New Memtrace Features: Instruction Encodings & Fast Seeking

How To Obtain Instruction Encodings

- Old way: mmap binary and decode from there
  - Requires the binaries and address mappings (modules.log file)
- New way: encoding bytes embedded in each instruction record
  - No need for binaries anymore

Modified Code Support

- New field indicates whether the encoding has changed so a tool can invalidate cached decoding information
  - Although modified code does not happen for existing shared workloads

New Instruction Record Fields

```
/**
 * The instruction's raw encoding.  This field is only valid when the the file type
 * (see #TRACE_MARKER_TYPE_FILETYPE) has #OFFLINE_FILE_TYPE_ENCODINGS set.
 * DynamoRIO's decode_from_copy() (or any other decoding library) can be used to
 * decode into a higher-level instruction representation.
 */
unsigned char encoding[MAX_ENCODING_LENGTH];
/**
 * Indicates whether the encoding field is the first instance of its kind for this
 * address.  This can be used to determine when to invalidate cached decoding
 * information.  This field may be set to true on internal file divisions and
 * not only when application code actually changed.
 */
bool encoding_is_new;
```

Example Traces

- Samples at https://github.com/DynamoRIO/drmemtrace_samples have been updated and can serve as test traces of the new fields

Example Code

```
if (TEST(OFFLINE_FILE_TYPE_ENCODINGS, shard->filetype) && memref.instr.encoding_is_new) {
    // The code may have changed: invalidate the cache.
    shard->worker->decode_cache.erase(memref.instr.addr);
}
if (shard->worker->decode_cache.find(memref.instr.addr) == shard->worker->decode_cache.end()) {
    if (TEST(OFFLINE_FILE_TYPE_ENCODINGS, shard->filetype)) {
        // The trace has instruction encodings inside it.
        decode_pc = memref.instr.encoding;
    } else {
        // Legacy trace support where we need the binaries.
        std::lock_guard<std::mutex> guard(mapper_mutex_);
        decode_pc = module_mapper_->find_mapped_trace_address(memref.instr.addr);
        if (!module_mapper_->get_last_error().empty()) return false;
    }
    // Now decode from `decode_pc` and populate the cache.
}
```

Fast Seeking

- Each software thread file is split into chunks of a fixed instruction count (say, 10 million instructions) with the chunks compressed separately and stored together as a .zip file
- Fast seeking is implemented by jumping to the nearest chunk and proceeding linearly from there
- Once-only information like instruction encodings are duplicated in each chunk (hidden by reader iterator)
- The underlying file change from .gz to .zip will not cause any disruption when using the provided reader library