OMR Architecture: Vector IL Opcodes

April 12th 2018

Gita Koblents

IBM Runtime Technologies

IBM

Outline

- Motivation
- Discussion
- Proposal

Motivation

- **We would like to support vector operations for various vector lengths – 128, 256, 512-bit**
  - To exploit current Intel and future vector register sizes on Power and z
  - Can be used by auto-SIMD
  - Are necessary for the Panama project
  - We would like to be able to use different sizes in one compilation
- **Current approach**
  - 128-bit length is implied
  - TR::DataTypes describe vector layout precisely, e.g. **VectorInt32** is 4 32-bit integers
  - Opcodes are ‘typelesss’, e.g. **vadd**, **vconst,** etc. but data type can be derived
    - From children
    - From symbol reference
    - Cached on the node, e.g. for **vconst**
- **Limitations of the current approach**
  - Only one length is supported
  - Precise description of the layout requires adding new data type for every possible combination of vector length and element type
  - Typeless opcodes reduce the number of opcodes but are error prone, e.g. incorrect type can be derived while node being constructed or modified

Discussion: terminology

- **Data:** some number of bits at a certain memory location. In JIT, we represent data as symbol references. JIT symbols have data type associated with them.
- **Data Type: <size, Int/FP, scale>.** For example <128, Int, 4> is vector of 4 32-bit integers.
- In high level languages, programmers describe data types precisely and then specify overloaded operations. Compilers quickly transform overloaded operations into precise ones, so type information on the data might not always be needed.
- **Compiler Actions**
  A program needs to interpret the data and perform various actions on it. Compiler actions include:
  - Mapping on stack
  - Putting into registers
  - Performing various operations (load, store, add, sub, etc) on registers and memory
- **Operation: <operator, size, Int/FP, scale>** e.g. <add, 128, Int, 4> is an addition of two vectors consisting of 4 32-bit integers.
- In JIT, operations and their parameters are represented as nodes. Nodes have opcodes and possibly other info describing the operation. Nodes need to have full type information available.

Discussion

- **Where to keep type information**

  We argued that data type should only contain **<size, Int/FP>**. Operation will still have full info available: **<operator, size, Int/FP, scale>**

- **How to represent operation information**
  - Should we represent operations in a flattened way, e.g. have an enum for every possible quadruple above? This creates big number of opcode enums, especially if vector-to-vector conversions are needed:
    - | non-conversion operators | * | types | + | types | **2
  - Should we represent it as a pair of opcode and data type? This requires measures to be taken to make sure any node has both opcode and data type set at all times.
  - Note: vector conversion operations are usually noops, we suspect we currently don't need them (eg. they are not generated by auto-SIMD)

Proposal

- Flatten all the type information in the opcodes. For example, we will have the following opcodes:

  i128add // addition of two integers of size 128
  i128add_4 // addition of two vectors consisting of 4 32-bit integers each
  i128add_2 // addition of two vectors consisting of 2 64-bit integers each

- We concluded that vector types are similar to aggregates where the exact layout is not reflected in the OMR type. Usually, it's only the size that is needed when we inquire data type from a SymbolReference.

  Because of that, we decided to limit OMR types to reflect only Int/FP and the size. For example, we will have OMR type: **Int128**, which will be used for both single 128-bit integer as well as any 128-bit vector consisting of smaller integral types: 2 longs, 4 ints, etc.

- If we find it is sometimes necessary to know if type is a vector type we will add **isVector** field into **SymbolReference**. Alternatively, we can have **VectorInt128** type in addition to **Int128**.

- If we find we also need to know vector layout, we will add **scale** field into **SymbolReference**, eg. 2, 4, 8, etc. We suspect it will be rarely needed since full type information is encoded into the opcode and we usually create new nodes based on existing nodes and opcodes.