syntax = "proto3";

package tensorflow;

import "tensorflow/core/framework/attr\_value.proto";

import "tensorflow/core/protobuf/verifier\_config.proto";

option cc\_enable\_arenas = true;

option java\_outer\_classname = "RewriterConfigProtos";

option java\_multiple\_files = true;

option java\_package = "org.tensorflow.framework";

option go\_package = "github.com/tensorflow/tensorflow/tensorflow/go/core/protobuf/for\_core\_protos\_go\_proto";

message AutoParallelOptions {

bool enable = 1;

int32 num\_replicas = 2;

}

message ScopedAllocatorOptions {

// If present, only perform optimization for these ops.

repeated string enable\_op = 1;

}

message RewriterConfig {

// Graph rewriting is experimental and subject to change, not covered by any

// API stability guarantees.

// Configuration options for the meta-optimizer. Unless otherwise noted, these

// configuration options do not apply to explicitly triggered optimization

// passes in the optimizers field.

enum Toggle {

DEFAULT = 0;

ON = 1;

OFF = 2;

// Enable some aggressive optimizations that use assumptions that TF graphs

// may break. For example, assume the shape of a placeholder matches its

// actual feed.

AGGRESSIVE = 3;

}

// Enum for layout conversion between NCHW and NHWC on CPU. Default is OFF.

enum CpuLayout {

NO\_CONVERSION\_ON\_CPU = 0;

NCHW\_TO\_NHWC = 1;

NHWC\_TO\_NCHW = 2;

}

// Enum controlling the number of times to run optimizers. The default is to

// run them twice.

enum NumIterationsType {

DEFAULT\_NUM\_ITERS = 0;

ONE = 1;

TWO = 2;

}

// CPU Conversion settings between NHCW and NCHW.

CpuLayout cpu\_layout\_conversion = 50;

// Optimize tensor layouts (default is ON)

// e.g. This will try to use NCHW layout on GPU which is faster.

Toggle layout\_optimizer = 1;

// Fold constants (default is ON)

// Statically infer the value of tensors when possible, and materialize the

// result using constants.

Toggle constant\_folding = 3;

// Shape optimizations (default is ON)

// Simplify computations made on shapes.

Toggle shape\_optimization = 13;

// Remapping (default is ON)

// Remap subgraphs onto more efficient implementations.

Toggle remapping = 14;

// Common subgraph elimination (default is ON)

// e.g. Simplify arithmetic ops; merge ops with same value (like constants).

Toggle common\_subgraph\_elimination = 24;

// Arithmetic optimizations (default is ON)

// e.g. Simplify arithmetic ops; merge ops with same value (like constants).

Toggle arithmetic\_optimization = 7;

// Control dependency optimizations (default is ON).

// Remove redundant control dependencies, which may enable other optimization.

Toggle dependency\_optimization = 8;

// Loop optimizations (default is ON).

Toggle loop\_optimization = 9;

// Function optimizations (default is ON).

Toggle function\_optimization = 10;

// Strips debug-related nodes from the graph (off by default).

Toggle debug\_stripper = 11;

// If true, don't remove unnecessary ops from the graph

bool disable\_model\_pruning = 2;

// Try to allocate some independent Op outputs contiguously in order to

// merge or eliminate downstream Ops (off by default).

Toggle scoped\_allocator\_optimization = 15;

// Force small ops onto the CPU (default is OFF).

Toggle pin\_to\_host\_optimization = 18;

// Enable the swap of kernel implementations based on the device placement

// (default is ON).

Toggle implementation\_selector = 22;

// Optimize data types for CUDA (default is OFF).

// This will try to use float16 on GPU which is faster.

// Note that this can change the numerical stability of the graph and may

// require the use of loss scaling to maintain model convergence.

Toggle auto\_mixed\_precision = 23;

// Optimize data types for MKL (default is OFF).

// This will try to use bfloat16 on CPUs, which is faster.

// Note that this can change the numerical stability of the graph.

Toggle auto\_mixed\_precision\_mkl = 25;

// Emulate a model using data type float16 on CPU (default is OFF).

// This will try to emulate the float16 inputs and outputs of an operator

// on CPU to have better correlation with float16 on GPU; however the

// computation in the operator is based on float32.

// Note that this can change the numerical stability of the graph.

Toggle auto\_mixed\_precision\_cpu = 29;

// Disable the entire meta optimizer (off by default).

bool disable\_meta\_optimizer = 19;

// Optimizers registered by plugin (default is ON)

Toggle use\_plugin\_optimizers = 28;

// Controls how many times we run the optimizers in meta optimizer (default

// is once).

NumIterationsType meta\_optimizer\_iterations = 12;

// The minimum number of nodes in a graph to optimizer. For smaller graphs,

// optimization is skipped.

// 0 means the system picks an appropriate number.

// < 0 means do not skip optimization.

int32 min\_graph\_nodes = 17;

// Disable optimizations that assume compressed tensors. Note that this flag

// is experimental and may be removed in the future.

bool experimental\_disable\_compressed\_tensor\_optimization = 26;

// Disable folding quantization emulation ops such as FakeQuantWithMinMax\* and

// QuantizeAndDequantize\*. Some compilers (e.g. the TF-to-tflite converter)

// have to extract quantization configs (e.g. min/max range, number of bits,

// and per-channel) from the quantization emulation ops. Note that this flag

// is experimental and may be removed in the future. See b/174138564 for more

// details.

bool experimental\_disable\_folding\_quantization\_emulation = 27;

enum MemOptType {

// The default setting (SCHEDULING and SWAPPING HEURISTICS only)

DEFAULT\_MEM\_OPT = 0;

// Disabled in the meta-optimizer.

NO\_MEM\_OPT = 1;

// Driven by manual op-level annotations.

MANUAL = 2;

// Driven by heuristics. The behavior of these heuristics is subject to

// change. Currently includes an experimental recomputation and swapping

// heuristics. Manual annotations are respected, but additional nodes are

// selected automatically.

// Swapping heuristic will move a tensor from the GPU to the CPU and move

// it back when needed to reduce peak memory usage.

SWAPPING\_HEURISTICS = 4;

// Recomputation heuristics will recompute ops (such as Relu activation)

// during backprop instead of storing them, reducing peak memory usage.

RECOMPUTATION\_HEURISTICS = 5;

// Scheduling will split big ops such as AddN and try to enforce a schedule

// of the new computations that decreases peak memory usage.

SCHEDULING\_HEURISTICS = 6;

// Use any combination of swapping and recomputation heuristics.

HEURISTICS = 3;

}

// Configures memory optimization passes through the meta-optimizer. Has no

// effect on manually requested memory optimization passes in the optimizers

// field.

MemOptType memory\_optimization = 4;

// A node name scope for node names which are valid outputs of recomputations.

// Inputs to nodes that match this scope may be recomputed (subject either to

// manual annotation of those input nodes or to manual annotation and

// heuristics depending on memory\_optimization), but the nodes themselves will

// not be recomputed. This matches any sub-scopes as well, meaning the scope

// can appear not just as a top-level scope. For example, if the value is

// "gradients/", the default, it will match node name "gradients/foo",

// "foo/gradients/bar", but not "foo\_gradients/"

string memory\_optimizer\_target\_node\_name\_scope = 6;

// Maximum number of milliseconds to spend optimizing a single graph before

// timing out. If less than or equal to 0 (default value) the optimizer will

// never time out.

int64 meta\_optimizer\_timeout\_ms = 20;

// Configures AutoParallel optimization passes either through the

// meta-optimizer or when manually specified through the optimizers field.

AutoParallelOptions auto\_parallel = 5;

// If true, any optimization pass failing will cause the MetaOptimizer to

// stop with an error. By default - or when set to false, failing passes are

// skipped silently.

bool fail\_on\_optimizer\_errors = 21;

ScopedAllocatorOptions scoped\_allocator\_opts = 16;

// If non-empty, will use this as an alternative way to specify a list of

// optimizations to turn on and the order of the optimizations (replacing the

// meta-optimizer).

//

// Of the RewriterConfig options, only the AutoParallel configuration options

// (the auto\_parallel field) apply to manually requested optimization passes

// ("autoparallel"). Memory optimization passes ("memory") invoked here are

// not configurable (in contrast to memory optimization passes through the

// meta-optimizer) and act only on manual op annotations.

//

// Custom optimizers (see custom\_optimizers) that are not part of this

// schedule will be run after - in the order that they were specified.

repeated string optimizers = 100;

// Message to describe custom graph optimizer and its parameters

message CustomGraphOptimizer {

string name = 1;

map<string, AttrValue> parameter\_map = 2;

}

// list of CustomGraphOptimizers to apply.

repeated CustomGraphOptimizer custom\_optimizers = 200;

// VerifierConfig specifying the verifiers to be run after every optimizer.

VerifierConfig inter\_optimizer\_verifier\_config = 300;

// VerifierConfig specifying the verifiers to be run at the end, after all

// optimizers have run.

VerifierConfig post\_optimization\_verifier\_config = 301;

}