# USI
# Dynamic Analysis Group
# Project

## Dynamically executed SIMD instructions profiler

# Introduction (1/2) - Project

**Statement**

○ Count the executed machine vector instructions (**SIMD** instructions) during a certain *Java Virtual Machine* (**JVM**) execution

**Requirements**

✓ The solution should report the numbers of executed vector instructions using counters

✓ The solution must work on *Linux Ubuntu*

✓ The solution must work on *OpenJDK 18*

# Introduction (2/2) - Features

**Not-Available features**

✗ Consider only the machine vector instructions emitted by the *Just-In-Time* (**JIT**) *compiler* (i.e. **C2**) (partially obtained, requires a more in-depth analysis of the JVM architecture)

✗ Expose a *Java* **API** to reset counters

**Available features**

✓ Report different counters for different types of instructions (e.g. *load*, *store*, *arithmetics*, etc.)

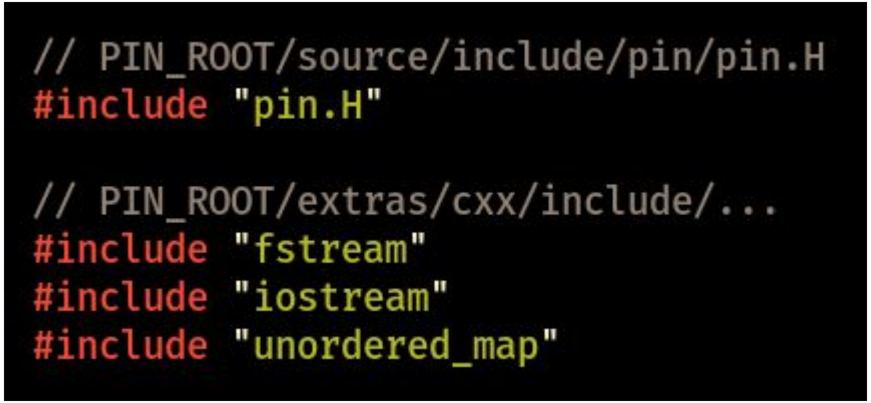
✓ Report thread-local counters (instead of a *single global counter* per *type*)

# Intel Pin (1/1) - Analysis

- Platform for creating analysis tools that comprises the following types of routines
  - **Instrumentation** - are called when code is about to be run (before recompilation) and enable the insertion of Analysis routines
  - **Analysis** - are called when the code associated with them is run
  - **Callback** - are called when specific conditions are met or when a certain event has occurred
- Portable architecture by isolating platform-specific code from generic code
- Performs a large collection of optimization techniques to reduce running time and memory overhead
- Used in **JIT mode** (it uses a JIT compiler to recompile the code and insert instrumentation)

# Pintool (1/9) - Headers (Portability)

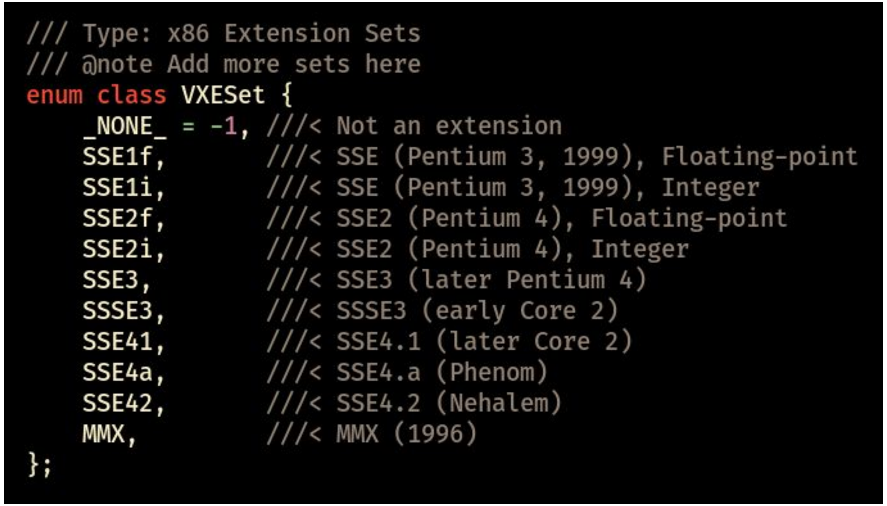
- A local copy of **Intel Pin 3.24 98612** is required
- `pin.H` is the entry-point header of the entire library
- The rest of the `#includes` refer to the portable `CXX stdlib` headers exposed by the library



```
// PIN_ROOT/source/include/pin/pin.H
#include "pin.H"

// PIN_ROOT/extras/cxx/include/...
#include "fstream"
#include "iostream"
#include "unordered_map"
```

# Pintool (2/9) - Data structures (x86 Extension sets)

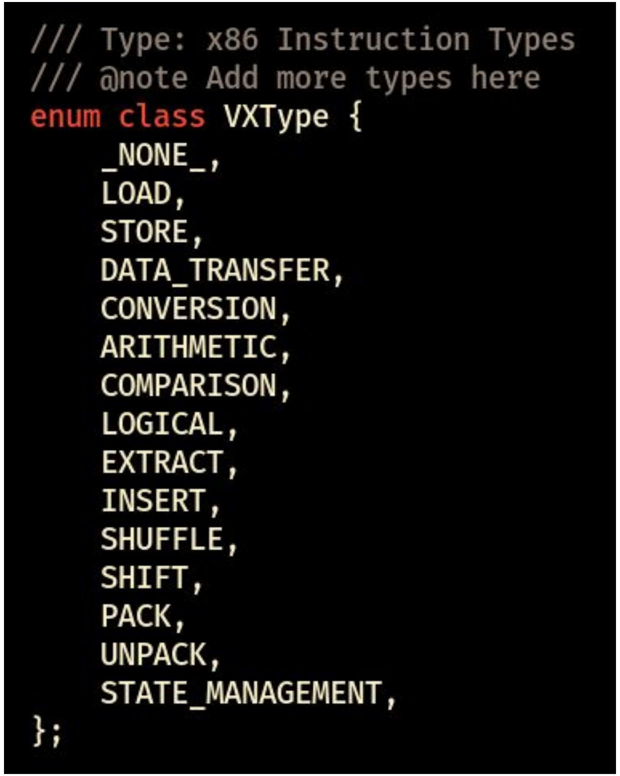
- Describes the available **x86 Extension Sets**
- More sets can be added here
- Used to filter the instructions by the required sets
- The `_NONE_` variant is used to describe an instruction that does not belongs to the defined Extesion sets



```
/// Type: x86 Extension Sets
/// @note Add more sets here
enum class VXESet {
    _NONE_ = -1, ///< Not an extension
    SSE1f,       ///< SSE (Pentium 3, 1999), Floating-point
    SSE1i,       ///< SSE (Pentium 3, 1999), Integer
    SSE2f,       ///< SSE2 (Pentium 4), Floating-point
    SSE2i,       ///< SSE2 (Pentium 4), Integer
    SSE3,        ///< SSE3 (later Pentium 4)
    SSSE3,       ///< SSSE3 (early Core 2)
    SSE41,       ///< SSE4.1 (later Core 2)
    SSE4a,       ///< SSE4.a (Phenom)
    SSE42,       ///< SSE4.2 (Nehalem)
    MMX,         ///< MMX (1996)
};
```

# Pintool (3/9) - Data structures (x86 Instruction types)

- Describes the available **x86 Instruction types**
- More types can be added here
- Used to filter the instructions by type
- The _NONE_ variant describes an unknown type (should not be used)

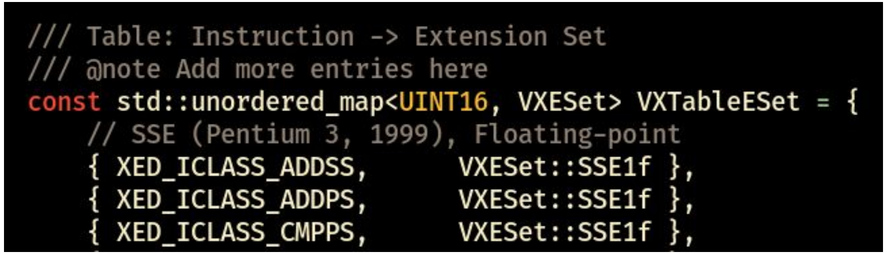


```
/// Type: x86 Instruction Types
/// @note Add more types here
enum class VXType {
    _NONE_,
    LOAD,
    STORE,
    DATA_TRANSFER,
    CONVERSION,
    ARITHMETIC,
    COMPARISON,
    LOGICAL,
    EXTRACT,
    INSERT,
    SHUFFLE,
    SHIFT,
    PACK,
    UNPACK,
    STATE_MANAGEMENT,
};
```

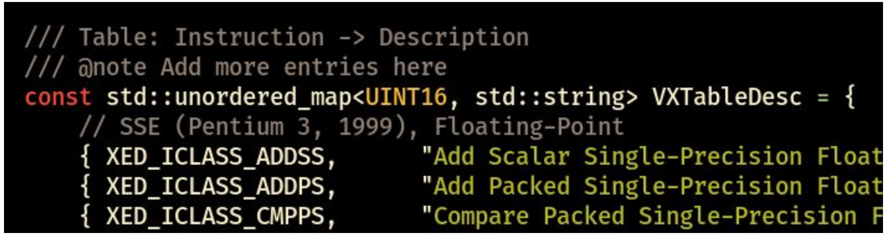
# Pintool (4/9) - Data structures (Associations)

- Definition of the associations between **Instructions** and **Extension sets**, **Descriptions** and **Types**
- The keys are **Instruction Opcodes** `UINT16` from `typedef enum { … } xed_iclass_enum_t` contained in `$PIN_ROOT/extras/xed-intel64/include/xed/xed-iclass-enum.h`

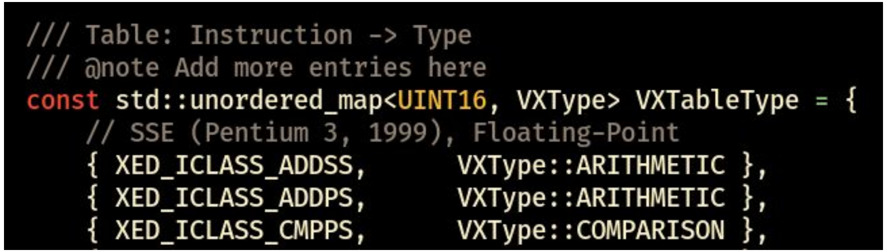
- More associations can be added here
- The *Intel Pin* **API** does not natively offer this fine-grained filtering
- **Improvements** - perfect hashing can be implemented



```
/// Table: Instruction -> Extension Set
/// @note Add more entries here
const std::unordered_map<UINT16, VXESet> VXTableESet = {
    // SSE (Pentium 3, 1999), Floating-point
    { XED_ICLASS_ADDSS,       VXESet::SSE1f },
    { XED_ICLASS_ADDPS,       VXESet::SSE1f },
    { XED_ICLASS_CMPPS,       VXESet::SSE1f },
```



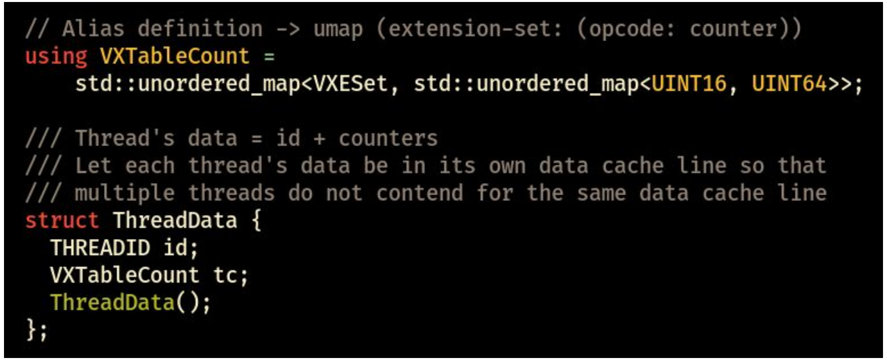
```
/// Table: Instruction -> Description
/// @note Add more entries here
const std::unordered_map<UINT16, std::string> VXTableDesc = {
    // SSE (Pentium 3, 1999), Floating-Point
    { XED_ICLASS_ADDSS,       "Add Scalar Single-Precision Float
    { XED_ICLASS_ADDPS,       "Add Packed Single-Precision Float
    { XED_ICLASS_CMPPS,       "Compare Packed Single-Precision F
```



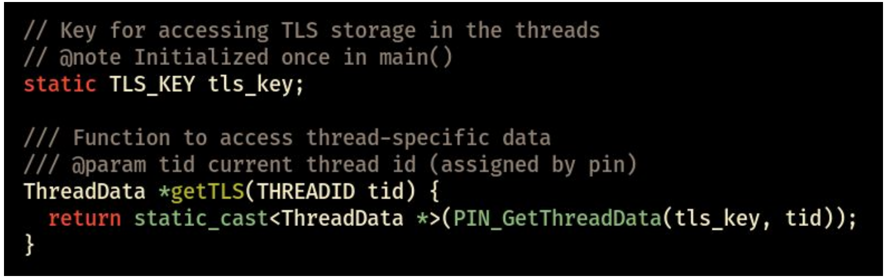
```
/// Table: Instruction -> Type
/// @note Add more entries here
const std::unordered_map<UINT16, VXType> VXTableType = {
    // SSE (Pentium 3, 1999), Floating-Point
    { XED_ICLASS_ADDSS,       VXType::ARITHMETIC },
    { XED_ICLASS_ADDPS,       VXType::ARITHMETIC },
    { XED_ICLASS_CMPPS,       VXType::COMPARISON },
```

# Pintool (5/9) - Data structures (Thread local data)

- Definition of the thread-local data
- The counters are defined inside a `unordered_map` (**Extension sets** driven) of `unordered_map` (**Instruction opcodes** driven)
- The data cache lines are separated to avoid race conditions (multithreading)
- The data are retrieved through a **TLS storage key** (intialized once in `main`)



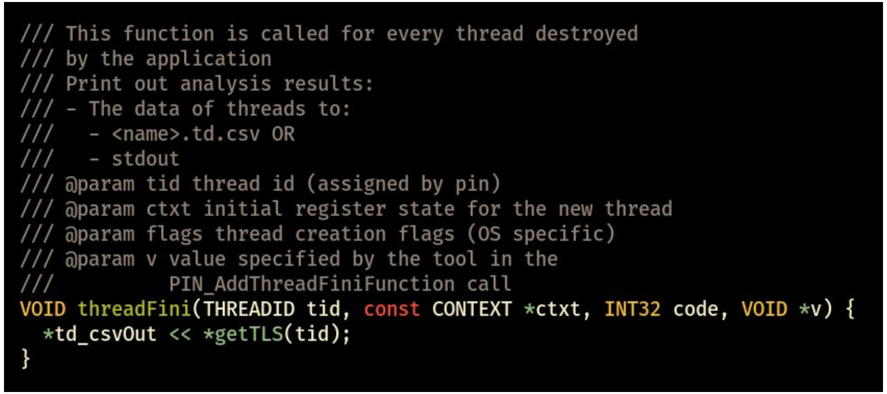
```
// Alias definition -> umap (extension-set: (opcode: counter))
using VXTableCount =
    std::unordered_map<VXESet, std::unordered_map<UINT16, UINT64>>;

/// Thread's data = id + counters
/// Let each thread's data be in its own data cache line so that
/// multiple threads do not contend for the same data cache line
struct ThreadData {
  THREADID id;
  VXTableCount tc;
  ThreadData();
};
```



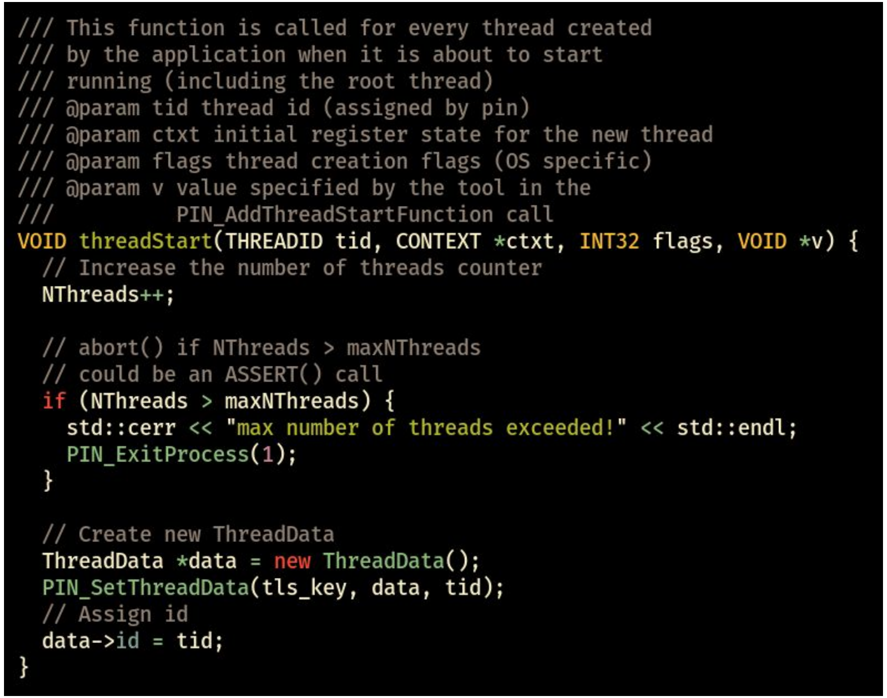
```
// Key for accessing TLS storage in the threads
// @note Initialized once in main()
static TLS_KEY tls_key;

/// Function to access thread-specific data
/// @param tid current thread id (assigned by pin)
ThreadData *getTLS(THREADID tid) {
  return static_cast<ThreadData *>(PIN_GetThreadData(tls_key, tid));
}
```

# Pintool (6/9) - Callback routines (Thread related)

- The threadStart hook is called for every thread created by the application (when it is about to start) and performs the ThreadData construction and initialization
- The threadFini hook is called for every thread destroyed by the application and prints out ThreadData in **CSV** format



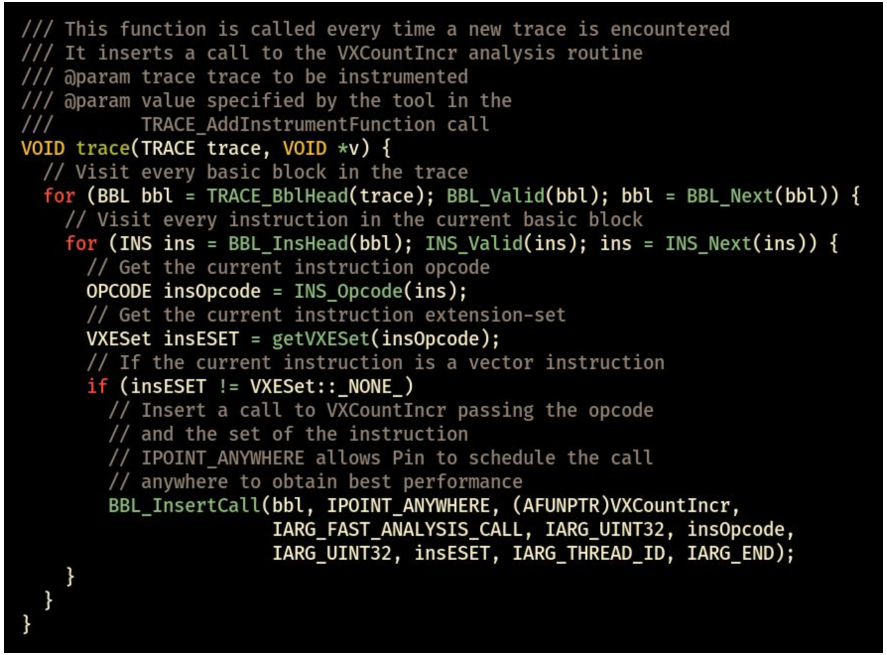
```
/// This function is called for every thread destroyed
/// by the application
/// Print out analysis results:
/// - The data of threads to:
///   - <name>.td.csv OR
///   - stdout
/// @param tid thread id (assigned by pin)
/// @param ctxt initial register state for the new thread
/// @param flags thread creation flags (OS specific)
/// @param v value specified by the tool in the
///          PIN_AddThreadFiniFunction call
VOID threadFini(THREADID tid, const CONTEXT *ctxt, INT32 code, VOID *v) {
  *td_csvOut << *getTLS(tid);
}
```



```
/// This function is called for every thread created
/// by the application when it is about to start
/// running (including the root thread)
/// @param tid thread id (assigned by pin)
/// @param ctxt initial register state for the new thread
/// @param flags thread creation flags (OS specific)
/// @param v value specified by the tool in the
///          PIN_AddThreadStartFunction call
VOID threadStart(THREADID tid, CONTEXT *ctxt, INT32 flags, VOID *v) {
  // Increase the number of threads counter
  NThreads++;

  // abort() if NThreads > maxNThreads
  // could be an ASSERT() call
  if (NThreads > maxNThreads) {
    std::cerr << "max number of threads exceeded!" << std::endl;
    PIN_ExitProcess(1);
  }

  // Create new ThreadData
  ThreadData *data = new ThreadData();
  PIN_SetThreadData(tls_key, data, tid);
  // Assign id
  data->id = tid;
}
```

# Pintool (7/9) - Instrumentation routine

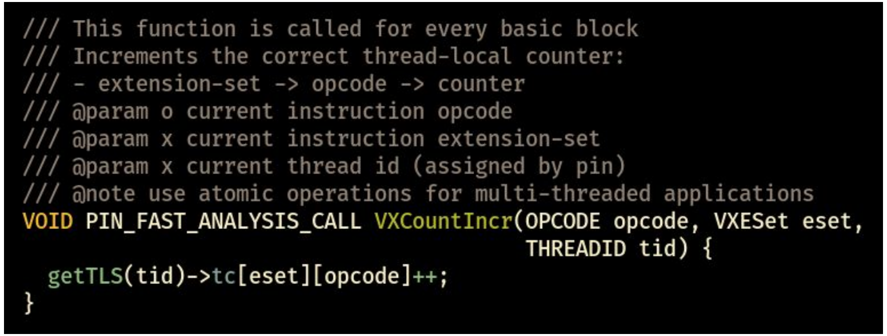
- The `trace` function is called every time a new trace is encountered
- The function inserts a call to the `VXCountIncr` *analysis routine*
- `IPOINT_ANYWHERE` is used to let *Pin* schedule the call anywhere and obtain best performance
- The function visit every basic block in the trace and every instruction inside it
- The function inserts the *analysis routine* call only if the current instruction belongs to one of the defined **Extension sets** (filtering)



```
/// This function is called every time a new trace is encountered
/// It inserts a call to the VXCountIncr analysis routine
/// @param trace trace to be instrumented
/// @param value specified by the tool in the
///        TRACE_AddInstrumentFunction call
VOID trace(TRACE trace, VOID *v) {
  // Visit every basic block in the trace
  for (BBL bbl = TRACE_BblHead(trace); BBL_Valid(bbl); bbl = BBL_Next(bbl)) {
    // Visit every instruction in the current basic block
    for (INS ins = BBL_InsHead(bbl); INS_Valid(ins); ins = INS_Next(ins)) {
      // Get the current instruction opcode
      OPCODE insOpcode = INS_Opcode(ins);
      // Get the current instruction extension-set
      VXESet insESET = getVXESet(insOpcode);
      // If the current instruction is a vector instruction
      if (insESET != VXESet::_NONE_)
        // Insert a call to VXCountIncr passing the opcode
        // and the set of the instruction
        // IPOINT_ANYWHERE allows Pin to schedule the call
        // anywhere to obtain best performance
        BBL_InsertCall(bbl, IPOINT_ANYWHERE, (AFUNPTR)VXCountIncr,
                       IARG_FAST_ANALYSIS_CALL, IARG_UINT32, insOpcode,
                       IARG_UINT32, insESET, IARG_THREAD_ID, IARG_END);
    }
  }
}
```

# Pintool (8/9) - Analysis routine

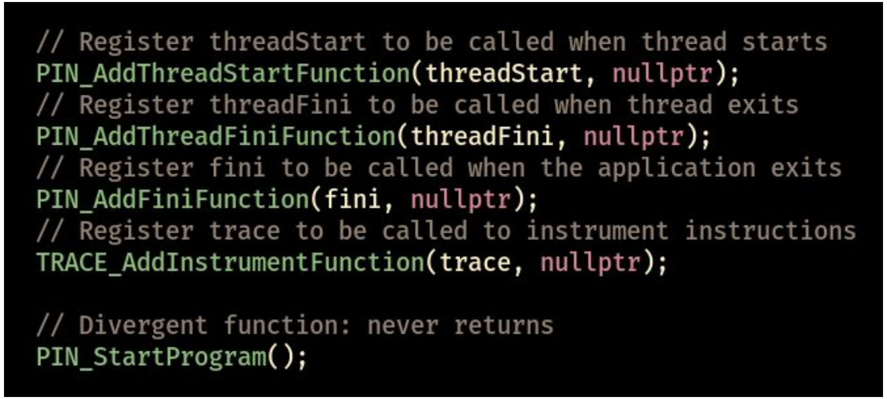
- The VXCountIncr function is called for every filtered instruction within each basic block
- The function increments by 1 the correct thread-local counter
- The macro PIN_FAST_ANALYSIS_CALL is used to let *Pin* perform a faster linkage for this call, increasing performances



```
/// This function is called for every basic block
/// Increments the correct thread-local counter:
/// - extension-set -> opcode -> counter
/// @param o current instruction opcode
/// @param x current instruction extension-set
/// @param x current thread id (assigned by pin)
/// @note use atomic operations for multi-threaded applications
VOID PIN_FAST_ANALYSIS_CALL VXCountIncr(OPCODE opcode, VXESet eset,
                                        THREADID tid) {
  getTLS(tid)->tc[eset][opcode]++;
}
```

# Pintool (9/9) - Application lifecycle

- Defined within the main function (not shown entirely)
- Registration of the threadStart and threadFini hooks to be called at the creation and destruction of a thread, respectively
- Registration of the fini hook (not showed) to be called when the application exits (simply prints out the number of created threads in **CSV** format)
- Registration of the trace hook to be called to instrument filtered instructions
- Call to the divergent function PIN_StartProgram, that never returns



```
// Register threadStart to be called when thread starts
PIN_AddThreadStartFunction(threadStart, nullptr);
// Register threadFini to be called when thread exits
PIN_AddThreadFiniFunction(threadFini, nullptr);
// Register fini to be called when the application exits
PIN_AddFiniFunction(fini, nullptr);
// Register trace to be called to instrument instructions
TRACE_AddInstrumentFunction(trace, nullptr);

// Divergent function: never returns
PIN_StartProgram();
```

# Usage (1/2) - Infrastructure

**Description**

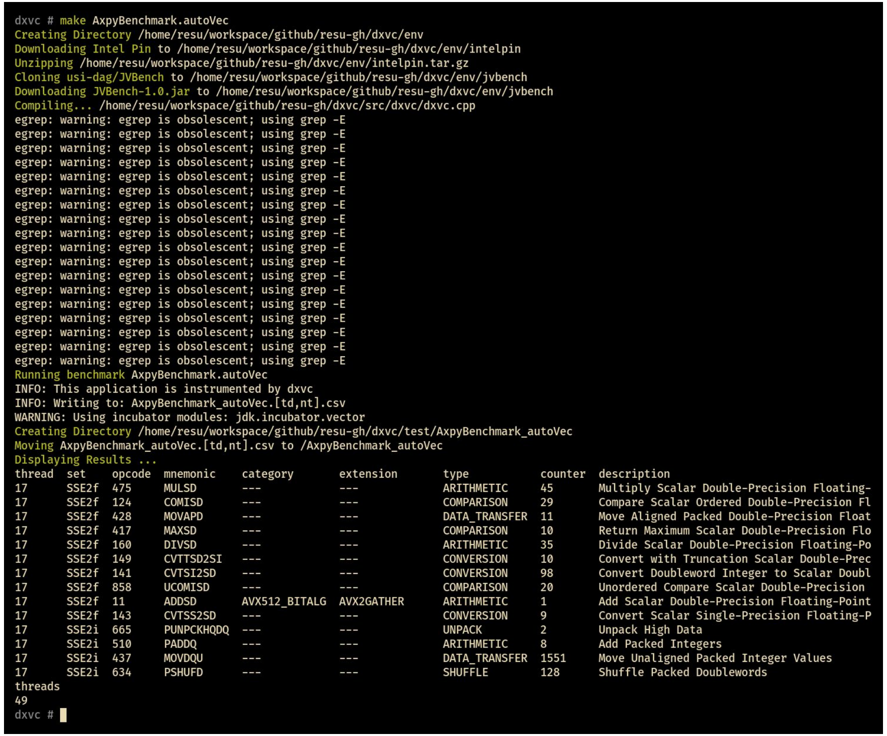
- Simple `make` driven infrastructure to test the solution (all the following details are automatically handled within the `Makefile`)
- Few dependencies with standard tools (e.g. `make`, `coreutils`, `linux-utils`, etc.)
- To compile the **Pintool** a C++ compiler, a local copy of *Intel Pin 3.24 98612* and the definition of the `PIN_ROOT` *environment variable* are required
- **JVBench** (*JVBench-1.0.jar*) is used for testing and all the tests in the suite are available (*OpenJDK* >= 16 is required)

**Testing**

- Simply call `make` followed by a (list) target test name(s) (call `make` or `make help` for info)
- The `jdk.incubator.vector` module is automatically included
- A simple `Python` script for bulk executions is also provided

# Usage (2/2) - Testing (example)

- Automatically builds the dependencies (*Intel Pin* and *JVBench*) environment within the `env` folder
- Compiles the Pintool
- Instrument and run the target test application (the execution is silenced and the result is redirected to a log file)
- Prints out the test results (**CSV**)
- Moves the test results within the `test` folder

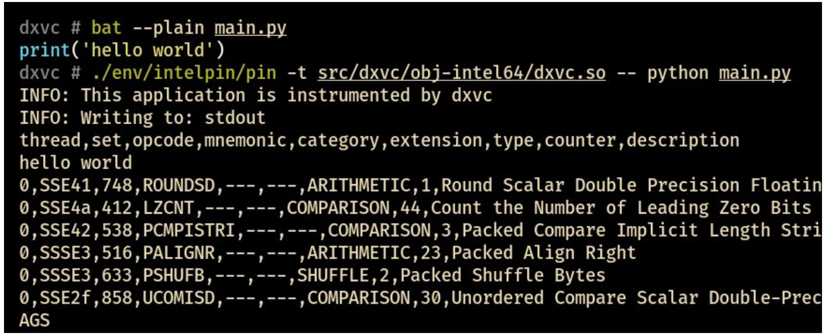


```
dxvc # make AxpyBenchmark.autoVec
Creating Directory /home/resu/workspace/github/resu-gh/dxvc/env
Downloading Intel Pin to /home/resu/workspace/github/resu-gh/dxvc/env/intelpin
Unzipping /home/resu/workspace/github/resu-gh/dxvc/env/intelpin.tar.gz
Cloning usi-dag/JVBench to /home/resu/workspace/github/resu-gh/dxvc/env/jvbench
Downloading JVBench-1.0.jar to /home/resu/workspace/github/resu-gh/dxvc/env/jvbench
Compiling... /home/resu/workspace/github/resu-gh/dxvc/src/dxvc/dxvc.cpp
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
egrep: warning: egrep is obsolescent; using grep -E
Running benchmark AxpyBenchmark.autoVec
INFO: This application is instrumented by dxvc
INFO: Writing to: AxpyBenchmark_autoVec.[td,nt].csv
WARNING: Using incubator modules: jdk.incubator.vector
Creating Directory /home/resu/workspace/github/resu-gh/dxvc/test/AxpyBenchmark_autoVec
Moving AxpyBenchmark_autoVec.[td,nt].csv to /AxpyBenchmark_autoVec
Displaying Results ...
thread  set    opcode  mnemonic    category       extension       type           counter  description
17      SSE2f  475     MULSD       ---            ---             ARITHMETIC     45       Multiply Scalar Double-Precision Floating-
17      SSE2f  124     COMISD      ---            ---             COMPARISON     29       Compare Scalar Ordered Double-Precision Fl
17      SSE2f  428     MOVAPD      ---            ---             DATA_TRANSFER  11       Move Aligned Packed Double-Precision Float
17      SSE2f  417     MAXSD       ---            ---             COMPARISON     10       Return Maximum Scalar Double-Precision Flo
17      SSE2f  160     DIVSD       ---            ---             ARITHMETIC     35       Divide Scalar Double-Precision Floating-Po
17      SSE2f  149     CVTTSD2SI   ---            ---             CONVERSION     10       Convert with Truncation Scalar Double-Prec
17      SSE2f  141     CVTSI2SD    ---            ---             CONVERSION     98       Convert Doubleword Integer to Scalar Doubl
17      SSE2f  858     UCOMISD     ---            ---             COMPARISON     20       Unordered Compare Scalar Double-Precision
17      SSE2f  11      ADDSD       AVX512_BITALG  AVX2GATHER      ARITHMETIC     1        Add Scalar Double-Precision Floating-Point
17      SSE2f  143     CVTSS2SD    ---            ---             CONVERSION     9        Convert Scalar Single-Precision Floating-P
17      SSE2i  665     PUNPCKHQDQ  ---            ---             UNPACK         2        Unpack High Data
17      SSE2i  510     PADDQ       ---            ---             ARITHMETIC     8        Add Packed Integers
17      SSE2i  437     MOVDQU      ---            ---             DATA_TRANSFER  1551     Move Unaligned Packed Integer Values
17      SSE2i  634     PSHUFD      ---            ---             SHUFFLE        128      Shuffle Packed Doublewords
threads
49
dxvc #
```

# Conclusion

**Advantages**

- The Pintool is not strictly binded with the **JVM** (e.g. execution on a simple “`hello world`” `Python` script)



```
dxvc # bat --plain main.py
print('hello world')
dxvc # ./env/intelpin/pin -t src/dxvc/obj-intel64/dxvc.so -- python main.py
INFO: This application is instrumented by dxvc
INFO: Writing to: stdout
thread,set,opcode,mnemonic,category,extension,type,counter,description
hello world
0,SSE41,748,ROUNDSD,---,---,ARITHMETIC,1,Round Scalar Double Precision Floatin
0,SSE4a,412,LZCNT,---,---,COMPARISON,44,Count the Number of Leading Zero Bits
0,SSE42,538,PCMPISTRI,---,---,COMPARISON,3,Packed Compare Implicit Length Stri
0,SSSE3,516,PALIGNR,---,---,ARITHMETIC,23,Packed Align Right
0,SSSE3,633,PSHUFB,---,---,SHUFFLE,2,Packed Shuffle Bytes
0,SSE2f,858,UCOMISD,---,---,COMPARISON,30,Unordered Compare Scalar Double-Prec
AGS
```

**Limitations**

- **JVBench** tests CLI arguments are not dynamically managed (use an environment variable)
- `java` CLI arguments are not dynamically managed (use an environment variable)
- Test results are not discriminated (add timestamps to filenames)
- Portability could be improved (e.g. `Docker`, `nix`, etc.)

# Thanks for your attention