USING GITHUB CODES & JAVA PROGRAMS CODE INSPECTION, DEBUGGING & STATIC ANALYSIS TOOL

Bhoomish Patel 202201414

[i]CODE INSPECTION:

First 400 loc-

```
#include "network.h"

#include <cstdlib>
#include <fstream>
#include <iostream>
#include <memory>
#include <optional>
#include <type_traits>
#include <vector>

#include "../evaluate.h"
#include "../incbin/incbin.h"
#include "../memory.h"
#include "../misc.h"
#include "../position.h"
#include "../types.h"
#include "nnue_architecture.h"
#include "nnue_common.h"
#include "nnue_misc.h"

namespace {
// Macro to embed the default efficiently updatable neural network (NNUE) file
// data in the engine binary (using incbin.h, by Dale Weiler).
// This macro invocation will declare the following three variables
//     const unsigned char        gEmbeddedNNUEData[];  // a pointer to the embedded data
//     const unsigned char *const gEmbeddedNNUEEnd;     // a marker to the end
//     const unsigned int         gEmbeddedNNUESize;    // the size of the embedded file
// Note that this does not work in Microsoft Visual Studio.
#if !defined(_MSC_VER) && !defined(NNUE_EMBEDDING_OFF)
INCBIN(EmbeddedNNUEBig, EvalFileDefaultNameBig);
INCBIN(EmbeddedNNUESmall, EvalFileDefaultNameSmall);
#else
const unsigned char        gEmbeddedNNUEBigData[1]   = {0x0};
const unsigned char* const gEmbeddedNNUEBigEnd       = &gEmbeddedNNUEBigData[1];
const unsigned int         gEmbeddedNNUEBigSize      = 1;
const unsigned char        gEmbeddedNNUESmallData[1] = {0x0};
const unsigned char* const gEmbeddedNNUESmallEnd     = &gEmbeddedNNUESmallData[1];
const unsigned int         gEmbeddedNNUESmallSize    = 1;
#endif

struct EmbeddedNNUE {
    EmbeddedNNUE(const unsigned char* embeddedData,
                 const unsigned char* embeddedEnd,
                 const unsigned int   embeddedSize) :
        data(embeddedData),
        end(embeddedEnd),
        size(embeddedSize) {}
    const unsigned char* data;
    const unsigned char* end;
    const unsigned int   size;
};

using namespace Stockfish::Eval::NNUE;

EmbeddedNNUE get_embedded(EmbeddedNNUEType type) {
    if (type == EmbeddedNNUEType::BIG)
        return EmbeddedNNUE(gEmbeddedNNUEBigData, gEmbeddedNNUEBigEnd,
gEmbeddedNNUEBigSize);
    else
        return EmbeddedNNUE(gEmbeddedNNUESmallData, gEmbeddedNNUESmallEnd,
gEmbeddedNNUESmallSize);
}

}


namespace Stockfish::Eval::NNUE {


namespace Detail {

// Read evaluation function parameters
template<typename T>
bool read_parameters(std::istream& stream, T& reference) {

    std::uint32_t header;
    header = read_little_endian<std::uint32_t>(stream);
    if (!stream || header != T::get_hash_value())
        return false;
    return reference.read_parameters(stream);
}

// Write evaluation function parameters
template<typename T>
bool write_parameters(std::ostream& stream, const T& reference) {

    write_little_endian<std::uint32_t>(stream, T::get_hash_value());
    return reference.write_parameters(stream);
}

}  // namespace Detail

template<typename Arch, typename Transformer>
Network<Arch, Transformer>::Network(const Network<Arch, Transformer>& other) :
    evalFile(other.evalFile),
    embeddedType(other.embeddedType) {

    if (other.featureTransformer)
        featureTransformer = make_unique_large_page<Transformer>(*other.featureTransformer);

    network = make_unique_aligned<Arch[]>(LayerStacks);

    if (!other.network)
        return;

    for (std::size_t i = 0; i < LayerStacks; ++i)
        network[i] = other.network[i];
}

template<typename Arch, typename Transformer>
Network<Arch, Transformer>&
Network<Arch, Transformer>::operator=(const Network<Arch, Transformer>& other) {
    evalFile     = other.evalFile;
    embeddedType = other.embeddedType;

    if (other.featureTransformer)
        featureTransformer = make_unique_large_page<Transformer>(*other.featureTransformer);

    network = make_unique_aligned<Arch[]>(LayerStacks);

    if (!other.network)
        return *this;
    for (std::size_t i = 0; i < LayerStacks; ++i)
        network[i] = other.network[i];

    return *this;
}

template<typename Arch, typename Transformer>
void Network<Arch, Transformer>::load(const std::string& rootDirectory, std::string evalfilePath)
{
#if defined(DEFAULT_NNUE_DIRECTORY)
    std::vector<std::string> dirs = {"<internal>", "", rootDirectory,
                                     stringify(DEFAULT_NNUE_DIRECTORY)};
#else
    std::vector<std::string> dirs = {"<internal>", "", rootDirectory};
#endif

    if (evalfilePath.empty())
        evalfilePath = evalFile.defaultName;

    for (const auto& directory : dirs)
    {
        if (evalFile.current != evalfilePath)
        {
            if (directory != "<internal>")
            {
                load_user_net(directory, evalfilePath);
            }

            if (directory == "<internal>" && evalfilePath == evalFile.defaultName)
            {
                load_internal();
            }
        }
    }
}


template<typename Arch, typename Transformer>
bool Network<Arch, Transformer>::save(const std::optional<std::string>& filename) const {
    std::string actualFilename;
    std::string msg;

    if (filename.has_value())
        actualFilename = filename.value();
    else
    {
        if (evalFile.current != evalFile.defaultName)
        {
            msg = "Failed to export a net. "
                  "A non-embedded net can only be saved if the filename is specified";

            sync_cout << msg << sync_endl;
            return false;
        }

        actualFilename = evalFile.defaultName;
    }

    std::ofstream stream(actualFilename, std::ios_base::binary);
    bool          saved = save(stream, evalFile.current, evalFile.netDescription);

    msg = saved ? "Network saved successfully to " + actualFilename : "Failed to export a net";

    sync_cout << msg << sync_endl;
    return saved;
}


template<typename Arch, typename Transformer>
NetworkOutput
Network<Arch, Transformer>::evaluate(const Position&                         pos,
                                     AccumulatorCaches::Cache<FTDimensions>* cache) const {
    // We manually align the arrays on the stack because with gcc < 9.3
    // overaligning stack variables with alignas() doesn't work correctly.

    constexpr uint64_t alignment = CacheLineSize;

#if defined(ALIGNAS_ON_STACK_VARIABLES_BROKEN)
    TransformedFeatureType
      transformedFeaturesUnaligned[FeatureTransformer<FTDimensions, nullptr>::BufferSize
                                   + alignment / sizeof(TransformedFeatureType)];

    auto* transformedFeatures = align_ptr_up<alignment>(&transformedFeaturesUnaligned[0]);
#else
    alignas(alignment) TransformedFeatureType
      transformedFeatures[FeatureTransformer<FTDimensions, nullptr>::BufferSize];
#endif
```

```
    ASSERT_ALIGNED(transformedFeatures, alignment);

    const int  bucket     = (pos.count<ALL_PIECES>() - 1) / 4;
    const auto psqt       = featureTransformer->transform(pos, cache, transformedFeatures,
bucket);
    const auto positional = network[bucket].propagate(transformedFeatures);
    return {static_cast<Value>(psqt / OutputScale), static_cast<Value>(positional / OutputScale)};
}


template<typename Arch, typename Transformer>
void Network<Arch, Transformer>::verify(std::string                                evalfilePath,
                                        const std::function<void(std::string_view)>& f) const {
    if (evalfilePath.empty())
        evalfilePath = evalFile.defaultName;

    if (evalFile.current != evalfilePath)
    {
        if (f)
        {
            std::string msg1 =
              "Network evaluation parameters compatible with the engine must be available.";
            std::string msg2 = "The network file " + evalfilePath + " was not loaded successfully.";
            std::string msg3 = "The UCI option EvalFile might need to specify the full path, "
                               "including the directory name, to the network file.";
            std::string msg4 = "The default net can be downloaded from: "
                               "https://tests.stockfishchess.org/api/nn/"
                             + evalFile.defaultName;
            std::string msg5 = "The engine will be terminated now.";

            std::string msg = "ERROR: " + msg1 + '\n' + "ERROR: " + msg2 + '\n' + "ERROR: " +
msg3
                            + '\n' + "ERROR: " + msg4 + '\n' + "ERROR: " + msg5 + '\n';

            f(msg);
        }

        exit(EXIT_FAILURE);
    }

    if (f)
    {
        size_t size = sizeof(*featureTransformer) + sizeof(Arch) * LayerStacks;
        f("info string NNUE evaluation using " + evalfilePath + " ("
          + std::to_string(size / (1024 * 1024)) + "MiB, ("
          + std::to_string(featureTransformer->InputDimensions) + ", "
          + std::to_string(network[0].TransformedFeatureDimensions) + ", "
          + std::to_string(network[0].FC_0_OUTPUTS) + ", " +
std::to_string(network[0].FC_1_OUTPUTS)
          + ", 1))");
    }
}


template<typename Arch, typename Transformer>
void Network<Arch, Transformer>::hint_common_access(
  const Position& pos, AccumulatorCaches::Cache<FTDimensions>* cache) const {
    featureTransformer->hint_common_access(pos, cache);
}

template<typename Arch, typename Transformer>
NnueEvalTrace
Network<Arch, Transformer>::trace_evaluate(const Position&                    pos,
                                           AccumulatorCaches::Cache<FTDimensions>* cache) const {
    // We manually align the arrays on the stack because with gcc < 9.3
    // overaligning stack variables with alignas() doesn't work correctly.
    constexpr uint64_t alignment = CacheLineSize;

#if defined(ALIGNAS_ON_STACK_VARIABLES_BROKEN)
    TransformedFeatureType
      transformedFeaturesUnaligned[FeatureTransformer<FTDimensions, nullptr>::BufferSize
                                   + alignment / sizeof(TransformedFeatureType)];

    auto* transformedFeatures = align_ptr_up<alignment>(&transformedFeaturesUnaligned[0]);
#else
    alignas(alignment) TransformedFeatureType
      transformedFeatures[FeatureTransformer<FTDimensions, nullptr>::BufferSize];
#endif

    ASSERT_ALIGNED(transformedFeatures, alignment);

    NnueEvalTrace t{};
    t.correctBucket = (pos.count<ALL_PIECES>() - 1) / 4;
    for (IndexType bucket = 0; bucket < LayerStacks; ++bucket)
    {
        const auto materialist =
          featureTransformer->transform(pos, cache, transformedFeatures, bucket);
        const auto positional = network[bucket].propagate(transformedFeatures);
        t.psqt[bucket]       = static_cast<Value>(materialist / OutputScale);
        t.positional[bucket] = static_cast<Value>(positional / OutputScale);
    }

    return t;
}


template<typename Arch, typename Transformer>
void Network<Arch, Transformer>::load_user_net(const std::string& dir,
                                               const std::string& evalfilePath) {
    std::ifstream stream(dir + evalfilePath, std::ios::binary);
    auto          description = load(stream);

    if (description.has_value())
    {
        evalFile.current        = evalfilePath;
        evalFile.netDescription = description.value();
    }
}


template<typename Arch, typename Transformer>
void Network<Arch, Transformer>::load_internal() {
    // C++ way to prepare a buffer for a memory stream
    class MemoryBuffer: public std::basic_streambuf<char> {
       public:
        MemoryBuffer(char* p, size_t n) {
            setg(p, p, p + n);
            setp(p, p + n);
        }
    };

    const auto embedded = get_embedded(embeddedType);

    MemoryBuffer buffer(const_cast<char*>(reinterpret_cast<const char*>(embedded.data)),
                        size_t(embedded.size));

    std::istream stream(&buffer);
    auto         description = load(stream);

    if (description.has_value())
    {
        evalFile.current        = evalFile.defaultName;
        evalFile.netDescription = description.value();
    }
}


template<typename Arch, typename Transformer>
void Network<Arch, Transformer>::initialize() {
    featureTransformer = make_unique_large_page<Transformer>();
    network            = make_unique_aligned<Arch[]>(LayerStacks);
}


template<typename Arch, typename Transformer>
bool Network<Arch, Transformer>::save(std::ostream&      stream,
                                      const std::string& name,
                                      const std::string& netDescription) const {
    if (name.empty() || name == "None")
        return false;

    return write_parameters(stream, netDescription);
}


template<typename Arch, typename Transformer>
std::optional<std::string> Network<Arch, Transformer>::load(std::istream& stream) {
    initialize();
    std::string description;

    return read_parameters(stream, description) ? std::make_optional(description) : std::nullopt;
}


// Read network header
template<typename Arch, typename Transformer>
bool Network<Arch, Transformer>::read_header(std::istream&  stream,
                                             std::uint32_t* hashValue,
                                             std::string*   desc) const {
    std::uint32_t version, size;

    version    = read_little_endian<std::uint32_t>(stream);
    *hashValue = read_little_endian<std::uint32_t>(stream);
    size       = read_little_endian<std::uint32_t>(stream);
    if (!stream || version != Version)
```

```
        return false;
    desc->resize(size);
    stream.read(&(*desc)[0], size);
    return !stream.fail();
}


// Write network header
template<typename Arch, typename Transformer>
bool Network<Arch, Transformer>::write_header(std::ostream&      stream,
                                              std::uint32_t      hashValue,
                                              const std::string& desc) const {
    write_little_endian<std::uint32_t>(stream, Version);
    write_little_endian<std::uint32_t>(stream, hashValue);
    write_little_endian<std::uint32_t>(stream, std::uint32_t(desc.size()));
    stream.write(&desc[0], desc.size());
    return !stream.fail();
}


template<typename Arch, typename Transformer>
bool Network<Arch, Transformer>::read_parameters(std::istream& stream,
                                                 std::string&  netDescription) const {
    std::uint32_t hashValue;
    if (!read_header(stream, &hashValue, &netDescription))
        return false;
    if (hashValue != Network::hash)
        return false;
    if (!Detail::read_parameters(stream, *featureTransformer))
        return false;
    for (std::size_t i = 0; i < LayerStacks; ++i)
    {
        if (!Detail::read_parameters(stream, network[i]))
            return false;
    }
    return stream && stream.peek() == std::ios::traits_type::eof();
}


template<typename Arch, typename Transformer>
bool Network<Arch, Transformer>::write_parameters(std::ostream&      stream,
                                                  const std::string& netDescription) const {
    if (!write_header(stream, Network::hash, netDescription))
        return false;
    if (!Detail::write_parameters(stream, *featureTransformer))
        return false;
    for (std::size_t i = 0; i < LayerStacks; ++i)
    {
        if (!Detail::write_parameters(stream, network[i]))
            return false;
    }
    return bool(stream);
}

// Explicit template instantiation

template class Network<
  NetworkArchitecture<TransformedFeatureDimensionsBig, L2Big, L3Big>,
  FeatureTransformer<TransformedFeatureDimensionsBig, &StateInfo::accumulatorBig>>;

template class Network<
  NetworkArchitecture<TransformedFeatureDimensionsSmall, L2Small, L3Small>,
  FeatureTransformer<TransformedFeatureDimensionsSmall, &StateInfo::accumulatorSmall>>;

}  // namespace Stockfish::Eval::NNUE
```

Program Inspection:

By reviewing the provided code, we can identify the following potential issues based on the checklist:

A. Data Reference Errors:

1. **Uninitialized Variables**:
   - `evalFile` is used in multiple places in the code (e.g., in `load()` and `save()` functions). If not initialized correctly, this could lead to errors.
   - Ensure that the memory for `network` and `featureTransformer` is allocated and initialized correctly in the constructors.
2. **Dangling Pointers**:
   - Be cautious with the `EmbeddedNNUE` struct, as it uses pointers to access memory regions like `gEmbeddedNNUEBigData`. These pointers must reference valid data throughout the program's execution.

B. Data Declaration Errors:

1. **Type Confusion**:

- The array `gEmbeddedNNUEBigData[]` is declared as `unsigned char`, which is correct for storing byte data, but ensure that other usages match this type correctly.
- The use of the `std::size_t` type for indexing might avoid potential data type inconsistencies, but ensure this is consistent across all loops.

C. Computation Errors:

1. **Division Operations**:
    - In the function `evaluate()`, ensure the values divided by `OutputScale` don't result in floating-point inaccuracies.
2. **Array Bound Checks**:
    - Ensure that the indexing into arrays (e.g., `network[i]`) stays within bounds for `LayerStacks`.

D. Control-Flow Errors:

1. **Loop Termination**:
    - Check the `for` loops in `load()` and `evaluate()` to confirm they terminate as expected. The `dirs` loop might fail to load a file if the root directory or eval file is incorrect.
2. **Conditionals**:
    - In functions like `save()`, there are checks for `evalFile.current != evalFile.defaultName`. Ensure this condition behaves as expected, especially when handling internal/external NNUE files.

E. Interface Errors:

1. **Parameter Consistency**:
    - The functions `save()` and `load()` take in strings, streams, and other parameters. Ensure that these are passed consistently and are of the correct types.

F. Input/Output Errors:

1. **File Handling**:
    - The `load_user_net()` function reads from the file system using `std::ifstream`. Ensure error handling for failed file reads (e.g., when the file is not found or cannot be opened).

800 loc

// A class that converts the input features of the NNUE evaluation function

#ifndef NNUE_FEATURE_TRANSFORMER_H_INCLUDED

```
#define NNUE_FEATURE_TRANSFORMER_H_INCLUDED

#include <algorithm>
#include <cassert>
#include <cstdint>
#include <cstring>
#include <iosfwd>
#include <utility>

#include "../position.h"
#include "../types.h"
#include "nnue_accumulator.h"
#include "nnue_architecture.h"
#include "nnue_common.h"

namespace Stockfish::Eval::NNUE {

using BiasType       = std::int16_t;
using WeightType     = std::int16_t;
using PSQTWeightType = std::int32_t;

// If vector instructions are enabled, we update and refresh the
// accumulator tile by tile such that each tile fits in the CPU's
// vector registers.
#define VECTOR

static_assert(PSQTBuckets % 8 == 0,
              "Per feature PSQT values cannot be processed at granularity lower than 8 at a time.");

#ifdef USE_AVX512
using vec_t      = __m512i;
using psqt_vec_t = __m256i;
    #define vec_load(a) _mm512_load_si512(a)
    #define vec_store(a, b) _mm512_store_si512(a, b)
    #define vec_add_16(a, b) _mm512_add_epi16(a, b)
    #define vec_sub_16(a, b) _mm512_sub_epi16(a, b)
    #define vec_mulhi_16(a, b) _mm512_mulhi_epi16(a, b)
    #define vec_zero() _mm512_setzero_epi32()
    #define vec_set_16(a) _mm512_set1_epi16(a)
    #define vec_max_16(a, b) _mm512_max_epi16(a, b)
    #define vec_min_16(a, b) _mm512_min_epi16(a, b)
    #define vec_slli_16(a, b) _mm512_slli_epi16(a, b)
    // Inverse permuted at load time
    #define vec_packus_16(a, b) _mm512_packus_epi16(a, b)
    #define vec_load_psqt(a) _mm256_load_si256(a)
    #define vec_store_psqt(a, b) _mm256_store_si256(a, b)
    #define vec_add_psqt_32(a, b) _mm256_add_epi32(a, b)
    #define vec_sub_psqt_32(a, b) _mm256_sub_epi32(a, b)
    #define vec_zero_psqt() _mm256_setzero_si256()
    #define NumRegistersSIMD 16
    #define MaxChunkSize 64

#elif USE_AVX2
using vec_t      = __m256i;
using psqt_vec_t = __m256i;
    #define vec_load(a) _mm256_load_si256(a)
    #define vec_store(a, b) _mm256_store_si256(a, b)
    #define vec_add_16(a, b) _mm256_add_epi16(a, b)
    #define vec_sub_16(a, b) _mm256_sub_epi16(a, b)
    #define vec_mulhi_16(a, b) _mm256_mulhi_epi16(a, b)
    #define vec_zero() _mm256_setzero_si256()
    #define vec_set_16(a) _mm256_set1_epi16(a)
    #define vec_max_16(a, b) _mm256_max_epi16(a, b)
    #define vec_min_16(a, b) _mm256_min_epi16(a, b)
    #define vec_slli_16(a, b) _mm256_slli_epi16(a, b)
    // Inverse permuted at load time
    #define vec_packus_16(a, b) _mm256_packus_epi16(a, b)
    #define vec_load_psqt(a) _mm256_load_si256(a)
    #define vec_store_psqt(a, b) _mm256_store_si256(a, b)
    #define vec_add_psqt_32(a, b) _mm256_add_epi32(a, b)
    #define vec_sub_psqt_32(a, b) _mm256_sub_epi32(a, b)
    #define vec_zero_psqt() _mm256_setzero_si256()
    #define NumRegistersSIMD 16
    #define MaxChunkSize 32

#elif USE_SSE2
using vec_t      = __m128i;
using psqt_vec_t = __m128i;
    #define vec_load(a) (*(a))
    #define vec_store(a, b) *(a) = (b)
    #define vec_add_16(a, b) _mm_add_epi16(a, b)
    #define vec_sub_16(a, b) _mm_sub_epi16(a, b)
    #define vec_mulhi_16(a, b) _mm_mulhi_epi16(a, b)
    #define vec_zero() _mm_setzero_si128()
    #define vec_set_16(a) _mm_set1_epi16(a)
    #define vec_max_16(a, b) _mm_max_epi16(a, b)
    #define vec_min_16(a, b) _mm_min_epi16(a, b)
    #define vec_slli_16(a, b) _mm_slli_epi16(a, b)
    #define vec_packus_16(a, b) _mm_packus_epi16(a, b)
    #define vec_load_psqt(a) (*(a))
    #define vec_store_psqt(a, b) *(a) = (b)
    #define vec_add_psqt_32(a, b) _mm_add_epi32(a, b)
    #define vec_sub_psqt_32(a, b) _mm_sub_epi32(a, b)
    #define vec_zero_psqt() _mm_setzero_si128()
    #define NumRegistersSIMD (Is64Bit ? 16 : 8)
    #define MaxChunkSize 16

#elif USE_NEON
using vec_t      = int16x8_t;
using psqt_vec_t = int32x4_t;
    #define vec_load(a) (*(a))
    #define vec_store(a, b) *(a) = (b)
    #define vec_add_16(a, b) vaddq_s16(a, b)
    #define vec_sub_16(a, b) vsubq_s16(a, b)
    #define vec_mulhi_16(a, b) vqdmulhq_s16(a, b)
    #define vec_zero() \
        vec_t { 0 }
    #define vec_set_16(a) vdupq_n_s16(a)
    #define vec_max_16(a, b) vmaxq_s16(a, b)
    #define vec_min_16(a, b) vminq_s16(a, b)
    #define vec_slli_16(a, b) vshlq_s16(a, vec_set_16(b))
    #define vec_packus_16(a, b) reinterpret_cast<vec_t>(vcombine_u8(vqmovun_s16(a),
vqmovun_s16(b)))
    #define vec_load_psqt(a) (*(a))
    #define vec_store_psqt(a, b) *(a) = (b)
    #define vec_add_psqt_32(a, b) vaddq_s32(a, b)
    #define vec_sub_psqt_32(a, b) vsubq_s32(a, b)
    #define vec_zero_psqt() \
        psqt_vec_t { 0 }
    #define NumRegistersSIMD 16
    #define MaxChunkSize 16

#else
    #undef VECTOR

#endif


#ifdef VECTOR

    // Compute optimal SIMD register count for feature transformer accumulation.
    // We use __m* types as template arguments, which causes GCC to emit warnings
    // about losing some attribute information. This is irrelevant to us as we
    // only take their size, so the following pragma are harmless.
    #if defined(__GNUC__)
        #pragma GCC diagnostic push
        #pragma GCC diagnostic ignored "-Wignored-attributes"
    #endif

template<typename SIMDRegisterType, typename LaneType, int NumLanes, int MaxRegisters>
static constexpr int BestRegisterCount() {
    #define RegisterSize sizeof(SIMDRegisterType)
    #define LaneSize sizeof(LaneType)

    static_assert(RegisterSize >= LaneSize);
    static_assert(MaxRegisters <= NumRegistersSIMD);
    static_assert(MaxRegisters > 0);
    static_assert(NumRegistersSIMD > 0);
    static_assert(RegisterSize % LaneSize == 0);
    static_assert((NumLanes * LaneSize) % RegisterSize == 0);

    const int ideal = (NumLanes * LaneSize) / RegisterSize;
    if (ideal <= MaxRegisters)
        return ideal;

    // Look for the largest divisor of the ideal register count that is smaller than MaxRegisters
    for (int divisor = MaxRegisters; divisor > 1; --divisor)
        if (ideal % divisor == 0)
            return divisor;

    return 1;
}
    #if defined(__GNUC__)
        #pragma GCC diagnostic pop
    #endif
#endif


// Input feature converter
template<IndexType                                TransformedFeatureDimensions,
         Accumulator<TransformedFeatureDimensions> StateInfo::*accPtr>
class FeatureTransformer {

    // Number of output dimensions for one side
    static constexpr IndexType HalfDimensions = TransformedFeatureDimensions;
```

```
  private:
#ifdef VECTOR
    static constexpr int NumRegs =
      BestRegisterCount<vec_t, WeightType, TransformedFeatureDimensions,
NumRegistersSIMD>();
    static constexpr int NumPsqtRegs =
      BestRegisterCount<psqt_vec_t, PSQTWeightType, PSQTBuckets, NumRegistersSIMD>();

    static constexpr IndexType TileHeight     = NumRegs * sizeof(vec_t) / 2;
    static constexpr IndexType PsqtTileHeight = NumPsqtRegs * sizeof(psqt_vec_t) / 4;
    static_assert(HalfDimensions % TileHeight == 0, "TileHeight must divide HalfDimensions");
    static_assert(PSQTBuckets % PsqtTileHeight == 0, "PsqtTileHeight must divide
PSQTBuckets");
#endif

  public:
    // Output type
    using OutputType = TransformedFeatureType;

    // Number of input/output dimensions
    static constexpr IndexType InputDimensions  = FeatureSet::Dimensions;
    static constexpr IndexType OutputDimensions = HalfDimensions;

    // Size of forward propagation buffer
    static constexpr std::size_t BufferSize = OutputDimensions * sizeof(OutputType);

    // Hash value embedded in the evaluation file
    static constexpr std::uint32_t get_hash_value() {
        return FeatureSet::HashValue ^ (OutputDimensions * 2);
    }

    static constexpr void order_packs([[maybe_unused]] uint64_t* v) {
#if defined(USE_AVX512)  // _mm512_packs_epi16 ordering
        uint64_t tmp0 = v[2], tmp1 = v[3];
        v[2] = v[8], v[3] = v[9];
        v[8] = v[4], v[9] = v[5];
        v[4] = tmp0, v[5] = tmp1;
        tmp0 = v[6], tmp1 = v[7];
        v[6] = v[10], v[7] = v[11];
        v[10] = v[12], v[11] = v[13];
        v[12] = tmp0, v[13] = tmp1;
#elif defined(USE_AVX2)  // _mm256_packs_epi16 ordering
        std::swap(v[2], v[4]);
        std::swap(v[3], v[5]);
#endif
    }

    static constexpr void inverse_order_packs([[maybe_unused]] uint64_t* v) {
#if defined(USE_AVX512)  // Inverse _mm512_packs_epi16 ordering
        uint64_t tmp0 = v[2], tmp1 = v[3];
        v[2] = v[4], v[3] = v[5];
        v[4] = v[8], v[5] = v[9];
        v[8] = tmp0, v[9] = tmp1;
        tmp0 = v[6], tmp1 = v[7];
        v[6] = v[12], v[7] = v[13];
        v[12] = v[10], v[13] = v[11];
        v[10] = tmp0, v[11] = tmp1;
#elif defined(USE_AVX2)  // Inverse _mm256_packs_epi16 ordering
        std::swap(v[2], v[4]);
        std::swap(v[3], v[5]);
#endif
    }

    void permute_weights([[maybe_unused]] void (*order_fn)(uint64_t*)) const {
#if defined(USE_AVX2)
    #if defined(USE_AVX512)
        constexpr IndexType di = 16;
    #else
        constexpr IndexType di = 8;
    #endif
        uint64_t* b = reinterpret_cast<uint64_t*>(const_cast<BiasType*>(&biases[0]));
        for (IndexType i = 0; i < HalfDimensions * sizeof(BiasType) / sizeof(uint64_t); i += di)
          order_fn(&b[i]);

        for (IndexType j = 0; j < InputDimensions; ++j)
        {
          uint64_t* w =
           reinterpret_cast<uint64_t*>(const_cast<WeightType*>(&weights[j * HalfDimensions]));
          for (IndexType i = 0; i < HalfDimensions * sizeof(WeightType) / sizeof(uint64_t);
             i += di)
            order_fn(&w[i]);
        }
#endif
    }

    inline void scale_weights(bool read) const {
        for (IndexType j = 0; j < InputDimensions; ++j)
```

```
        {
            WeightType* w = const_cast<WeightType*>(&weights[j * HalfDimensions]);
            for (IndexType i = 0; i < HalfDimensions; ++i)
                w[i] = read ? w[i] * 2 : w[i] / 2;
        }

        BiasType* b = const_cast<BiasType*>(biases);
        for (IndexType i = 0; i < HalfDimensions; ++i)
            b[i] = read ? b[i] * 2 : b[i] / 2;
    }

    // Read network parameters
    bool read_parameters(std::istream& stream) {

        read_leb_128<BiasType>(stream, biases, HalfDimensions);
        read_leb_128<WeightType>(stream, weights, HalfDimensions * InputDimensions);
        read_leb_128<PSQTWeightType>(stream, psqtWeights, PSQTBuckets *
InputDimensions);

        permute_weights(inverse_order_packs);
        scale_weights(true);
        return !stream.fail();
    }

    // Write network parameters
    bool write_parameters(std::ostream& stream) const {

        permute_weights(order_packs);
        scale_weights(false);

        write_leb_128<BiasType>(stream, biases, HalfDimensions);
        write_leb_128<WeightType>(stream, weights, HalfDimensions * InputDimensions);
        write_leb_128<PSQTWeightType>(stream, psqtWeights, PSQTBuckets *
InputDimensions);

        permute_weights(inverse_order_packs);
        scale_weights(true);
        return !stream.fail();
    }

    // Convert input features
    std::int32_t transform(const Position&                       pos,
                           AccumulatorCaches::Cache<HalfDimensions>* cache,
                           OutputType*                               output,
                 int                     bucket) const {
        update_accumulator<WHITE>(pos, cache);
        update_accumulator<BLACK>(pos, cache);

        const Color perspectives[2]  = {pos.side_to_move(), ~pos.side_to_move()};
        const auto& psqtAccumulation = (pos.state()->*accPtr).psqtAccumulation;
        const auto  psqt =
         (psqtAccumulation[perspectives[0]][bucket] - psqtAccumulation[perspectives[1]][bucket])
         / 2;

        const auto& accumulation = (pos.state()->*accPtr).accumulation;

        for (IndexType p = 0; p < 2; ++p)
        {
            const IndexType offset = (HalfDimensions / 2) * p;

#if defined(VECTOR)

            constexpr IndexType OutputChunkSize = MaxChunkSize;
            static_assert((HalfDimensions / 2) % OutputChunkSize == 0);
            constexpr IndexType NumOutputChunks = HalfDimensions / 2 / OutputChunkSize;

            const vec_t Zero = vec_zero();
            const vec_t One  = vec_set_16(127 * 2);

            const vec_t* in0 = reinterpret_cast<const vec_t*>(&(accumulation[perspectives[p]][0]));
            const vec_t* in1 =
             reinterpret_cast<const vec_t*>(&(accumulation[perspectives[p]][HalfDimensions / 2]));
            vec_t* out = reinterpret_cast<vec_t*>(output + offset);

            // Per the NNUE architecture, here we want to multiply pairs of
            // clipped elements and divide the product by 128. To do this,
            // we can naively perform min/max operation to clip each of the
            // four int16 vectors, mullo pairs together, then pack them into
            // one int8 vector. However, there exists a faster way.

            // The idea here is to use the implicit clipping from packus to
            // save us two vec_max_16 instructions. This clipping works due
            // to the fact that any int16 integer below zero will be zeroed
            // on packus.

            // Consider the case where the second element is negative.
            // If we do standard clipping, that element will be zero, which
            // means our pairwise product is zero. If we perform packus and
            // remove the lower-side clip for the second element, then our
            // product before packus will be negative, and is zeroed on pack.
            // The two operation produce equivalent results, but the second
            // one (using packus) saves one max operation per pair.

            // But here we run into a problem: mullo does not preserve the
            // sign of the multiplication. We can get around this by doing
            // mulhi, which keeps the sign. But that requires an additional
            // tweak.

            // mulhi cuts off the last 16 bits of the resulting product,
            // which is the same as performing a rightward shift of 16 bits.
            // We can use this to our advantage. Recall that we want to
            // divide the final product by 128, which is equivalent to a
            // 7-bit right shift. Intuitively, if we shift the clipped
            // value left by 9, and perform mulhi, which shifts the product
            // right by 16 bits, then we will net a right shift of 7 bits.
            // However, this won't work as intended. Since we clip the
            // values to have a maximum value of 127, shifting it by 9 bits
            // might occupy the signed bit, resulting in some positive
            // values being interpreted as negative after the shift.

            // There is a way, however, to get around this limitation. When
            // loading the network, scale accumulator weights and biases by
            // 2. To get the same pairwise multiplication result as before,
            // we need to divide the product by 128 * 2 * 2 = 512, which
            // amounts to a right shift of 9 bits. So now we only have to
            // shift left by 7 bits, perform mulhi (shifts right by 16 bits)
            // and net a 9 bit right shift. Since we scaled everything by
            // two, the values are clipped at 127 * 2 = 254, which occupies
            // 8 bits. Shifting it by 7 bits left will no longer occupy the
            // signed bit, so we are safe.

            // Note that on NEON processors, we shift left by 6 instead
            // because the instruction "vqdmulhq_s16" also doubles the
            // return value after the multiplication, adding an extra shift
            // to the left by 1, so we compensate by shifting less before
            // the multiplication.

            constexpr int shift =
        #if defined(USE_SSE2)
             7;
        #else
             6;
```

```
  #endif

      for (IndexType j = 0; j < NumOutputChunks; ++j)
      {
        const vec_t sum0a =
         vec_slli_16(vec_max_16(vec_min_16(in0[j * 2 + 0], One), Zero), shift);
        const vec_t sum0b =
         vec_slli_16(vec_max_16(vec_min_16(in0[j * 2 + 1], One), Zero), shift);
        const vec_t sum1a = vec_min_16(in1[j * 2 + 0], One);
        const vec_t sum1b = vec_min_16(in1[j * 2 + 1], One);

        const vec_t pa = vec_mulhi_16(sum0a, sum1a);
        const vec_t pb = vec_mulhi_16(sum0b, sum1b);

        out[j] = vec_packus_16(pa, pb);
      }

#else

      for (IndexType j = 0; j < HalfDimensions / 2; ++j)
      {
        BiasType sum0 = accumulation[static_cast<int>(perspectives[p])][j + 0];
        BiasType sum1 =
         accumulation[static_cast<int>(perspectives[p])][j + HalfDimensions / 2];
        sum0               = std::clamp<BiasType>(sum0, 0, 127 * 2);
        sum1               = std::clamp<BiasType>(sum1, 0, 127 * 2);
        output[offset + j] = static_cast<OutputType>(unsigned(sum0 * sum1) / 512);
      }

#endif
    }

    return psqt;
  }  // end of function transform()

  void hint_common_access(const Position&                           pos,
                          AccumulatorCaches::Cache<HalfDimensions>* cache) const {
    hint_common_access_for_perspective<WHITE>(pos, cache);
    hint_common_access_for_perspective<BLACK>(pos, cache);
  }

 private:
  template<Color Perspective>
  StateInfo* try_find_computed_accumulator(const Position& pos) const {
    // Look for a usable accumulator of an earlier position. We keep track
    // of the estimated gain in terms of features to be added/subtracted.
    StateInfo* st   = pos.state();
    int        gain = FeatureSet::refresh_cost(pos);
    while (st->previous && !(st->*accPtr).computed[Perspective])
    {
      // This governs when a full feature refresh is needed and how many
      // updates are better than just one full refresh.
      if (FeatureSet::requires_refresh(st, Perspective)
        || (gain -= FeatureSet::update_cost(st) + 1) < 0)
        break;
      st = st->previous;
    }
    return st;
  }

  // It computes the accumulator of the next position, or updates the
  // current position's accumulator if CurrentOnly is true.
  template<Color Perspective, bool CurrentOnly>
  void update_accumulator_incremental(const Position& pos, StateInfo* computed) const {
    assert((computed->*accPtr).computed[Perspective]);
    assert(computed->next != nullptr);

#ifdef VECTOR
    // Gcc-10.2 unnecessarily spills AVX2 registers if this array
    // is defined in the VECTOR code below, once in each branch.
    vec_t      acc[NumRegs];
    psqt_vec_t psqt[NumPsqtRegs];
#endif

    const Square ksq = pos.square<KING>(Perspective);

    // The size must be enough to contain the largest possible update.
    // That might depend on the feature set and generally relies on the
    // feature set's update cost calculation to be correct and never allow
    // updates with more added/removed features than MaxActiveDimensions.
    FeatureSet::IndexList removed, added;

    if constexpr (CurrentOnly)
      for (StateInfo* st = pos.state(); st != computed; st = st->previous)
        FeatureSet::append_changed_indices<Perspective>(ksq, st->dirtyPiece, removed,
                                                        added);
    else
      FeatureSet::append_changed_indices<Perspective>(ksq, computed->next->dirtyPiece,
                              removed, added);

    StateInfo* next = CurrentOnly ? pos.state() : computed->next;
    assert(!(next->*accPtr).computed[Perspective]);

#ifdef VECTOR
    if ((removed.size() == 1 || removed.size() == 2) && added.size() == 1)
    {
      auto accIn =
       reinterpret_cast<const vec_t*>(&(computed->*accPtr).accumulation[Perspective][0]);
      auto accOut = reinterpret_cast<vec_t*>(&(next->*accPtr).accumulation[Perspective][0]);

      const IndexType offsetR0 = HalfDimensions * removed[0];
      auto            columnR0 = reinterpret_cast<const vec_t*>(&weights[offsetR0]);
      const IndexType offsetA  = HalfDimensions * added[0];
      auto            columnA  = reinterpret_cast<const vec_t*>(&weights[offsetA]);

      if (removed.size() == 1)
      {
        for (IndexType i = 0; i < HalfDimensions * sizeof(WeightType) / sizeof(vec_t); ++i)
          accOut[i] = vec_add_16(vec_sub_16(accIn[i], columnR0[i]), columnA[i]);
      }
      else
      {
        const IndexType offsetR1 = HalfDimensions * removed[1];
        auto            columnR1 = reinterpret_cast<const vec_t*>(&weights[offsetR1]);

        for (IndexType i = 0; i < HalfDimensions * sizeof(WeightType) / sizeof(vec_t); ++i)
          accOut[i] = vec_sub_16(vec_add_16(accIn[i], columnA[i]),
                                 vec_add_16(columnR0[i], columnR1[i]));
      }

      auto accPsqtIn = reinterpret_cast<const psqt_vec_t*>(
       &(computed->*accPtr).psqtAccumulation[Perspective][0]);
      auto accPsqtOut =
       reinterpret_cast<psqt_vec_t*>(&(next->*accPtr).psqtAccumulation[Perspective][0]);

      const IndexType offsetPsqtR0 = PSQTBuckets * removed[0];
      auto columnPsqtR0 = reinterpret_cast<const psqt_vec_t*>(&psqtWeights[offsetPsqtR0]);
      const IndexType offsetPsqtA = PSQTBuckets * added[0];
      auto columnPsqtA = reinterpret_cast<const psqt_vec_t*>(&psqtWeights[offsetPsqtA]);

      if (removed.size() == 1)
      {
        for (std::size_t i = 0;
           i < PSQTBuckets * sizeof(PSQTWeightType) / sizeof(psqt_vec_t); ++i)
          accPsqtOut[i] = vec_add_psqt_32(vec_sub_psqt_32(accPsqtIn[i],
columnPsqtR0[i]),
                                          columnPsqtA[i]);
      }
      else
      {
        const IndexType offsetPsqtR1 = PSQTBuckets * removed[1];
        auto columnPsqtR1 = reinterpret_cast<const
psqt_vec_t*>(&psqtWeights[offsetPsqtR1]);

        for (std::size_t i = 0;
           i < PSQTBuckets * sizeof(PSQTWeightType) / sizeof(psqt_vec_t); ++i)
          accPsqtOut[i] =
           vec_sub_psqt_32(vec_add_psqt_32(accPsqtIn[i], columnPsqtA[i]),
                   vec_add_psqt_32(columnPsqtR0[i], columnPsqtR1[i]));
      }
    }
    else
    {
      for (IndexType i = 0; i < HalfDimensions / TileHeight; ++i)
      {
        // Load accumulator
        auto accTileIn = reinterpret_cast<const vec_t*>(
         &(computed->*accPtr).accumulation[Perspective][i * TileHeight]);
        for (IndexType j = 0; j < NumRegs; ++j)
          acc[j] = vec_load(&accTileIn[j]);

        // Difference calculation for the deactivated features
        for (const auto index : removed)
        {
          const IndexType offset = HalfDimensions * index + i * TileHeight;
          auto       column = reinterpret_cast<const vec_t*>(&weights[offset]);
          for (IndexType j = 0; j < NumRegs; ++j)
            acc[j] = vec_sub_16(acc[j], column[j]);
        }

        // Difference calculation for the activated features
        for (const auto index : added)
        {
          const IndexType offset = HalfDimensions * index + i * TileHeight;
          auto       column = reinterpret_cast<const vec_t*>(&weights[offset]);
          for (IndexType j = 0; j < NumRegs; ++j)
```

```
            acc[j] = vec_add_16(acc[j], column[j]);
        }

        // Store accumulator
        auto accTileOut = reinterpret_cast<vec_t*>(
         &(next->*accPtr).accumulation[Perspective][i * TileHeight]);
        for (IndexType j = 0; j < NumRegs; ++j)
          vec_store(&accTileOut[j], acc[j]);
      }

      for (IndexType i = 0; i < PSQTBuckets / PsqtTileHeight; ++i)
      {
        // Load accumulator
        auto accTilePsqtIn = reinterpret_cast<const psqt_vec_t*>(
         &(computed->*accPtr).psqtAccumulation[Perspective][i * PsqtTileHeight]);
        for (std::size_t j = 0; j < NumPsqtRegs; ++j)
          psqt[j] = vec_load_psqt(&accTilePsqtIn[j]);

        // Difference calculation for the deactivated features
        for (const auto index : removed)
        {
          const IndexType offset = PSQTBuckets * index + i * PsqtTileHeight;
          auto columnPsqt = reinterpret_cast<const psqt_vec_t*>(&psqtWeights[offset]);
          for (std::size_t j = 0; j < NumPsqtRegs; ++j)
            psqt[j] = vec_sub_psqt_32(psqt[j], columnPsqt[j]);
        }

        // Difference calculation for the activated features
        for (const auto index : added)
        {
          const IndexType offset = PSQTBuckets * index + i * PsqtTileHeight;
          auto columnPsqt = reinterpret_cast<const psqt_vec_t*>(&psqtWeights[offset]);
          for (std::size_t j = 0; j < NumPsqtRegs; ++j)
            psqt[j] = vec_add_psqt_32(psqt[j], columnPsqt[j]);
        }

        // Store accumulator
        auto accTilePsqtOut = reinterpret_cast<psqt_vec_t*>(
         &(next->*accPtr).psqtAccumulation[Perspective][i * PsqtTileHeight]);
        for (std::size_t j = 0; j < NumPsqtRegs; ++j)
          vec_store_psqt(&accTilePsqtOut[j], psqt[j]);
      }
    }
#else
```

```
        std::memcpy((next->*accPtr).accumulation[Perspective],
                    (computed->*accPtr).accumulation[Perspective],
                    HalfDimensions * sizeof(BiasType));
        std::memcpy((next->*accPtr).psqtAccumulation[Perspective],
                    (computed->*accPtr).psqtAccumulation[Perspective],
                    PSQTBuckets * sizeof(PSQTWeightType));

        // Difference calculation for the deactivated features
        for (const auto index : removed)
        {
            const IndexType offset = HalfDimensions * index;
            for (IndexType i = 0; i < HalfDimensions; ++i)
                (next->*accPtr).accumulation[Perspective][i] -= weights[offset + i];

            for (std::size_t i = 0; i < PSQTBuckets; ++i)
                (next->*accPtr).psqtAccumulation[Perspective][i] -=
                  psqtWeights[index * PSQTBuckets + i];
        }

        // Difference calculation for the activated features
        for (const auto index : added)
        {
            const IndexType offset = HalfDimensions * index;
            for (IndexType i = 0; i < HalfDimensions; ++i)
                (next->*accPtr).accumulation[Perspective][i] += weights[offset + i];

            for (std::size_t i = 0; i < PSQTBuckets; ++i)
                (next->*accPtr).psqtAccumulation[Perspective][i] +=
                  psqtWeights[index * PSQTBuckets + i];
        }
#endif

        (next->*accPtr).computed[Perspective] = true;

        if (!CurrentOnly && next != pos.state())
            update_accumulator_incremental<Perspective, false>(pos, next);
    }

    template<Color Perspective>
    void update_accumulator_refresh_cache(const Position&                          pos,
                                          AccumulatorCaches::Cache<HalfDimensions>* cache) const {
        assert(cache != nullptr);

        Square                ksq   = pos.square<KING>(Perspective);
    auto&                 entry = (*cache)[ksq][Perspective];
    FeatureSet::IndexList removed, added;

    for (Color c : {WHITE, BLACK})
    {
        for (PieceType pt = PAWN; pt <= KING; ++pt)
        {
            const Piece    piece    = make_piece(c, pt);
            const Bitboard oldBB    = entry.byColorBB[c] & entry.byTypeBB[pt];
            const Bitboard newBB    = pos.pieces(c, pt);
            Bitboard       toRemove = oldBB & ~newBB;
            Bitboard       toAdd    = newBB & ~oldBB;

            while (toRemove)
            {
                Square sq = pop_lsb(toRemove);
                removed.push_back(FeatureSet::make_index<Perspective>(sq, piece, ksq));
            }
            while (toAdd)
            {
                Square sq = pop_lsb(toAdd);
                added.push_back(FeatureSet::make_index<Perspective>(sq, piece, ksq));
            }
        }
    }

    auto& accumulator                 = pos.state()->*accPtr;
    accumulator.computed[Perspective] = true;

#ifdef VECTOR
    vec_t      acc[NumRegs];
    psqt_vec_t psqt[NumPsqtRegs];

    for (IndexType j = 0; j < HalfDimensions / TileHeight; ++j)
    {
        auto accTile =
          reinterpret_cast<vec_t*>(&accumulator.accumulation[Perspective][j * TileHeight]);
        auto entryTile = reinterpret_cast<vec_t*>(&entry.accumulation[j * TileHeight]);

        for (IndexType k = 0; k < NumRegs; ++k)
            acc[k] = entryTile[k];

        int i = 0;
        for (; i < int(std::min(removed.size(), added.size())); ++i)
```

```
  {
    IndexType       indexR  = removed[i];
    const IndexType offsetR = HalfDimensions * indexR + j * TileHeight;
    auto            columnR = reinterpret_cast<const vec_t*>(&weights[offsetR]);
    IndexType       indexA  = added[i];
    const IndexType offsetA = HalfDimensions * indexA + j * TileHeight;
    auto            columnA = reinterpret_cast<const vec_t*>(&weights[offsetA]);

    for (unsigned k = 0; k < NumRegs; ++k)
      acc[k] = vec_add_16(acc[k], vec_sub_16(columnA[k], columnR[k]));
  }
  for (; i < int(removed.size()); ++i)
  {
    IndexType       index  = removed[i];
    const IndexType offset = HalfDimensions * index + j * TileHeight;
    auto            column = reinterpret_cast<const vec_t*>(&weights[offset]);

    for (unsigned k = 0; k < NumRegs; ++k)
      acc[k] = vec_sub_16(acc[k], column[k]);
  }
  for (; i < int(added.size()); ++i)
  {
    IndexType       index  = added[i];
    const IndexType offset = HalfDimensions * index + j * TileHeight;
    auto            column = reinterpret_cast<const vec_t*>(&weights[offset]);

    for (unsigned k = 0; k < NumRegs; ++k)
      acc[k] = vec_add_16(acc[k], column[k]);
  }

  for (IndexType k = 0; k < NumRegs; k++)
    vec_store(&entryTile[k], acc[k]);
  for (IndexType k = 0; k < NumRegs; k++)
    vec_store(&accTile[k], acc[k]);
}

for (IndexType j = 0; j < PSQTBuckets / PsqtTileHeight; ++j)
{
  auto accTilePsqt = reinterpret_cast<psqt_vec_t*>(
   &accumulator.psqtAccumulation[Perspective][j * PsqtTileHeight]);
  auto entryTilePsqt =
   reinterpret_cast<psqt_vec_t*>(&entry.psqtAccumulation[j * PsqtTileHeight]);

  for (std::size_t k = 0; k < NumPsqtRegs; ++k)
```

```
                psqt[k] = entryTilePsqt[k];

            for (int i = 0; i < int(removed.size()); ++i)
            {
                IndexType       index  = removed[i];
                const IndexType offset = PSQTBuckets * index + j * PsqtTileHeight;
                auto columnPsqt        = reinterpret_cast<const psqt_vec_t*>(&psqtWeights[offset]);

                for (std::size_t k = 0; k < NumPsqtRegs; ++k)
                    psqt[k] = vec_sub_psqt_32(psqt[k], columnPsqt[k]);
            }
            for (int i = 0; i < int(added.size()); ++i)
            {
                IndexType       index  = added[i];
                const IndexType offset = PSQTBuckets * index + j * PsqtTileHeight;
                auto columnPsqt        = reinterpret_cast<const psqt_vec_t*>(&psqtWeights[offset]);

                for (std::size_t k = 0; k < NumPsqtRegs; ++k)
                    psqt[k] = vec_add_psqt_32(psqt[k], columnPsqt[k]);
            }

            for (std::size_t k = 0; k < NumPsqtRegs; ++k)
                vec_store_psqt(&entryTilePsqt[k], psqt[k]);
            for (std::size_t k = 0; k < NumPsqtRegs; ++k)
                vec_store_psqt(&accTilePsqt[k], psqt[k]);
        }

#else

        for (const auto index : removed)
        {
            const IndexType offset = HalfDimensions * index;
            for (IndexType j = 0; j < HalfDimensions; ++j)
                entry.accumulation[j] -= weights[offset + j];

            for (std::size_t k = 0; k < PSQTBuckets; ++k)
                entry.psqtAccumulation[k] -= psqtWeights[index * PSQTBuckets + k];
        }
        for (const auto index : added)
        {
            const IndexType offset = HalfDimensions * index;
            for (IndexType j = 0; j < HalfDimensions; ++j)
                entry.accumulation[j] += weights[offset + j];
            for (std::size_t k = 0; k < PSQTBuckets; ++k)
                entry.psqtAccumulation[k] += psqtWeights[index * PSQTBuckets + k];
        }

        // The accumulator of the refresh entry has been updated.
        // Now copy its content to the actual accumulator we were refreshing.

        std::memcpy(accumulator.accumulation[Perspective], entry.accumulation,
                    sizeof(BiasType) * HalfDimensions);

        std::memcpy(accumulator.psqtAccumulation[Perspective], entry.psqtAccumulation,
                    sizeof(int32_t) * PSQTBuckets);
#endif

        for (Color c : {WHITE, BLACK})
            entry.byColorBB[c] = pos.pieces(c);

        for (PieceType pt = PAWN; pt <= KING; ++pt)
            entry.byTypeBB[pt] = pos.pieces(pt);
    }

    template<Color Perspective>
    void hint_common_access_for_perspective(const Position&                           pos,
                                            AccumulatorCaches::Cache<HalfDimensions>* cache) const {

        // Works like update_accumulator, but performs less work.
        // Updates ONLY the accumulator for pos.

        // Look for a usable accumulator of an earlier position. We keep track
        // of the estimated gain in terms of features to be added/subtracted.
        // Fast early exit.
        if ((pos.state()->*accPtr).computed[Perspective])
            return;

        StateInfo* oldest = try_find_computed_accumulator<Perspective>(pos);

        if ((oldest->*accPtr).computed[Perspective] && oldest != pos.state())
            update_accumulator_incremental<Perspective, true>(pos, oldest);
        else
            update_accumulator_refresh_cache<Perspective>(pos, cache);
    }

    template<Color Perspective>
    void update_accumulator(const Position&                           pos,
              AccumulatorCaches::Cache<HalfDimensions>* cache) const {

    StateInfo* oldest = try_find_computed_accumulator<Perspective>(pos);

    if ((oldest->*accPtr).computed[Perspective] && oldest != pos.state())
      // Start from the oldest computed accumulator, update all the
      // accumulators up to the current position.
      update_accumulator_incremental<Perspective, false>(pos, oldest);
    else
      update_accumulator_refresh_cache<Perspective>(pos, cache);
  }

  template<IndexType Size>
  friend struct AccumulatorCaches::Cache;

  alignas(CacheLineSize) BiasType biases[HalfDimensions];
  alignas(CacheLineSize) WeightType weights[HalfDimensions * InputDimensions];
  alignas(CacheLineSize) PSQTWeightType psqtWeights[InputDimensions * PSQTBuckets];
};

}  // namespace Stockfish::Eval::NNUE

#endif  // #ifndef NNUE_FEATURE_TRANSFORMER_H_INCLUDED
```

A. Data Reference Errors:

1. **Uninitialized Variables**:
   - The static asserts, such as `static_assert(HalfDimensions % TileHeight == 0)` and `static_assert(PSQTBuckets % 8 == 0)`, rely on these constants being set correctly. If not, the program could encounter unexpected behavior.
2. **Vector Variables**:
   - The vector types `vec_t` and `psqt_vec_t` could potentially lead to memory alignment issues if not handled properly in different architectures (especially between AVX512, AVX2, SSE2, and NEON).

B. Data Declaration Errors:

1. **Type Confusion**:
   - Ensure that vector types like `vec_t` and `psqt_vec_t` are correctly defined for the specific architecture being used (e.g., AVX2 vs SSE2). Improper architecture support could lead to runtime errors if the compiler doesn't support certain instructions.
2. **Conditionally Defined Macros**:

    - Macros like `VECTOR` depend on the specific architecture. If undefined or misconfigured, functions that depend on vector operations (e.g., `transform()`) could fail.

C. Computation Errors:

1. **Integer Overflow**:
    - Multiplications, such as `w[i] = read ? w[i] * 2 : w[i] / 2`, can cause overflow if `w[i]` contains values large enough to exceed the bounds of the data type.
2. **Division by Constants**:
    - Ensure that constants such as `512` in the division (`unsigned(sum0 * sum1) / 512`) are well-defined and chosen appropriately to avoid loss of precision.

D. Control-Flow Errors:

1. **Loop Termination**:
    - The loop inside `transform()` is bounded by constants like `NumOutputChunks`. If these constants are incorrectly computed, the loop could either overrun or underutilize resources.

E. Interface Errors:

1. **Parameter Matching**:
    - Ensure the proper number and types of arguments are passed to functions like `permute_weights()` and `scale_weights()`. Any mismatch could cause crashes or miscalculations.

F. Input/Output Errors:

1. **Stream Handling**:
    - The functions `read_parameters()` and `write_parameters()` handle input and output streams. Ensure proper error handling, such as checking if the stream has failed (e.g., `!stream.fail()`), to prevent issues with file reading or writing.

[ii]DEBUGGING:

1. Armstrong Number Program

- Error: Incorrect computation of the remainder.
- Fix: Use breakpoints to check the remainder calculation.

Corrected Code:

```
class Armstrong {
public static void main(String args[]) {
```

```
int num = Integer.parseInt(args[0]);
int n = num, check = 0, remainder;
while (num > 0) {
remainder = num % 10;
check += Math.pow(remainder, 3);
num /= 10;
}
if (check == n) {
System.out.println(n + " is an Armstrong Number");
} else {
System.out.println(n + " is not an Armstrong Number");
}
}
}
```

2. GCD and LCM Program

- Errors:

1. Incorrect while loop condition in GCD.
2. Incorrect LCM calculation logic.

- Fix: Breakpoints at the GCD loop and LCM logic.

Corrected Code:

```
import java.util.Scanner;
public class GCD_LCM {
static int gcd(int x, int y) {
while (y != 0) {
int temp = y;
y = x % y;
x = temp;
}
return x;
}
static int lcm(int x, int y) {
return (x * y) / gcd(x, y);
}
public static void main(String args[]) {
Scanner input = new Scanner(System.in);
System.out.println("Enter the two numbers: ");
int x = input.nextInt();
int y = input.nextInt();
System.out.println("The GCD of two numbers is: " + gcd(x, y));
System.out.println("The LCM of two numbers is: " + lcm(x, y));
input.close();
}
}
```

3. Knapsack Program

- Error: Incrementing n inappropriately in the loop.
- Fix: Breakpoint to check loop behavior.

Corrected Code:

```
public class Knapsack {
public static void main(String[] args) {
int N = Integer.parseInt(args[0]);
int W = Integer.parseInt(args[1]);
int[] profit = new int[N + 1], weight = new int[N + 1];
int[][] opt = new int[N + 1][W + 1];
boolean[][] sol = new boolean[N + 1][W + 1];
for (int n = 1; n <= N; n++) {
for (int w = 1; w <= W; w++) {
int option1 = opt[n - 1][w];
int option2 = (weight[n] <= w) ? profit[n] + opt[n - 1][w - weight[n]] :
Integer.MIN_VALUE;
opt[n][w] = Math.max(option1, option2);
sol[n][w] = (option2 > option1);
}
}
}
}
```

4. Magic Number Program

- Errors:

1. Incorrect condition in the inner while loop.
2. Missing semicolons in expressions.

- Fix: Set breakpoints at the inner while loop and check variable values.

Corrected Code:

```
import java.util.Scanner;
public class MagicNumberCheck {
public static void main(String args[]) {
Scanner ob = new Scanner(System.in);
System.out.println("Enter the number to be checked.");
int n = ob.nextInt();
int sum = 0, num = n;
while (num > 9) {
sum = num;
int s = 0;
while (sum > 0) {
s = s * (sum / 10); // Fixed missing semicolon
sum = sum % 10;
}
num = s;
}
if (num == 1) {
System.out.println(n + " is a Magic Number.");
} else {
System.out.println(n + " is not a Magic Number.");
}
}
}
```

5. Merge Sort Program

- Errors:

1. Incorrect array splitting logic.
2. Incorrect inputs for the merge method.

- Fix: Breakpoints at array split and merge operations.

Corrected Code:

```
import java.util.Scanner;
public class MergeSort {
public static void main(String[] args) {
int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
System.out.println("Before: " + Arrays.toString(list));
mergeSort(list);
System.out.println("After: " + Arrays.toString(list));
}
public static void mergeSort(int[] array) {
if (array.length > 1) {
int[] left = leftHalf(array);
int[] right = rightHalf(array);
mergeSort(left);
mergeSort(right);
merge(array, left, right);
}
}
public static int[] leftHalf(int[] array) {
int size1 = array.length / 2;
int[] left = new int[size1];
System.arraycopy(array, 0, left, 0, size1);
return left;
}
public static int[] rightHalf(int[] array) {
int size1 = array.length / 2;
int size2 = array.length - size1;
int[] right = new int[size2];
System.arraycopy(array, size1, right, 0, size2);
return right;
}
public static void merge(int[] result, int[] left, int[] right) {
```

```java
int i1 = 0, i2 = 0;
for (int i = 0; i < result.length; i++) {
if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2])) {
result[i] = left[i1];
i1++;
} else {
result[i] = right[i2];
i2++;
}
}
}
}
```

6. Multiply Matrices Program

- Errors:

1. Incorrect loop indices.
2. Wrong error message.

- Fix: Set breakpoints to check matrix multiplication and correct messages.

Corrected Code:

```java
import java.util.Scanner;
class MatrixMultiplication {
public static void main(String args[]) {
int m, n, p, q, sum = 0, c, d, k;
Scanner in = new Scanner(System.in);
System.out.println("Enter the number of rows and columns of the first matrix");
m = in.nextInt();
n = in.nextInt();
int first[][] = new int[m][n];
System.out.println("Enter the elements of the first matrix");
for (c = 0; c < m; c++)
for (d = 0; d < n; d++)
first[c][d] = in.nextInt();
System.out.println("Enter the number of rows and columns of the second matrix");
p = in.nextInt();
q = in.nextInt();
if (n != p)
System.out.println("Matrices with entered orders can't be multiplied.");
else {
int second[][] = new int[p][q];
int multiply[][] = new int[m][q];
System.out.println("Enter the elements of the second matrix");
for (c = 0; c < p; c++)
for (d = 0; d < q; d++)
second[c][d] = in.nextInt();
for (c = 0; c < m; c++) {
for (d = 0; d < q; d++) {
for (k = 0; k < p; k++) {
sum += first[c][k] * second[k][d];
}
multiply[c][d] = sum;
sum = 0;
}
}
System.out.println("Product of entered matrices:");
for (c = 0; c < m; c++) {
for (d = 0; d < q; d++)
System.out.print(multiply[c][d] + "\t");
System.out.print("\n");
}
}
}
}
```

7. Quadratic Probing Hash Table Program

● Errors:

1. Typos in insert, remove, and get methods.
2. Incorrect logic for rehashing.

● Fix: Set breakpoints and step through logic for insert, remove, and get methods.

Corrected Code:

```
import java.util.Scanner;
class QuadraticProbingHashTable {
private int currentSize, maxSize;
private String[] keys, vals;
public QuadraticProbingHashTable(int capacity) {
currentSize = 0;
maxSize = capacity;
keys = new String[maxSize];
vals = new String[maxSize];
}
public void insert(String key, String val) {
int tmp = hash(key), i = tmp, h = 1;
do {
if (keys[i] == null) {
keys[i] = key;
vals[i] = val;
currentSize++;
return;
}
if (keys[i].equals(key)) {
vals[i] = val;
return;
}
i += (h * h++) % maxSize;
} while (i != tmp);
}
public String get(String key) {
int i = hash(key), h = 1;
while (keys[i] != null) {
if (keys[i].equals(key))
return vals[i];
i = (i + h * h++) % maxSize;
}
return null;
}
public void remove(String key) {
if (!contains(key)) return;
int i = hash(key), h = 1;
while (!key.equals(keys[i]))
i = (i + h * h++) % maxSize;
keys[i] = vals[i] = null;
}
private boolean contains(String key) {
return get(key) != null;
}
private int hash(String key) {
return key.hashCode() % maxSize;
}
}
public class HashTableTest {
public static void main(String[] args) {
Scanner scan = new Scanner(System.in);
QuadraticProbingHashTable hashTable = new
QuadraticProbingHashTable(scan.nextInt());
hashTable.insert("key1", "value1");
System.out.println("Value: " + hashTable.get("key1"));
}
}
```

8. Sorting Array Program

- Errors:

1. Incorrect class name with an extra space.
2. Incorrect loop condition and extra semicolon.

- Fix: Set breakpoints to check the loop and class name.

Corrected Code:

```
import java.util.Scanner;
public class AscendingOrder {
public static void main(String[] args) {
int n, temp;
Scanner s = new Scanner(System.in);
System.out.print("Enter the number of elements: ");
n = s.nextInt();
int[] a = new int[n];
System.out.println("Enter all the elements:");
for (int i = 0; i < n; i++) a[i] = s.nextInt();
for (int i = 0; i < n; i++) {
for (int j = i + 1; j < n; j++) {
if (a[i] > a[j]) {
temp = a[i];
a[i] = a[j];
a[j] = temp;
}
}
}
System.out.println("Sorted Array: " + Arrays.toString(a));
}
}
```

9. Stack Implementation Program

- Errors:

1. Incorrect top-- instead of top++ in push.
2. Incorrect loop condition in display.
3. Missing pop method.

- Fix: Add breakpoints to check push, pop, and display methods.

Corrected Code:

```
public class StackMethods {
private int top;
private int[] stack;
public StackMethods(int size) {
stack = new int[size];
top = -1;
}
public void push(int value) {
if (top == stack.length - 1) {
System.out.println("Stack full");
} else {
```

```
stack[++top] = value;
}
}
public void pop() {
if (top == -1) {
System.out.println("Stack empty");
} else {
top--;
}
}
public void display() {
for (int i = 0; i <= top; i++) {
System.out.print(stack[i] + " ");
}
System.out.println();
}
}
```

10. Tower of Hanoi Program

- Error: Incorrect increment/decrement in recursive call.
- Fix: Breakpoints at the recursive calls to verify logic.

Corrected Code:

```
public class TowerOfHanoi {
public static void main(String[] args) {
int nDisks = 3;
doTowers(nDisks, 'A', 'B', 'C');
}
public static void doTowers(int topN, char from, char inter, char to) {
if (topN == 1) {
System.out.println("Disk 1 from " + from + " to " + to);
} else {
doTowers(topN - 1, from, to, inter);
System.out.println("Disk " + topN + " from " + from + " to " + to);
doTowers(topN - 1, inter, from, to);
}
}
}
```

```
[202201414_Lab3_2.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201414_Lab3_2.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201414_Lab3_2.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201414_Lab3_2.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
```

```
[202201414_Lab3_2.c:5]: (information) Include file: <unistd.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201414_Lab3_2.c:6]: (information) Include file: <dirent.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201414_Lab3_2.c:7]: (information) Include file: <fcntl.h> not found. Please note: Cppcheck
does not need standard library headers to get proper results.
[202201414_Lab3_2.c:8]: (information) Include file: <libgen.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201414_Lab3_2.c:9]: (information) Include file: <errno.h> not found. Please note: Cppcheck
does not need standard library headers to get proper results.
[202201414_Lab3_2.c:10]: (information) Include file: <string.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201414_Lab3_2.c:0]: (information) Limiting analysis of branches. Use
--check-level=exhaustive to analyze all branches.
[202201414_Lab3_2.c:116]: (warning) scanf() without field width limits can crash with huge input
data.
[202201414_Lab3_2.c:120]: (warning) scanf() without field width limits can crash with huge
input data.
[202201414_Lab3_2.c:126]: (warning) scanf() without field width limits can crash with huge
input data.
[202201414_Lab3_2.c:127]: (warning) scanf() without field width limits can crash with huge
input data.
[202201414_Lab3_2.c:133]: (warning) scanf() without field width limits can crash with huge
input data.
[202201414_Lab3_2.c:34]: (style) The scope of the variable 'ch' can be reduced.
[202201414_Lab3_2.c:115]: (style) The scope of the variable 'path2' can be reduced.
[202201414_Lab3_2.c:16]: (style) Parameter 'file' can be declared as pointer to const.
[202201414_Lab3_2.c:55]: (style) Variable 'direntp' can be declared as pointer to const.
[202201414_Lab3_2.c:40]: (warning) Storing fgetc() return value in char variable and then
comparing with EOF.
[202201414_Lab3_3.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck
does not need standard library headers to get proper results.
[202201414_Lab3_3.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck
does not need standard library headers to get proper results.
[202201414_Lab3_3.c:3]: (information) Include file: <sys/types.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201414_Lab3_3.c:4]: (information) Include file: <sys/stat.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201414_Lab3_3.c:5]: (information) Include file: <unistd.h> not found. Please note:
Cppcheck does not need standard library headers to get proper results.
[202201414_lab3_1.c:1]: (information) Include file: <stdio.h> not found. Please note: Cppcheck
does not need standard library headers to get proper results.
[202201414_lab3_1.c:2]: (information) Include file: <stdlib.h> not found. Please note: Cppcheck
does not need standard library headers to get proper results.
```

```
[202201414_lab3_1.c:3]: (information) Include file: <sys/types.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201414_lab3_1.c:4]: (information) Include file: <sys/stat.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201414_lab3_1.c:5]: (information) Include file: <unistd.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201414_lab3_1.c:6]: (information) Include file: <dirent.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201414_lab3_1.c:7]: (information) Include file: <fcntl.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201414_lab3_1.c:8]: (information) Include file: <libgen.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201414_lab3_1.c:9]: (information) Include file: <errno.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.
[202201414_lab3_1.c:29]: (style) The scope of the variable 'ch' can be reduced.
[202201414_lab3_1.c:11]: (style) Parameter 'file' can be declared as pointer to const.
[202201414_lab3_1.c:50]: (style) Variable 'direntp' can be declared as pointer to const.
[202201414_lab3_1.c:35]: (warning) Storing fgetc() return value in char variable and then comparing with EOF.
```