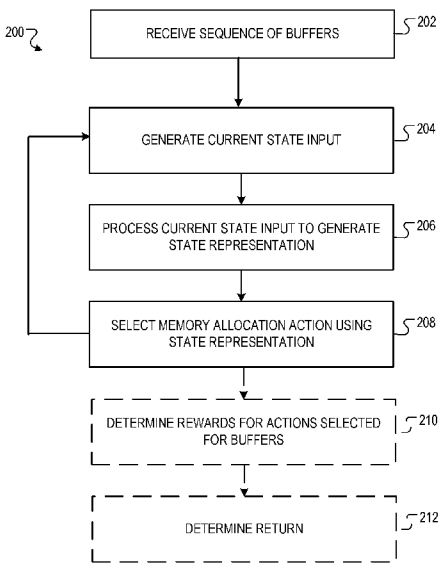



FIG. 2

WO 2024/189144 A1

[Continued on next page]

SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

**Published:**

— *with international search report (Art. 21(3))*

# OPTIMIZING MEMORY ALLOCATION USING REPRESENTATION NEURAL NETWORKS

## CROSS-REFERENCE TO RELATED APPLICATION

This application claims priority to U.S. Provisional Application No. 63/452,143, filed on March 14, 2023. The disclosure of the prior application is considered part of and is incorporated by reference in the disclosure of this application.

## BACKGROUND

This specification relates to processing inputs using neural networks.

Neural networks are machine learning models that employ one or more layers of nonlinear units to predict an output for a received input. Some neural networks include one or more hidden layers in addition to an output layer. The output of each hidden layer is used as input to the next layer in the network, i.e., another hidden layer or the output layer. Each layer of the network generates an output from a received input in accordance with current values of a respective set of parameters.

## SUMMARY

This specification describes a system implemented as computer programs on one or more computers in one or more locations that optimizes the memory allocation of a target program when executed on a target computer system having a plurality of memories.

The subject matter described in this specification can be implemented in particular embodiments so as to realize one or more of the following advantages.

Compute resource efficiency is critical in large-scale, real-world computer systems, and plays an increasingly greater role as demand for compute increases. For example, in the domain of machine learning, the demand for increased compute is accelerating at a fast pace, as workloads grow larger, and applications proliferate. Improving resource efficiency for compute intensive workload, e.g., machine learning workloads, hence presents an important opportunity to effectively manage the available compute of a given computer system.

One avenue towards this goal is to improve solutions to the memory mapping problem.

Generally, modern hardware architectures have multiple layers of memory hierarchy, differing in their sizes and speeds. Typically these memory layers range from large, but slow

memory layers (e.g. HBM on TPUv4), to increasingly smaller, but faster layers (e.g. CMEM on TPUv4).

The problem of determining when to use which memory layer, and managing data transfer between layers, is the memory mapping problem. More specifically, a solution to the memory mapping problem may define which buffers are allocated at what offsets in the fast memory, as well as the time interval each buffer is allocated in memory.

A good memory mapping means that the faster memory layers are utilized effectively, which can significantly reduce the overall execution time of the program.

However, finding optimal, or even just good solutions to this problem is an extremely challenging problem, as finding a solution requires balancing the resource trade-offs between fast memory space, execution time, and inter-memory bandwidth used for prefetching. This can be seen as an NP-hard scheduling problem. Conventionally, the memory mapping problem is "solved" by compilers through a series of expert-designed rule-based heuristics. While these approaches often perform well on average, they also frequently yield suboptimal results, as a fixed set of rules cannot cover all complex cases.

Instead, this specification introduces an approach that use a representation neural network trained through reinforcement learning to solve this problem, enabling the system find more optimal mappings for a variety of target programs, e.g., machine learning workloads. In particular, the described approach frames the memory mapping problem as a reinforcement learning problem and uses the representation neural network to represent the structure of the memory allocation problem and allow the system to incrementally build a solution for the memory mapping problem one buffer at a time.

This is a very challenging problem for a number of reasons. Firstly, as fast memory is limited (that is, in a standard system, there is a finite amount of memory with relatively fast processing speeds available for use), it is typically not possible to serve all instructions from fast memory. In addition, the copy bandwidth between fast and slow memories is limited, and moving buffers between memories can add additional execution overhead. As such, the system needs to balance the trade-off between available memory space, copy bandwidth and execution time efficiently. In addition, a program can have a large number, e.g., over one hundred, instructions which can make a single trajectory of buffer allocations very long. This results in an extremely large, combinatorial search space of over 10^(4000) possible allocation trajectories, exceeding other challenging problems. In this situations, early allocation decisions have long-lasting consequences. For instance, blocking memory space that could be used more efficiently later on or taking up too much copy bandwidth to copy an

important buffer into slow memory, could lead to sub-optimal performance results. As a result, planning is important in this problem domain, and the system described herein leverages the representation neural network in order to perform this task effectively. For example, the system can perform a search using representations generated by the neural network in order to plan the impact of performing a given allocation action for a given buffer.

The details of one or more embodiments of the subject matter of this specification are set forth in the accompanying drawings and the description below.

Other features, aspects, and advantages of the subject matter will become apparent from the description, the drawings, and the claims.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagram of an example memory optimization system.

FIG. 2 is a flow diagram of an example process for generating a memory allocation for a target program.

FIG. 3 shows an example of selecting a memory allocation action for a given buffer.

FIG. 4 shows an example of the resource trade-offs that are represented by different memory allocation actions.

FIG. 5 shows an example architecture of the representation neural network.

FIG. 6 shows an example of the performance of the described techniques.

Like reference numbers and designations in the various drawings indicate like elements.

## DETAILED DESCRIPTION

FIG. 1 shows an example memory optimization system 100. The memory optimization system 100 is an example of a system implemented as computer programs on one or more computers in one or more locations, in which the systems, components, and techniques described below can be implemented.

The memory optimization system 100 optimizes the memory allocation of a target program 102 when executed on a target computer system 110 having a plurality of memories 112.

The target computer system 110 can be any appropriate system of one or more computers that has a plurality of memories 112, e.g., that has a hierarchy of a plurality of memories where memories in lower levels of the hierarchy have a smaller capacity, e.g.,

memory capacity, but can be accessed, i.e., read to or written from, with less latency than memories in higher levels of the hierarchy.

As a particular example, the computer system 110 can be a hardware accelerator that is associated with multiple memories 112 arranged in a hierarchy. Memories in lower levels of the hierarchy have a smaller capacity but can be accessed, i.e., read to or written from, with less latency by the accelerator than memories in higher levels of the hierarchy.

As a particular example, the accelerator can have a large, but slow memory layer (e.g., a High Bandwidth Memory (HBM)) and a smaller, but faster memory layer (e.g. CMEM).

For example, the slow memory layer (also referred to in this specification as the "second memory") can have a large enough memory capacity to store all required data for executing the program, but storing data in the slow memory layer incurs data transfer costs during execution of the program (relative to storing the data in the faster memory). As another example, the faster memory layer (also referred to in this specification as the "first memory") can be accessed much quicker than the slower memory, but has limited capacity, so storing all data in the faster memory layer is not possible.

To optimize the memory allocation, the system 100 determines an allocation of each "buffer" 104 associated with the target program 102 across the plurality of memories 112.

Generally, the target program 102 includes a plurality of instructions, e.g., a sequence of instructions, and each buffer 104 represents a respective operand or a respective output of one of the plurality of instructions. During execution of the target program 102, each instruction is executed at a respective time during the execution. This time will be referred to in this specification as the "logical time" of the instruction and the buffers corresponding to the instruction.

The target program 102 can represent any of a variety of computational workloads to be executed by the target computer system 110.

For example, the target program 102 can be a program for performing inference using a machine learning model or for training a machine learning model, e.g., a deep neural network.

As another example, the target program 102 can be a workload for compressing or decompressing data, e.g., video data, image data, or other high-dimensional data. In such examples, in use the target program 102 may receive, as an input, one or more data items comprising video data, image data, and/or other high-dimensional data, process the received

data items to generate compressed or decompressed data items, and output the generated compressed or decompressed data items.

As another example, the target program 102 can be a workload for editing video data, image data, or other high-dimensional data. In such examples, in use the target program 102 may receive, as an input, one or more data items comprising video data, image data, and/or other high-dimensional data, process the received data items to generate edited data items, and output the generated edited data items.

More generally, for example, the target program 102 may be one which has high data processing requirements.

More specifically, to determine the memory allocation, the system 100 receives data specifying a sequence of buffers 104 that are associated with the target program 102 to be executed on the target computer system 110. For example, the buffers can be arranged within the sequence chronologically, i.e., according to the logical times of the buffers (and their corresponding instructions).

The system 100 generates a memory allocation 120 for the target program 102 that allocates each of the buffers 104 across the plurality of memories 112 during execution of the target program 102 on the target computer system 110. The memory allocation 120 is suitable for use in a compiling operation to generate an executable program that causes the target computer system 110 to allocate the buffers 104 across the plurality of memories 112 in accordance with the memory allocation 120 while executing the target program.

For example, the memory allocation 120 can attempt to allocate buffers 104 so that the total latency of executing the target program 102 on the target computer system 110 is minimized.

To generate the memory allocation 120, the system 100 allocates each buffer 104 according to the sequence, i.e., one by one in an order specified by positions in the sequence, with earlier buffers being allocated before later buffers in the sequence.

As part of generating the allocation, for each buffer, the system 100 generates a current state input specifying 128 a current state of the memory allocation 120 as of a position of the buffer in the sequence and processes the current state input using a representation neural network 130 that is configured to process the current state input to generate a state representation 132 of the current state of the memory allocation 120.

Generally, the state representation 132 is an ordered collection of numerical values, e.g., a vector, matrix, or higher-order tensor of floating point or other numerical values, that

represents the current state of the allocation 102, including properties of the current buffer and of the allocations of the preceding buffers in the sequence.

The system 100 then selects a memory allocation action for the buffer from a set of memory allocation actions using the state representation 132.

Each memory allocation action allocates the buffer differently across the plurality of memories during the execution of the target program on the target computer system.

The set of memory allocation actions generally includes, for each of the memories 112, one or more actions that allocate the buffer in the memory.

For example, when the memories include the first memory and the second memory, one or more of the actions can allocate the buffer into the first memory and one or more of the actions can allocate the buffer into the second memory.

When two actions allocate the buffer to the same memory, each action allocates the buffer differently within the memory, e.g., with one action having the effect of maintaining an earlier allocation of the tensor specified by the buffer by extending the duration for which the tensor is stored in the memory and another action having the effect of generating a new allocation for the buffer (even if the corresponding tensor had previously been allocated to the memory).

As a particular example, when the memories include a first memory and the second memory, the set of memory allocation actions includes a first memory allocation action that assigns the buffer to the first memory for a specified time interval. This action is also referred to as the "Copy" action.

The specified time interval includes ("covers") the time required to transfer the corresponding buffer from the second memory to the first memory, i.e., the time required to load the corresponding buffer into the first memory. The time required to transfer the corresponding buffer between the second memory and the first memory may be referred to as a data transfer time. The specified time interval including or covering the data transfer time means that the specified time interval is at least as long as (e.g. the same amount of time as or a greater amount of time compared to) the data transfer time.

Thus, selecting the Copy action allocates the buffer to the current available offset within the first memory, i.e., to the first offset within the first memory that is designated as available after the previous buffer was allocated.

In this example, the set of memory allocation actions also includes a second memory allocation action that, when the buffer specifies a same tensor as an earlier buffer that is earlier in the sequence, assigns the buffer to the same allocation in the first memory as the

earlier buffer and extends the specified time interval for the earlier buffer to include the logical time at which an instruction corresponding to the buffer is executed by the target program.

Thus, this memory allocation action is only valid when another buffer in the same alias group as the current buffer has already been allocated to the first memory and will be referred to as the "NoCopy" action.

That is, this second memory allocation action modifies the allocation of memory within the first memory so that the buffer is allocated within the first memory through the time at which an instruction corresponding to the buffer is executed by the target program. In other words, the NoCopy action extends the allocation of the tensor corresponding to the buffer at the same offset as the earlier allocation but for a longer period of time.

Additionally, the set of memory allocation actions can include a third action that assigns the buffer to the second memory without allocating the buffer to the first memory. In other words, the third action assigns the buffer to the second memory instead of to the first memory. This action will also be referred to as the "Drop" action.

An example of the "Copy, "NoCopy, and "Drop" actions is shown below with reference to FIG. 3.

In some cases, to select the action using the state representation 132, the system 100 can perform a search through a state space using the state representation 132 and then select the action using statistics computed during the search.

As another example, the system 100 can process the state representation 132 of the current state using a policy neural network to generate a probability distribution over the set of actions. The system can then select, of the actions that are valid given the current state, the action using the probability distribution, e.g., by selecting the action with the highest probability or by sampling from the distribution using an appropriate sampling technique.

Selecting memory allocation actions using the representation neural network 130 is described in more detail below.

Generally, the system 100 selects the memory allocation actions subject to a set of constraints.

For example, as described above, the NoCopy action can only be applied if there is a matching buffer already in the first memory.

As another example, the set of constraints can include an aliasing constraint that imposes that all buffers within the same alias group must either all be assigned to the second

memory or are all be assigned to the first memory (e.g., using either the Copy or NoCopy action).

As another example, the set of constrains can include a constraint that the total allocation of buffers to the first memory cannot exceed the size of the first memory at any point during execution of the target program.

Other constraints, e.g., on data transfer bandwidth and capacity between memories, can also be imposed. These additional example constraints are described in more detail below.

Once the memory allocation 120 has been determined, the system 100 can compile the target program 102 to generate an executable program that, when executed by the target computer system 110, causes the target computer system 110 to allocate the buffers 104 across the plurality of memories 112 in accordance with the memory allocation 120 while executing the target program 102.

The system 100 can perform this compilation using any appropriate compiler for the target computer system 110. For example, the compiler can generate an intermediate representation (IR) of the target program 102 and then generate, from the IR, machine code for running the program 102 on the computer system 110. One example of such a compiler is the accelerated linear algebra (XLA) compiler.

The system 100 can then cause the target computer system 110 to execute the compiled program or can provide the compiled program to another system for use in executing the target program 102 on the target computer system 110.

Alternatively, the system 100 can provide data specifying the memory allocation 120 to another system so that the other system can compile the target program 102 according to the memory allocation 120.

Optionally, before using the memory allocation 120 as the final memory allocation for the target program 102, the system 100 can first verify that the memory allocation 120 improves over a heuristic-based allocation for the target program 102, e.g., one generated by a compiler for the target computer system 110 by applying a set of heuristics to the buffers in the target program 102.

In particular, the system 100 can determine a second memory allocation for the target program using a different, heuristic-based solver.

The system 100 can then determine a first expected latency for the memory allocation and a second expected latency for the second memory allocation. For example, the system 100 can determine these expected latencies based on rewards computed for the allocations of

the buffers 104 in the two allocations, as will be described in more detail below.

The system 100 then selects the memory allocation 120 as the final memory allocation for the target program only when the first expected latency is lower than the second expected latency. That is, the system 100 only uses memory allocations 120 determined using the representation neural network 130 when those allocations improve over a heuristic-based allocation.

Generally, prior to using the representation neural network 130 to select the actions in the final memory allocation 120, the system 100 trains the representation neural network 130 and any other neural network components of the system 100 so that memory allocations 120 generated using the representation neural network 130 effectively minimize execution latency. The representation neural network 130 and any other neural network components will collectively be referred to as the "learned components" of the system 100.

For example, the system 100 can train the representation neural network 130 and, more generally, the learned components, through reinforcement learning.

In some cases, the system 100 trains the representation neural network 130 through reinforcement learning on data corresponding to memory allocations for the target program 102 on the target computer system 110.

In some other cases, the system 100 trains the representation neural network 130 through reinforcement learning on data corresponding to memory allocations for different target programs on the target computer system 110, and then generates the final memory allocation 120 for the target program 102 "zero shot." Generating the allocation 120 "zero shot" refers to generating the allocation 120 without further training the representation neural network 130 using training data corresponding to the target program 102.

Alternatively, the system 100 can train the representation neural network 130 through reinforcement learning on data corresponding to memory allocations for different target programs on the target computer system 110, and then generate the final allocation 120 after fine-tuning the representation neural network 120 on data corresponding to the target program 102 on the target computer system 110.

In particular, when training the learned components through reinforcement learning, the system 100 trains the learned components to optimize an expected return that will be received for a memory allocation generated by the system 100.

Generally, the return for a given memory allocation for a given computer program estimates a latency of the computer program when the computer program is executed on the target computer system in accordance with the given memory allocation.

For example, for a given sequence of buffers associated with a given program, the return can be a combination of respective rewards for the selected memory allocation actions for the buffers in the given sequence. For example, the return can be a sum or a time-discounted sum of the respective rewards for the buffers in the sequence.

For example, the respective reward for a selected memory allocation action for a given buffer in a given sequence can be the expected speedup achieved as a result of the selected memory allocation action.

The "expected speedup" is the expected speedup, i.e., the expected decrease in execution time, in executing the given program on the computer system that is achieved as a result of performing the memory allocation action given the selected memory allocation actions for the buffers that precede the given buffer in the given sequence.

In some implementations, the system 100 determines these expected speedups based on executing the instructions in the given program on the target computer system 110. That is, the system 100 can execute the program on the target computer system 110 numerous times with different combinations of buffers being allocated to the first memory, and then compute the expected speedups from the results of the execution.

In some implementations, the expected speedup for assigning the given buffer to the second memory is zero and the expected speedup for assigning the given buffer to the first memory is equal to an expected speedup of accessing the given buffer from the first memory instead from the second memory.

Thus, in this example, the returns estimate a relative speedup of assigning buffers according to the memory allocation relative to assigning all buffers to the second memory.

Thus, in this example, during training, the system 100 can use the learned components to generate training memory allocations and determine rewards for the selected memory allocation actions within the training memory allocations.

For each training memory allocation, the system 100 can generate a training example that identifies the selected memory allocation actions within the training memory allocation, the respective rewards for the selected memory allocation actions, and, optionally, the respective return for the training memory allocation.

The system 100 can then train the learned components on these training examples through reinforcement learning. For example, the system can use one of the reinforcement learning objectives described in Schrittwieser, et al, Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model, at arXiv: 1911.08265, Schrittwieser, et al, Online and Offline Reinforcement Learning by Planning with a Learned Model, at arXiv:2104.006294,

or Hubert, et al, Learning and Planning in Complex Action Spaces, at arXiv:2104.06303, to train the learned components.

By alternating between generating training examples and training the learned components using already-generated training examples, the system 100 can train the learned components to generate improved memory allocations.

FIG. 2 is a flow diagram of an example process 200 for generating a memory allocation for a target program. For convenience, the process 200 will be described as being performed by a system of one or more computers located in one or more locations. For example, a memory optimization system, e.g., the memory optimization system 100 depicted in FIG. 1, appropriately programmed in accordance with this specification, can perform the process 200.

The system receives data specifying a sequence of buffers that are associated with a target program to be executed on a target computer system (step 202).

As described above, the target computer system has a plurality of memories, the target program has a plurality of instructions, and each buffer in the sequence specifies a respective operand or a respective output of one of the plurality of instructions. In some cases, different buffers can represent the same tensor, e.g., if one tensor that is the output of one instruction is used as the operand for another instruction that tensor will be specified by two different buffers in the sequence. Two buffers that represent the same tensor will be referred to as having the same alias.

As a particular example, the buffers can be ordered in the sequence chronologically with respect to the appearance of each buffer within the target program. Chronological may be interpreted with respect to the logical time of the instruction and the buffers corresponding to the instruction of the target program.

The system then generates a memory allocation for the target program that allocates each of the buffers across the plurality of memories during execution of the target program on the target computer system. That is, the system can allocate each buffer according to the sequence, one after the other, by performing steps 204-208. That is, the system performs one iteration of steps 204-208 for each buffer in order to select a memory allocation action for the buffer to be included in the final memory allocation.

The system generates a current state input specifying a current state of the memory allocation as of the position of the buffer in the sequence (step 204).

The current state input can generally include any of a variety of features that characterize the current state of the memory location. The features may include any one or

more of: a memory map grid, a memory occupancy feature, a supply profile feature, a respective feature for each of the memory allocation actions, data characterizing the buffer, data characterizing other buffers with a same alias as the buffer, as described in more detail below.

As one example, the current state input can include a memory map grid that specifies a usage, e.g., according to the memory allocation, of at least one of the memories of the target computer system in a local neighborhood around the logical time at which the instruction corresponding to the buffer is executed by the target program.

More specifically, the memory map grid $M_t$ at logical time $t$ can be two-dimensional binary grid, with one axis corresponding to the logical time steps in the program, and the other corresponding to the memory locations in the first memory. A grid cell at coordinate ($t$, $o$) is occupied if at time step $t$, the memory location at offset $o$ is occupied, and it is empty if that memory location is free at $t$.

Because of the large size of the grid, the system can include, in the current state input, a local neighborhood around the logical time at which the instruction corresponding to the buffer is executed by the target program, e.g., a fixed size window centered at the logical time. Optionally, the system can downsample the local window to generate a fixed size binary image and include the fixed size binary image in the state input as the memory map grid feature.

As another example, the current state input can include a memory occupancy feature that specifies a usage, e.g., according to the memory allocation, of at least one of the plurality of memories at the logical time at which the instruction corresponding to the buffer is executed by the target program. For example, the memory occupancy feature can be a binary occupancy vector for the first memory that indicates whether, at time $t$, each memory offset within the first memory is occupied or not.

As another example, the current state input can include a supply profile feature that provides information about data transfer times of buffers between two or more of the plurality of memories.

For example, the supply profile feature can be a window of a supply profile vector $W_t$ centered at the target time of the current buffer. The supply profile vector is a vector describing the currently available copy supply value at each time step. The copy supply value is the time required for the corresponding instruction to be executed, i.e., the amount of time the execution of the program spends on the program.

As another example, the current state input can include a respective feature for each of the memory allocation actions. For example, for each of the actions, the system can include a feature that identifies the legality of the action, the start and end times of its corresponding time interval, and its offset of the corresponding placement.

As another example, the current state input can include features characterizing the buffer.

One example of information that can be included in a feature that characterizes a buffer is shown below in Table 1, where CMEM refers to the first memory and HBM refers to the second memory.

| Feature | Description |
|---|---|
| size | Size of the buffer in bytes. |
| is_output | Whether the buffer is an output or an operand. |
| target_time | Logical time of the instruction using the buffer. |
| tensor_id | Id of the corresponding tensor. |
| alias_id | Id of the corresponding alias group. |
| live_range | Logical time interval for which the buffer is available in the program. |
| demand | Required data transfer time to move the buffer between HBM and CMEM. |
| benefit | Estimated speedup if the buffer were placed in CMEM. |

Table 1

As another example, the current state input can include data characterizing other buffers with the same alias as the buffer.

As another example, the current state input can include data characterizing one or more next buffers that follow the current buffer in the sequence.

The system processes the current state input using a representation neural network that is configured to process the current state input to generate a state representation of the current state of the memory allocation (step 206).

The state representation neural network can generally have any appropriate architecture that allows the neural network to map a current state input that includes one or more of the above features to the state representation.

As one example, when the current state input includes at least the memory map grid feature and the buffer feature, the state representation neural network can be configured to process the memory map grid using an image encoder neural network to generate an encoded representation of the memory map grid and then generate a combined feature by combining the encoded representation with at least the feature of the buffer.

The state representation can then process the combined feature using a combined feature encoder neural network to generate the state representation of the current state of the

memory allocation.

A particular example of the architecture of the state representation neural network is described below with reference to FIG. 5.

The system selects, using the state representation, a memory allocation action for the buffer from a set of memory allocation actions (step 208). As described above, each memory allocation action allocates the buffer differently across the plurality of memories during the execution of the target program on the target computer system.

For example, the system can perform a tree search using the state representation in order to select the memory allocation action. Performing the tree search will generally make use of additional components of the neural network, i.e., components in addition to the representation neural network, e.g., one or more of a policy neural network, a dynamics neural network, or a prediction neural network.

The policy neural network is a neural network, e.g., a feed-forward neural network, that is configured to receive a state representation and to process the state representation to generate a probability distribution over the set of memory allocation actions.

The dynamics neural network is a neural network, e.g., a feed-forward neural network, that is configured to receive a state representation and data identifying a memory allocation action and to generate a predicted next state representation that represents the state of the memory allocation if the identified memory allocation action is selected when the allocation is in the state represented by the input state representation. The dynamics neural network can also predict a reward that will be received if the input memory allocation action is performed when the memory allocation is in the state represented by the input state representation.

As described above, the reward for performing a given memory allocation action when the memory allocation is in a given state represented by a given state representation is the expected speedup achieved as a result of performing the given memory allocation when the memory allocation is in the given state.

The prediction neural network is a neural network, e.g., a feed-forward neural network, that receives the input state representation and predicts a return that will be received starting from the state of the memory allocation is in a state represented by the input state representation.

These other components of the neural network and example techniques for performing a tree search to select an action is described in Schrittwieser, et al., Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model, at arXiv: 1911.08265,

Schrittwieser, et al, Online and Offline Reinforcement Learning by Planning with a Learned Model, at arXiv:2104.006294, and Hubert, et al., Learning and Planning in Complex Action Spaces, at arXiv:2104.06303, all of which are hereby incorporated by reference herein in their entirety.

An example technique for performing the look-ahead search to select an action is described in more detail below.

As another example, the system can process the state representation of the current state using the policy neural network to generate a probability distribution over the set of actions. The system can then select the action using the probability distribution, e.g., by selecting the action with the highest probability or by sampling from the distribution using an appropriate sampling technique.

In either of the above cases, the system can select only an action that does not violate any of the constraints on the memory allocation.

After the memory allocation action is performed, the system updates the state of the allocation, i.e., by updating the memory grid for the first memory and the data supply vector in accordance with the assigned memory allocation action for the current buffer.

The system can continue performing the steps 204-208 until the system has selected a memory allocation action for all of the buffers in the sequence. The system can then use the memory allocation to compile the target program or for another appropriate purpose as described above.

As described above, the system can train the representation neural network and, when used, the other learned components using reinforcement learning.

Accordingly, the system can also perform iterations of the process 200 during training to generate training data for training the learned components.

That is, when the process 200 is being performed during training, the system can also compute a respective reward for each selected memory allocation action (step 210) and, optionally, compute a return for the sequence of buffers from the respective rewards for the selected memory allocation actions (step 212). For example, the return can be the sum of the respective rewards for the selected memory allocation actions.

As described above, the rewards for the memory allocation actions can be expected speedups for the actions. For example, the expected speedup for assigning the given buffer to the second memory cane be zero and the expected speedup for assigning the given buffer to the first memory can be equal to an expected speedup of accessing the given buffer from the first memory instead from the second memory.

The system then generates a training example that includes the respective rewards for the selected memory allocation actions and, optionally, the computed return, and adds the training example to a set of training data for training the learned components of the system.

In some cases, during training, the system may select an action that causes the memory allocation to enter an infeasible state, i.e., a state where there are no valid memory allocation actions. For example, the allocation may enter a state where the current buffer must be placed into the first memory due to aliasing constraints but cannot be placed into the first memory because this would violate memory or bandwidth constraints. When this occurs, in some implementations, the system terminates generating the memory allocation and assigns a return of zero or other value that indicates a failed attempt to the partially-completed allocation.

The existence of these infeasible states can provide a significant challenge to learning. Since the total return for the allocation resets to zero when entering into an infeasible state, it generally comes with a large negative reward spike that negates the accumulated reward from the episode, meaning large discontinuities in the reward function. Correctly assigning this large negative reward to the offending decisions is very difficult, since decisions leading to the conflict can be arbitrarily far removed from the step where the conflict materializes. Moreover, it can be a combination of actions that collectively lead to a conflict later on, rather than a single incorrect action. Determining which actions lead to infeasibility later on in the game is generally a very hard problem, as it requires reasoning over a combinatorically large number of rollouts and showing that no possible continuation can successfully complete the game.

In some implementations, to account for this, the system can make use of a "backup" mechanism that resets the allocation to a feasible state, i.e., instead of terminating the generation. Generally, the system can use this mechanism to improve learning by resetting allocations that enter an infeasible state to a feasible state to allow for the neural network to learn from the accumulated rewards for the episode.

In particular, the system implements the backup mechanism based on the following observation: if in a given state, no future buffer shares the same alias with any already allocated buffer, then dropping all remaining buffers to the second memory is a valid complete solution to the allocation.

More specifically, if, while allocating the buffers in the sequence, the system determines that the memory allocation has entered an infeasible state, the system can identify a backup state at which no future buffer shares the same alias identifier with any already

placed buffer that has been allocated to the first memory and then generate a training example that, for each buffer to be allocated after the backup state, selects the third memory allocation action ("drop") that allocates the buffer to the second memory. This results in a valid memory allocation with a valid, non-zero return, allowing learning to progress.

FIG. 3 shows an example 300 of selecting a memory allocation action for a given buffer.

In particular, in the example of FIG. 3, the target computer system has two memories: the first memory (CMEM) and the second, slower memory (HBM).

The system receives the data specifying the program $\mathcal{P}$ as a sequence of $T$ instructions $(\mathcal{I}_1, ..., \mathcal{I}_T)$. The indices of the instructions in the sequence can be considered to be the logical times of the instructions in the program. Each instruction has a set of inputs and outputs, which are collectively referred to as the buffers used by the instruction. Each buffer has a set of properties, e.g., its size, the logical time of its instruction (its position in the instruction sequence of the program), or the expected speedup when reading (or writing) the buffer from the first memory (in our case CMEM) instead of the second memory (HBM).

Furthermore, the system also receives data specifying the total size of CMEM available (max_size) and an indication that the HBM is large enough to contain all buffers of the program.

The memory mapping problem for a given program $\mathcal{P}$ with buffers $\mathscr{B}$ is then to decide for each buffer $b \in \mathscr{B}$ whether to allocate space for it in CMEM, and if it is, for which logical time range and at what offset within CMEM.

That is, a solution to the memory mapping problem is a pair of functions $O : \mathscr{B} \rightarrow [0,$ max_size$) \cup \{\otimes\}$ and $I : \mathscr{B} \rightarrow [0, T]^2$.

The offset mapping $O$ assigns each buffer to its offset location in CMEM if it is allocated to it, or it assigns it to a special symbol $\otimes$, denoting that the buffer is to be allocated in HBM.

At the same time, the interval mapping $I$ assigns each buffer a logical time interval determining the time it is to be allocated in CMEM (and is undefined for buffers).

Another aspect that is considered by the system in the memory mapping problem is the data transfer cost to move buffers between HBM and CMEM. In order for an instruction $\mathcal{I}$ to use a buffer $b$ from CMEM, the system needs to allocate memory for a time interval that starts long enough before $\mathcal{I}$ to also take transfer time into account (sometimes called

prefetching). Overall, the system attempts make sure that time spent on transfer never slows down actual execution time, i.e., that copies are always overlapped fully by computation.

To model this, the system can keep track of the transfer cost of each buffer, which is referred as its copy demand value. For example, the copy demand value of a buffer can be proportional to the size of the buffer. As a particular example, the system can set the copy demand value of the buffer to the size of the buffer multiplied by a hardware-specific bandwidth constant.

The system also keeps track of the time available at each instruction for copies to be fully overlapped, which is referred to as the supply value of a logical step $t$. Now, when allocating a buffer $b$ into CMEM, the system need to allocate memory for a logical time interval such that the supply values during the copy duration cover the demand value of $b$.

As described above, the construction of $O$ and $I$ needs to adhere to a number of constraints, for instance ensuring that at no point a memory location is oversubscribed to multiple buffers, or that the time range a buffer is allocated to CMEM needs to account for data transfer times. As another example, the system can require that there is only a single buffer being copied between memories at any given point in time, e.g., to ensure that any copy of a buffer proceeds with the undivided maximum bandwidth available on the hardware.

More specifically, in the example 300, the system is placing a buffer $b_t$ that is at a time step $t$ in the sequence of buffers and that occurs at target_time($b_t$) within the computer program, i.e., at a specified logical time within the target program.

In the example 300, the set of memory allocation actions includes a "Drop" action, a "Copy" action, and a "NoCopy" action, as described above.

As described above, the system determines a copy interval for each buffer. In some cases, the copy interval can be determined based on which action was selected for the buffer.

After the system selects the action (in the example of FIG. 3, the "copy" action), the memory state $M_{t+1}$ is updated before a memory allocation action is selected for the buffer $b_{t+1}$ at time step $t$+1 within the sequence of buffers.

In particular, given a state $s_t$ of the memory allocation and a chosen action $a_t$ for the buffer $b_t$, the components of the next state $s_{t+1}$ are $<b_{t+1}, O_{t+1}, I_{t+1}, W_{t+1}, \mathcal{B}>$ and can be determined as follows:

$b_{t+1}$ – The next buffer is chosen as the next buffer from the set $\mathcal{B}$, in the chronological sequence of program instructions.

$O_{t+1}$ – The offset mapping is updated depending on the action $a_t$: For Copy and NoCopy actions, $O_{t+1}(b_{t+1})$ is assigned a valid offset, and for a Drop action $O_{t+1}(b_{t+1}) = \otimes$.

$I_{t+1}$ – If the action was Copy or NoCopy, $I_{t+1}(b_t)$ maps to the time interval the buffer occupies memory; and it maps to an empty interval if the action was Drop.

$W_{t+1}$– If the action was Copy, the system modifies $W_t$ by subtracting from it a vector $(u_0, ..., u_T)$ where $u_i$ corresponds to the data transfer time used by the assignment of $b_t$ at time step $i$.

$\mathscr{B}$ – The set of all buffers $\mathscr{B}$ does not change from state to state.

An example of how to determine the offset for the Copy action is as follows:

If $b$ is an input, then $I(b) = [s, \text{target_time}(b)]$ where $s$ is the latest logical time step, such that the constraints on the memory allocation are satisfied.

If $b$ is an output, then $I(b) = [\text{target_time}(b), e]$ where $e$ is the earliest logical time step, that the constraints on the memory allocation are satisfied.

In either case, $O(b)$ is chosen as the lowest offset, such that the CMEM offset range $[O(b), O(b) + \text{size}(b))$ is fully available across the full time interval $I(b)$, and the constraints on the memory allocation are satisfied.

An example of how to determine the offset for the NoCopy action is as follows:

If $b$ is an input, then $I(b) = (s, \text{target_time}(b)]$ where $s$ is the latest logical time step that lies within a time interval assigned to a buffer $b'$ with $\text{tensor_id}(b) = \text{tensor_id}(b')$.

If $b$ is an output, then $I(b) = \text{live_range}(b)$.

In either case, $O(b)$ is chosen in the same way as for Copy.

An example of determining the copy interval ($\text{copy}(b)$) for the action is as follows:

$\text{copy}(b) := [s, \text{target_time}(b))$ if $b$ is an input buffer placed using a Copy action.

$\text{copy}(b) := (\text{target_time}(b), e]$ if $b$ is an output buffer placed using a Copy action.

$\text{copy}(b)$ is the empty interval for any buffer placed with NoCopy or Drop actions.

In this example, the constraint on supply values and demand values can be expressed as:

$$\sum_{t \in Copy(b)} \text{supply}(\mathcal{I}_t) \geq \text{demand}(b).$$

Additionally, the constraint that there is only a single buffer being copied between memories at any given point in time can be expressed by requiring that the copy intervals of any two buffers have no internal intersections:

$$\forall b1, b2 : |\text{copy}(b1) \cap \text{copy}(b2)| \leq 1.$$

FIG. 4 shows an example 400 of the resource trade-offs that are represented by different memory allocation actions.

As in the example 300, the buffer $b_t$ is the current buffer being allocated. In the example 400, the buffer $b$' is a previous buffer that corresponds to the same buffer as the buffer $b_t$ and has already been committed to CMEM. As can be seen from the example 400, choosing the "Copy" action will allocate $b_t$ in CMEM and reduce available data transfer time in $W_{t+1}$ because the buffer $b_t$ will need to be copied from the HBM to CMEM. Choosing the "NoCopy" action will occupy CMEM for a longer time interval, extending the allocation interval of $b'$, but will not impact available data transfer. Choosing the "Drop" action will not impact memory nor data transfer, but may slow down the execution time of instruction $I_t$ to which the buffer $b_t$ corresponds, e.g., because the buffer will need to be retrieved from a slower memory.

FIG. 5 shows an example 500 of the architecture of the representation neural network.

In particular, in the example of FIG. 5, the representation neural network receives a current state input that includes (i) buffer features of the current buffer and optionally one or more additional buffers, (ii) a memory map grid feature, (iii) a memory profile feature, (iv), a supply profile feature, (v) action features, and (vi) global features.

In particular, in the example 500, the representation neural network is configured to process the memory map grid using an image encoder neural network to generate an encoded representation of the memory map grid and then generate a combined feature by combining the encoded representation with at least the feature of the buffer. In the example 500, the image encoder neural network is a convolutional neural network and, more specifically, a ResNet.

In the example 500, the representation neural network also processes the memory profile feature using a 1D convolutional encoder neural network to generate an encoded representation of the memory profile feature and processes the supply profile feature using a 1D convolutional encoder neural network to generate an encoded representation of the supply profile feature.

The representation neural network then generates the combined feature by concatenating (i) the buffer features, (ii) the encoded representation of the memory map grid feature, (iii) the encoded representation of the memory profile feature, (iv), the encoded representation of the supply profile feature, (v) the action features, and (vi) the global features.

The global features can include any of a variety of features that characterize the memory allocation problem. Examples include data identifying the current buffer index $t$, the index of $b_t$ in the order of buffers with the same alias_id, and the number of buffers remaining with the same alias_id.

The state representation can then process the combined feature using a combined feature encoder neural network to generate the state representation of the current state of the memory allocation. In the example 500, the combined feature encoder neural network is a multi-layer perceptron (MLP).

The example 500 also shows an example of the other learned components that are used by the system, e.g., to perform the tree search described above.

In particular, as shown in the example 500, the system also includes (i) a dynamics neural network that receives as input a state representation generated by the representation neural network and a memory allocation action and predicts (a) an output state representation that represents a state of the memory allocation if the input memory allocation action is performed when the memory allocation is in a state represented by the input state representation and (b) a reward that will be received if the input memory allocation action is performed when the memory allocation is in the state represented by the input state representation, a (ii) a prediction neural network that receives the input state representation and predicts a return that will be received starting from the state of the memory allocation is in a state represented by the input state representation (referred to as the "value neural network" in the example 500), and (iii) a policy neural network that receives the input state representation and predicts a probability distribution over the set of memory allocation actions.

More specifically, while the example 500 shows a separate reward neural network that predicts the reward that will be received if the input memory allocation action is performed when the memory allocation is in the state represented by the input state representation, the reward neural network can equivalently be implemented as part of the dynamics neural network, e.g., as a different neural network head within the dynamics neural network.

An example of performing a look ahead search using these components at a given time step now follows.

In particular, in the example, the state of the memory allocation at the given time step is represented by a root node in a state tree that includes nodes representing states of the memory allocation and edges representing memory allocation actions.

The system performs, using the state representation and starting from the root node in the state tree, a look-ahead search through the state tree.

For example, the system can perform a Monte Carlo Tree Search (MCTS) by, at each of multiple search iterations, traversing the state tree until a leaf node is reached. Once a leaf node is reached, the system can expand the leaf node and then terminate the search iteration. The system can continue performing search iterations until a budget for the tree search is exhausted.

During each search iteration, the system determines which edge to traverse from a given node using at least a probability distribution for the corresponding state generated by the policy neural network and the one or more returns generated by the one or more value neural networks.

For example, at each node, the system can select actions using an optimistic strategy called the predictor upper confidence tree bound (PUCT), meant to balance exploration (trying new actions) and exploitation (progressing further down the subtree of the current estimate of the best action). This strategy starts out by following the predicted policy closely, and gradually shifts towards maximizing the predicted value function.

In some implementations, the system evaluates the leaf node in the actual state space of the memory allocation. That is, the system maintains a memory allocation that has a state that corresponds to the state represented by the leaf node and determines which node is connected to the leaf node by a given edge by adding the corresponding action to the memory allocation.

In some other implementations, the system performs the evaluation in the (latent) state representation space. In these implementations, the system uses the dynamics neural network to generate a predicted state representation for the node that is connected to the leaf node by the given edge by processing an input that includes a state representation corresponding to the leaf node and identifies the corresponding action.

Performing these look ahead searches is described in Schrittwieser, et al., Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model, at arXiv: 1911.08265, Schrittwieser, et al, Online and Offline Reinforcement Learning by Planning with a Learned Model, at arXiv:2104.006294, and Hubert, et al., Learning and Planning in Complex Action Spaces, at arXiv:2104.06303.

As part of performing the look-ahead search, the system updates statistics for edges from the root node. For example, the system can maintain a visit count for each edge from the root node and can increment the visit count for a given edge from the root node whenever the edge is traversed during a search iteration.

After performing the search, the system selects, based on the statistics for edges from the root node after the look-ahead search is performed, an action from the set of actions.

For example, as described above, the system can maintain a visit count for each edge that represents the number of times that the edge was traversed during the look-ahead search. The system can then assign, to each edge, a probability that is proportional to the visit count for the edge and then sample an edge in accordance with the probabilities.

FIG. 6 shows an example 600 of the performance of the described techniques relative to other approaches. In particular, the example of FIG. 6 shows the reward achieved by the described techniques (MMap-MuZero) relative to two baselines techniques (evolutionary search and random policy) across time on the same hardware budget for four separate machine learning workloads. As can be seen from FIG. 6, the described techniques generally significantly outperform the random policy and the evolutionary search technique across all non-trivial time periods and across all four tasks.

This specification uses the terms ‘fast’, ‘slow’ in relation to memory. A memory or memory layer which can be accessed with low latency (e.g. less latency than another memory) may be referred to as a fast memory or fast memory layer, or as having a lower access time. A memory or memory layer which can only be accessed with high latency (e.g. more latency than another memory) may be referred to as a slow memory or slow memory layer, or as having a higher access time. The terms ‘fast’ and ‘slow’ are to be construed in relative terms, for example in relation to other memories and memory layers within the hierarchy. A fast memory may comprise memory with a high bandwidth. A slow memory may comprise memory with low bandwidth.

This specification uses the terms ‘large’ and ‘small’ in relation to memory. A memory or memory layer with a large memory capacity (e.g. with higher memory capacity than another memory layer) may be referred to a large memory or large memory layer. A memory or memory layer with a small memory capacity (e.g. with less memory capacity than another memory layer) may be referred to a small memory or small memory layer. The terms ‘large’ and ‘small’ are to be construed in relative terms, for example in relation to other memories and memory layers within the hierarchy.

This specification uses the term "configured" in connection with systems and computer program components. For a system of one or more computers to be configured to perform particular operations or actions means that the system has installed on it software, firmware, hardware, or a combination of them that in operation cause the system to perform the operations or actions. For one or more computer programs to be configured to perform particular operations or actions means that the one or more programs include instructions that, when executed by data processing apparatus, cause the apparatus to perform the operations or actions.

Embodiments of the subject matter and the functional operations described in this specification can be implemented in digital electronic circuitry, in tangibly-embodied computer software or firmware, in computer hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them. Embodiments of the subject matter described in this specification can be implemented as one or more computer programs, e.g., one or more modules of computer program instructions encoded on a tangible non transitory storage medium for execution by, or to control the operation of, data processing apparatus. The computer storage medium can be a machine-readable storage device, a machine-readable storage substrate, a random or serial access memory device, or a combination of one or more of them. Alternatively or in addition, the program instructions can be encoded on an artificially generated propagated signal, e.g., a machine-generated electrical, optical, or electromagnetic signal, that is generated to encode information for transmission to suitable receiver apparatus for execution by a data processing apparatus.

The term "data processing apparatus" refers to data processing hardware and encompasses all kinds of apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can also be, or further include, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit). The apparatus can optionally include, in addition to hardware, code that creates an execution environment for computer programs, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, or a combination of one or more of them.

A computer program, which may also be referred to or described as a program, software, a software application, an app, a module, a software module, a script, or code, can be written in any form of programming language, including compiled or interpreted

languages, or declarative or procedural languages; and it can be deployed in any form, including as a stand alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A program may, but need not, correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data, e.g., one or more scripts stored in a markup language document, in a single file dedicated to the program in question, or in multiple coordinated files, e.g., files that store one or more modules, sub programs, or portions of code. A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a data communication network.

In this specification, the term "database" is used broadly to refer to any collection of data: the data does not need to be structured in any particular way, or structured at all, and it can be stored on storage devices in one or more locations. Thus, for example, the index database can include multiple collections of data, each of which may be organized and accessed differently.

Similarly, in this specification the term "engine" is used broadly to refer to a software-based system, subsystem, or process that is programmed to perform one or more specific functions. Generally, an engine will be implemented as one or more software modules or components, installed on one or more computers in one or more locations. In some cases, one or more computers will be dedicated to a particular engine; in other cases, multiple engines can be installed and running on the same computer or computers.

The processes and logic flows described in this specification can be performed by one or more programmable computers executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows can also be performed by special purpose logic circuitry, e.g., an FPGA or an ASIC, or by a combination of special purpose logic circuitry and one or more programmed computers.

Computers suitable for the execution of a computer program can be based on general or special purpose microprocessors or both, or any other kind of central processing unit. Generally, a central processing unit will receive instructions and data from a read only memory or a random access memory or both. The essential elements of a computer are a central processing unit for performing or executing instructions and one or more memory devices for storing instructions and data. The central processing unit and the memory can be supplemented by, or incorporated in, special purpose logic circuitry. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or

optical disks. However, a computer need not have such devices. Moreover, a computer can be embedded in another device, e.g., a mobile telephone, a personal digital assistant (PDA), a mobile audio or video player, a game console, a Global Positioning System (GPS) receiver, or a portable storage device, e.g., a universal serial bus (USB) flash drive, to name just a few.

Computer readable media suitable for storing computer program instructions and data include all forms of non volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and CD ROM and DVD-ROM disks.

To provide for interaction with a user, embodiments of the subject matter described in this specification can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input. In addition, a computer can interact with a user by sending documents to and receiving documents from a device that is used by the user; for example, by sending web pages to a web browser on a user's device in response to requests received from the web browser. Also, a computer can interact with a user by sending text messages or other forms of message to a personal device, e.g., a smartphone that is running a messaging application, and receiving responsive messages from the user in return.

Data processing apparatus for implementing machine learning models can also include, for example, special-purpose hardware accelerator units for processing common and compute-intensive parts of machine learning training or production, e.g., inference, workloads.

Machine learning models can be implemented and deployed using a machine learning framework, .e.g., a TensorFlow framework or a Jax framework.

Embodiments of the subject matter described in this specification can be implemented in a computing system that includes a back end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front end component, e.g., a client computer having a graphical user interface, a web browser, or an app through which a user can interact with an implementation of the subject matter described

in this specification, or any combination of one or more such back end, middleware, or front end components. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network (LAN) and a wide area network (WAN), e.g., the Internet.

The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other. In some embodiments, a server transmits data, e.g., an HTML page, to a user device, e.g., for purposes of displaying data to and receiving user input from a user interacting with the device, which acts as a client. Data generated at the user device, e.g., a result of the user interaction, can be received at the server from the device.

While this specification contains many specific implementation details, these should not be construed as limitations on the scope of any invention or on the scope of what may be claimed, but rather as descriptions of features that may be specific to particular embodiments of particular inventions. Certain features that are described in this specification in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially be claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or variation of a subcombination.

Similarly, while operations are depicted in the drawings and recited in the claims in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system modules and components in the embodiments described above should not be understood as requiring such separation in all embodiments, and it should be understood that the described program components and systems can generally be integrated together in a single software product or packaged into multiple software products.

Particular embodiments of the subject matter have been described. Other embodiments are within the scope of the following claims. For example, the actions recited in the claims can be performed in a different order and still achieve desirable results. As one example, the processes depicted in the accompanying figures do not necessarily require the particular order shown, or sequential order, to achieve desirable results. In some cases, multitasking and parallel processing may be advantageous.

Aspects of the present disclosure may be as set out in the following clauses:

Clause 1. A method performed by one or more computers, the method comprising:

receiving data specifying a sequence of buffers that are associated with a target program to be executed on a target computer system, the target computer system comprising a plurality of memories, the program comprising a plurality of instructions, and each buffer in the sequence specifying a respective operand or a respective output of one of the plurality of instructions; and

generating a memory allocation for the target program that allocates each of the buffers across the plurality of memories during execution of the target program on the target computer system, the generating comprising allocating each buffer according to the sequence, comprising, for each buffer:

generating a current state input specifying a current state of the memory allocation as of a position of the buffer in the sequence,

processing the current state input using a representation neural network that is configured to process the current state input to generate a state representation of the current state of the memory allocation; and

selecting, using the state representation, a memory allocation action for the buffer from a set of memory allocation actions, wherein each memory allocation action allocates the buffer differently across the plurality of memories during the execution of the target program on the target computer system.

Clause 2. The method of clause 1, wherein the buffers are ordered in the sequence chronologically with respect to appearance of each buffer within the target program.

Clause 3. The method of clause 1 or clause 2, further comprising:

compiling the target program to generate an executable program that, when executed by the target computer system, causes the target computer system to allocate the buffers

across the plurality of memories in accordance with the memory allocation while executing the target program.

Clause 4. The method of clause 3, further comprising:

causing the target computer system to execute the executable machine code to perform the target algorithm.

Clause 5. The method of any preceding clause, wherein the plurality of memories comprise a first memory associated with a first level of a memory hierarchy and a second memory associated with a second, different level of the memory hierarchy.

Clause 6. The method of clause 5, wherein the first memory has a smaller capacity than the second memory and the first memory has a lower access time than the second memory.

Clause 7. The method of clause 5 or clause 6, wherein the set of memory allocation actions comprises a first memory allocation action that assigns the buffer to the first memory for a specified time interval.

Clause 8. The method of clause 7, wherein the specified time interval covers data transfer time of the buffer between the second memory and the first memory.

Clause 9. The method of any one of clauses 7 or 8, wherein the set of memory allocation actions comprises a second memory allocation action that, when the buffer specifies a same tensor as an earlier buffer that is earlier in the sequence, assigns the buffer to a same allocation in the first memory as the earlier buffer and extends the specified time interval for the earlier buffer to include a logical time at which an instruction corresponding to the buffer is executed by the target program.

Clause 10. The method of any one of clauses 7-9, wherein the set of memory allocation actions comprises a third action that assigns the buffer to the second memory without allocating the buffer to the first memory.

Clause 11. The method of any preceding clause, wherein the current state input comprises features of the buffer.

Clause 12. The method of clause 11, wherein the current state input comprises a memory map grid that specifies a usage of at least one of the plurality of memories in a local neighborhood around a logical time at which an instruction corresponding to the buffer is executed by the target program.

Clause 13. The method of clause 11 or clause 12, wherein the current state input comprises a memory occupancy feature that specifies a usage of each of at least one of the plurality of memories at a logical time at which an instruction corresponding to the buffer is executed by the target program.

Clause 14. The method of any one of clauses 11-13, wherein the current state input comprises a supply profile feature that provides information about data transfer times of buffers between two or more of the plurality of memories.

Clause 15. The method of any one of clauses 11-14, wherein the current state input comprises a respective feature for each of the memory allocation actions.

Clause 16. The method of any one of clauses 11-15, wherein the current state input comprises data characterizing other buffers with a same alias as the buffer.

Clause 17. The method of any one of clauses 12-16 when dependent upon clause 12, wherein the representation neural network is configured to:

process the memory map grid using an image encoder neural network to generate an encoded representation of the memory map grid;

generate a combined feature by combining the encoded representation with at least the feature of the buffer; and

process the combined feature using a combined feature encoder neural network to generate the state representation of the current state of the memory allocation.

Clause 18. The method of any preceding clause, wherein selecting a memory allocation action for the buffer from a set of memory allocation actions comprises:

performing, using the state representation generated by the neural network and starting from a root node representing the current state, a look-ahead search through a state tree that includes nodes representing states of the memory allocations and edges representing

the memory allocation actions; and

selecting, based on statistics for edges from the root node after the look-ahead search is performed, the memory allocation action from the set of memory allocation actions.

Clause 19. The method of any preceding clause, wherein performing, using the state representation generated by the neural network and starting from a root node representing the current state, a look-ahead search through a state tree that includes nodes representing states of the memory allocations and edges representing the memory allocation actions comprises:

performing the look-ahead search using:

(i) a dynamics neural network that receives as input a state representation and a memory allocation action and predicts an (i) output state representation that represents a state of the memory allocation if the input memory allocation action is performed when the memory allocation is in a state represented by the input state representation and a (ii) reward that will be received if the input memory allocation action is performed when the memory allocation is in the state represented by the input state representation,

(ii) a prediction neural network that receives the input state representation and predicts a return that will be received starting from the state of the memory allocation that is represented by the input state representation, and

(iii) a policy neural network that receives the input state representation and predicts a probability distribution over the set of memory allocation actions.

Clause 20. The method of clause 19, wherein the reward for performing a given memory allocation action when the memory allocation is in a given state represented by a given state representation is an expected speedup achieved as a result of performing the given memory allocation when the memory allocation is in the given state.

Clause 21. The method of any preceding clause, wherein the representation neural network has been trained through reinforcement learning to maximize a return that estimates a latency of the computer program when executed on the target computer system in accordance with the memory allocation.

Clause 22. The method of clause 21, wherein, for a given sequence of buffers associated with a given program, the return is a combination of respective rewards for the selected

memory allocation actions for the buffers in the given sequence.

Clause 23. The method of clause 22, wherein the respective reward for the selected memory allocation action for a given buffer in a given sequence is an expected speedup, of executing the given program on the computer system, achieved as a result of performing the memory allocation action given the selected memory allocation actions for the buffers that precede the given buffer in the given sequence.

Clause 24. The method of clause 23, wherein the expected speedup for the given buffer is computed based on executing the instructions in the given program on the target computer system.

Clause Clause 25. The method of clause 23 or clause 24 when dependent on clause 6, wherein the expected speedup for assigning the given buffer to the second memory is zero and the expected speedup for assigning the given buffer to the first memory is equal to an expected speedup of accessing the given buffer from the first memory instead from the second memory.

26. The method of any one of clauses 21-25 when dependent on clause 10, further comprising generating a training example for training the state representation neural network, comprising:

determining, while allocating the buffers in the sequence, that the memory allocation has entered an infeasible state;

in response, identifying a backup state at which no future buffer shares a same alias identifier with any already placed buffer that has been allocated to the first memory and generating a training example that, for each buffer to be allocated after the backup state, selects the third memory allocation action that allocates the buffer to the second memory.

Clause 27. The method of any preceding clause, further comprising:

determining a second memory allocation for the target program using a different, heuristic-based solver;

determining a first expected latency for the memory allocation and a second expected latency for the second memory allocation; and

selecting the memory allocation as a final memory allocation for the target program

only when the first expected latency is lower than the second expected latency.

Clause 28. The method of any preceding clause when dependent on clause 6, wherein, for each memory allocation action that assigns a given buffer to the first memory, the memory allocation action defines (i) an offset within the first memory at which to store the given buffer and (ii) a time interval for which to store the given buffer in the first memory.

Clause 29. A system comprising:

one or more computers; and

one or more storage devices storing instructions that, when executed by the one or more computers, cause the one or more computers to perform the respective operations of any one of clauses 1-28.

Clause 30. One or more computer-readable storage media storing instructions that when executed by one or more computers cause the one or more computers to perform the respective operations of the method of any one of clauses 1-28.

# CLAIMS

1. A method performed by one or more computers, the method comprising:

receiving data specifying a sequence of buffers that are associated with a target program to be executed on a target computer system, the target computer system comprising a plurality of memories, the program comprising a plurality of instructions, and each buffer in the sequence specifying a respective operand or a respective output of one of the plurality of instructions; and

generating a memory allocation for the target program that allocates each of the buffers across the plurality of memories during execution of the target program on the target computer system, the generating comprising allocating each buffer according to the sequence, comprising, for each buffer:

generating a current state input specifying a current state of the memory allocation as of a position of the buffer in the sequence,

processing the current state input using a representation neural network that is configured to process the current state input to generate a state representation of the current state of the memory allocation; and

selecting, using the state representation, a memory allocation action for the buffer from a set of memory allocation actions, wherein each memory allocation action allocates the buffer differently across the plurality of memories during the execution of the target program on the target computer system.

2. The method of claim 1, wherein the buffers are ordered in the sequence chronologically with respect to appearance of each buffer within the target program.

3. The method of claim 1 or claim 2, further comprising:

compiling the target program to generate an executable program that, when executed by the target computer system, causes the target computer system to allocate the buffers across the plurality of memories in accordance with the memory allocation while executing the target program.

4. The method of claim 3, further comprising:

causing the target computer system to execute the executable machine code to perform the target algorithm.

5. The method of any preceding claim, wherein the plurality of memories comprise a first memory associated with a first level of a memory hierarchy and a second memory associated with a second, different level of the memory hierarchy.

6. The method of claim 5, wherein the first memory has a smaller capacity than the second memory and the first memory has a lower access time than the second memory.

7. The method of claim 5 or claim 6, wherein the set of memory allocation actions comprises a first memory allocation action that assigns the buffer to the first memory for a specified time interval.

8. The method of claim 7, wherein the specified time interval covers data transfer time of the buffer between the second memory and the first memory.

9. The method of any one of claims 7 or 8, wherein the set of memory allocation actions comprises a second memory allocation action that, when the buffer specifies a same tensor as an earlier buffer that is earlier in the sequence, assigns the buffer to a same allocation in the first memory as the earlier buffer and extends the specified time interval for the earlier buffer to include a logical time at which an instruction corresponding to the buffer is executed by the target program.

10. The method of any one of claims 7-9, wherein the set of memory allocation actions comprises a third action that assigns the buffer to the second memory without allocating the buffer to the first memory.

11. The method of any preceding claim, wherein the current state input comprises features of the buffer.

12. The method of claim 11, wherein the current state input comprises a memory map grid that specifies a usage of at least one of the plurality of memories in a local neighborhood around a logical time at which an instruction corresponding to the buffer is executed by the target program.

13. The method of claim 11 or claim 12, wherein the current state input comprises a memory occupancy feature that specifies a usage of each of at least one of the plurality of memories at a logical time at which an instruction corresponding to the buffer is executed by the target program.

14. The method of any one of claims 11-13, wherein the current state input comprises a supply profile feature that provides information about data transfer times of buffers between two or more of the plurality of memories.

15. The method of any one of claims 11-14, wherein the current state input comprises a respective feature for each of the memory allocation actions.

16. The method of any one of claims 11-15, wherein the current state input comprises data characterizing other buffers with a same alias as the buffer.

17. The method of any one of claims 12-16 when dependent upon claim 12, wherein the representation neural network is configured to:

process the memory map grid using an image encoder neural network to generate an encoded representation of the memory map grid;

generate a combined feature by combining the encoded representation with at least the feature of the buffer; and

process the combined feature using a combined feature encoder neural network to generate the state representation of the current state of the memory allocation.

18. The method of any preceding claim, wherein selecting a memory allocation action for the buffer from a set of memory allocation actions comprises:

performing, using the state representation generated by the neural network and starting from a root node representing the current state, a look-ahead search through a state tree that includes nodes representing states of the memory allocations and edges representing the memory allocation actions; and

selecting, based on statistics for edges from the root node after the look-ahead search is performed, the memory allocation action from the set of memory allocation actions.

19. The method of any preceding claim, wherein performing, using the state representation generated by the neural network and starting from a root node representing the current state, a look-ahead search through a state tree that includes nodes representing states of the memory allocations and edges representing the memory allocation actions comprises:

performing the look-ahead search using:

(i) a dynamics neural network that receives as input a state representation and a memory allocation action and predicts an (i) output state representation that represents a state of the memory allocation if the input memory allocation action is performed when the memory allocation is in a state represented by the input state representation and a (ii) reward that will be received if the input memory allocation action is performed when the memory allocation is in the state represented by the input state representation,

(ii) a prediction neural network that receives the input state representation and predicts a return that will be received starting from the state of the memory allocation that is represented by the input state representation, and

(iii) a policy neural network that receives the input state representation and predicts a probability distribution over the set of memory allocation actions.

20. The method of claim 19, wherein the reward for performing a given memory allocation action when the memory allocation is in a given state represented by a given state representation is an expected speedup achieved as a result of performing the given memory allocation when the memory allocation is in the given state.

21. The method of any preceding claim, wherein the representation neural network has been trained through reinforcement learning to maximize a return that estimates a latency of the computer program when executed on the target computer system in accordance with the memory allocation.

22. The method of claim 21, wherein, for a given sequence of buffers associated with a given program, the return is a combination of respective rewards for the selected memory allocation actions for the buffers in the given sequence.

23. The method of claim 22, wherein the respective reward for the selected memory allocation action for a given buffer in a given sequence is an expected speedup, of executing the given program on the computer system, achieved as a result of performing the memory

allocation action given the selected memory allocation actions for the buffers that precede the given buffer in the given sequence.

24. The method of claim 23, wherein the expected speedup for the given buffer is computed based on executing the instructions in the given program on the target computer system.

25. The method of claim 23 or claim 24 when dependent on claim 6, wherein the expected speedup for assigning the given buffer to the second memory is zero and the expected speedup for assigning the given buffer to the first memory is equal to an expected speedup of accessing the given buffer from the first memory instead from the second memory.

26. The method of any one of claims 21-25 when dependent on claim 10, further comprising generating a training example for training the state representation neural network, comprising:

determining, while allocating the buffers in the sequence, that the memory allocation has entered an infeasible state;

in response, identifying a backup state at which no future buffer shares a same alias identifier with any already placed buffer that has been allocated to the first memory and generating a training example that, for each buffer to be allocated after the backup state, selects the third memory allocation action that allocates the buffer to the second memory.

27. The method of any preceding claim, further comprising:

determining a second memory allocation for the target program using a different, heuristic-based solver;

determining a first expected latency for the memory allocation and a second expected latency for the second memory allocation; and

selecting the memory allocation as a final memory allocation for the target program only when the first expected latency is lower than the second expected latency.

28. The method of any preceding claim when dependent on claim 6, wherein, for each memory allocation action that assigns a given buffer to the first memory, the memory

allocation action defines (i) an offset within the first memory at which to store the given buffer and (ii) a time interval for which to store the given buffer in the first memory.

29. A system comprising:

one or more computers; and

one or more storage devices storing instructions that, when executed by the one or more computers, cause the one or more computers to perform the respective operations of the method of any one of claims 1-28.

30. One or more computer-readable storage media storing instructions that when executed by one or more computers cause the one or more computers to perform the respective operations of the method of any one of claims 1-28.

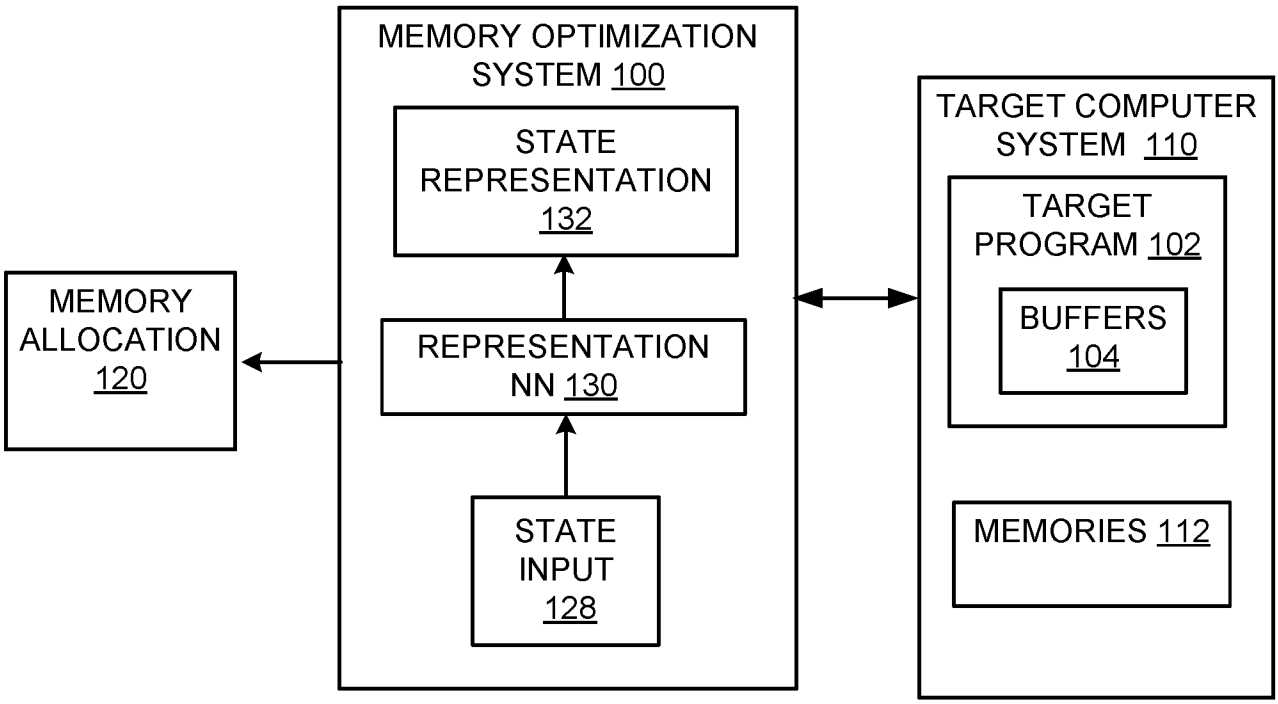

MEMORY OPTIMIZATION SYSTEM 100
STATE REPRESENTATION 132
REPRESENTATION NN 130
STATE INPUT 128
MEMORY ALLOCATION 120
TARGET COMPUTER SYSTEM 110
TARGET PROGRAM 102
BUFFERS 104
MEMORIES 112

FIG. 1

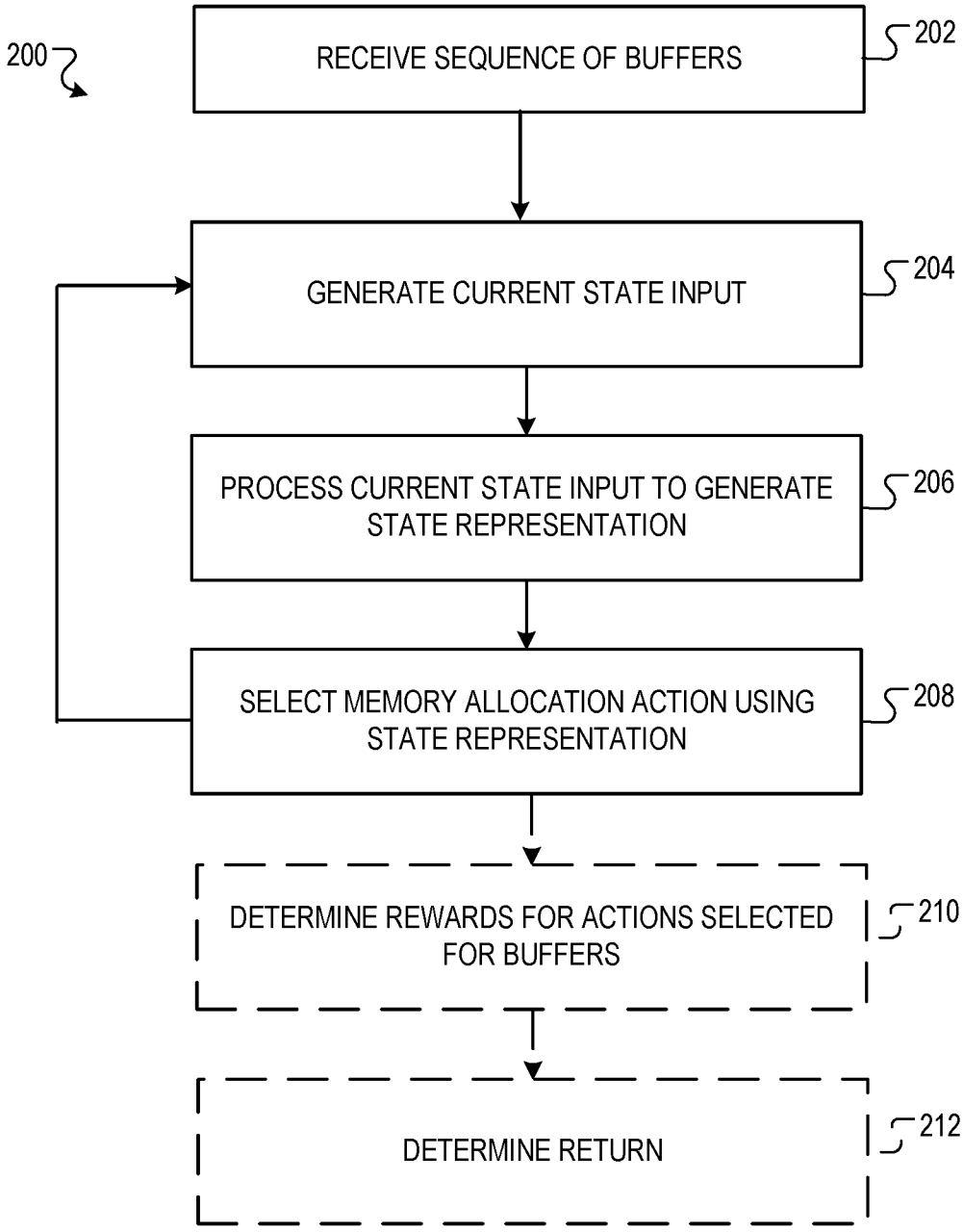

200
RECEIVE SEQUENCE OF BUFFERS
202
GENERATE CURRENT STATE INPUT
204
PROCESS CURRENT STATE INPUT TO GENERATE STATE REPRESENTATION
206
SELECT MEMORY ALLOCATION ACTION USING STATE REPRESENTATION
208
DETERMINE REWARDS FOR ACTIONS SELECTED FOR BUFFERS
210
DETERMINE RETURN
212

FIG. 2

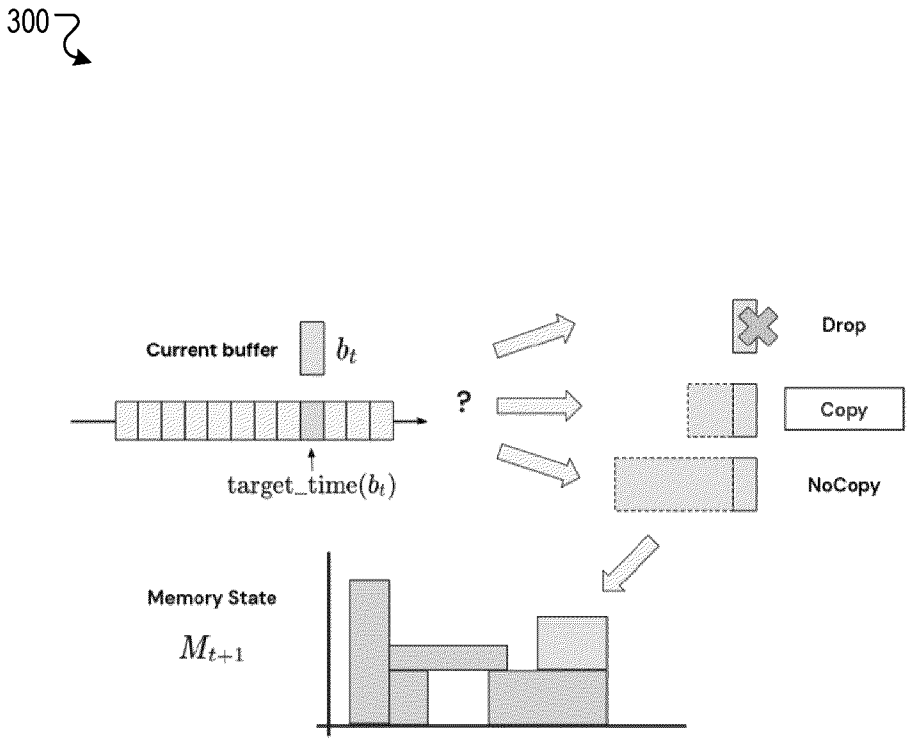

300
Current buffer
$b_t$
?
target_time($b_t$)
Drop
Copy
NoCopy
Memory State
$M_{t+1}$

FIG. 3



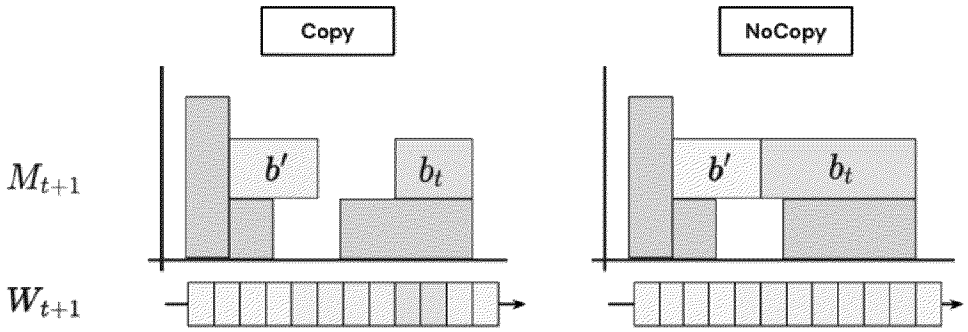




FIG. 4

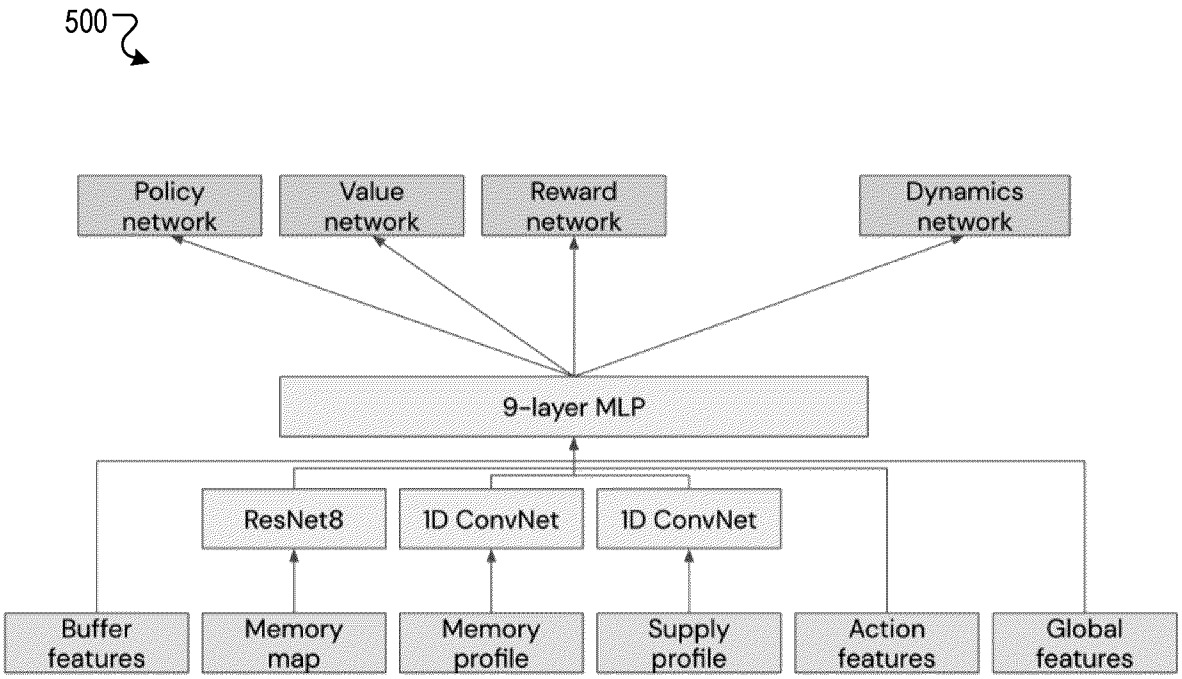

500
Policy network
Value network
Reward network
Dynamics network
9-layer MLP
ResNet8
1D ConvNet
1D ConvNet
Buffer features
Memory map
Memory profile
Supply profile
Action features
Global features

FIG. 5

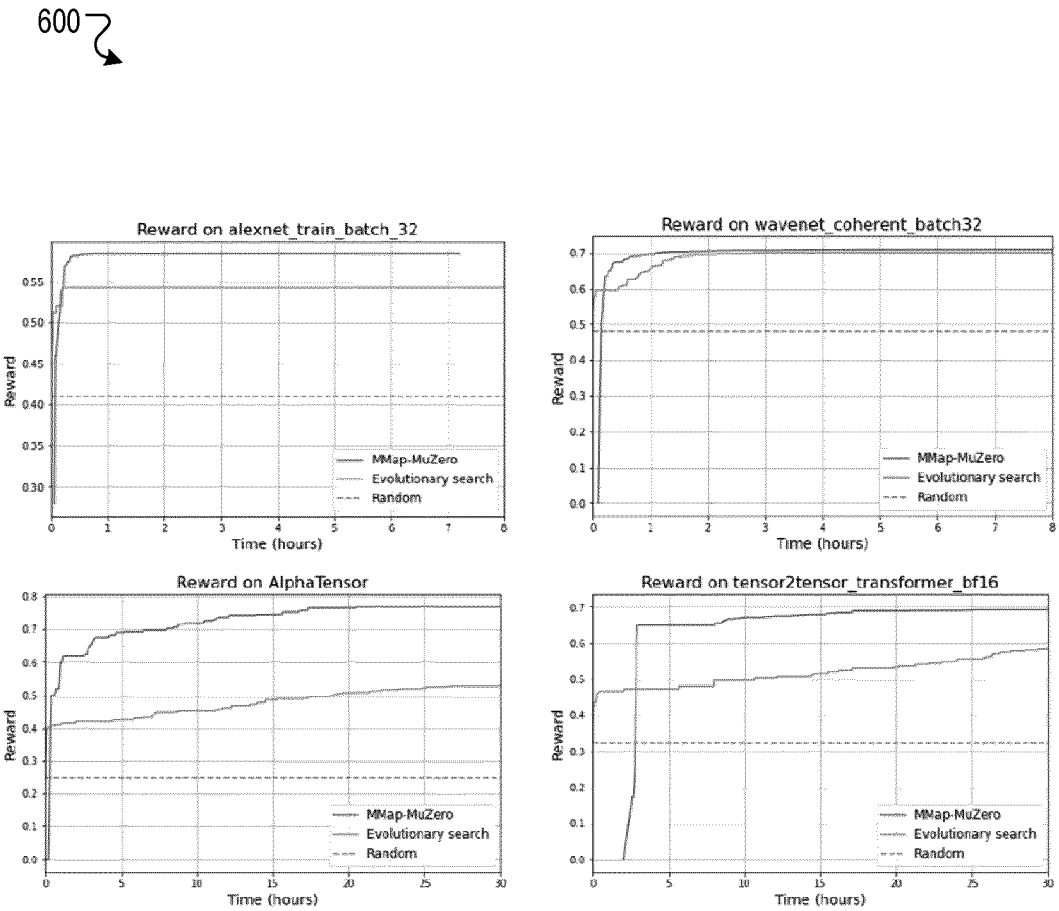

600
Reward on alexnet_train_batch_32
Reward on wavenet_coherent_batch32
Reward on AlphaTensor
Reward on tensor2tensor_transformer_bf16
Reward
Time (hours)
MMap-MuZero
Evolutionary search
Random

FIG. 6

# INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2024/056811

**A. CLASSIFICATION OF SUBJECT MATTER**
INV. G06F9/50 G06N3/02 G06N3/092
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

**B. FIELDS SEARCHED**

Minimum documentation searched (classification system followed by classification symbols)
G06F G06N

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, INSPEC

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | HILDEBRAND MARK MHILDEBRAND@UCDAVIS EDU ET AL: "AutoTM: Automatic Tensor Movement in Heterogeneous Memory Systems using Integer Linear Programming", PROCEEDINGS OF THE TWENTY-FIFTH INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, ACM, NEW YORK, NY, USA, 9 March 2020 (2020-03-09), pages 875-890, XP058460449, DOI: 10.1145/3373376.3378465 ISBN: 978-1-4503-7102-5 the whole document<br>-----<br>-/-- | 1-30 |

[x] Further documents are listed in the continuation of Box C.
[ ] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance;; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance;; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 28 May 2024 | 12/06/2024 |

| Name and mailing address of the ISA/ | Authorized officer |
|---|---|
| European Patent Office, P.B. 5818 Patentlaan 2<br>NL - 2280 HV Rijswijk<br>Tel. (+31-70) 340-2040,<br>Fax: (+31-70) 340-3016 | Touloupis, Emmanuel |

1

Form PCT/ISA/210 (second sheet) (April 2005)

# INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2024/056811

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| A | DATABASE INSPEC [Online]<br>THE INSTITUTION OF ELECTRICAL ENGINEERS, STEVENAGE, GB;<br>February 1999 (1999-02),<br>CAVALIERI S: "Solving linear integer programming problems by a novel neural model",<br>XP002811640,<br>Database accession no. 6312084<br>abstract<br>& INTERNATIONAL JOURNAL OF NEURAL SYSTEMS WORLD SCIENTIFIC SINGAPORE,<br>vol. 9, no. 1,<br>1 February 1999 (1999-02-01), pages 27-39,<br>ISSN: 0129-0657 | 1-30 |
| A | NASIRA G M ET AL: "A Backpropagation neural network implementation for hybrid algorithm in solving Integer Linear Programming Problems",<br>COMPUTING COMMUNICATION AND NETWORKING TECHNOLOGIES (ICCCNT), 2010 INTERNATIONAL CONFERENCE ON, IEEE, PISCATAWAY, NJ, USA,<br>29 July 2010 (2010-07-29), pages 1-6,<br>XP031768094,<br>ISBN: 978-1-4244-6591-0<br>abstract | 1-30 |
| A | ARJMANDZADEH ZIBA ET AL: "A new neural network model for solving random interval linear programming problems",<br>NEURAL NETWORKS, ELSEVIER SCIENCE PUBLISHERS, BARKING, GB,<br>vol. 89, 9 February 2017 (2017-02-09), pages 11-18, XP029968071,<br>ISSN: 0893-6080, DOI: 10.1016/J.NEUNET.2016.12.007<br>abstract | 1-30 |
| A | SHAUHARDA KHADKA: "Optimizing Memory Placement using Evolutionary Graph Reinforcement Learning",<br>, [Online]<br>4 May 2021 (2021-05-04), pages 1-17,<br>XP093165892,<br>Retrieved from the Internet:<br>URL:https://openreview.net/pdf?id=-6vS_4Kfz0><br>[retrieved on 2024-05-27]<br>the whole document | 1-30 |

1