Supporting Vector Expressions and Patterns in MLton

Student: Krishna Ravikumar (kr5748@rit.edu)
Advisor: Dr. Matthew Fluet (mtf@cs.rit.edu)
Department of Computer Science, Rochester Institute of Technology

INTRODUCTION

- Standard ML (SML) is a general-purpose, modular, functional programming language with compile-time type checking and type inference.
- Two of the most popular compilers for SML are Standard ML of New Jersey (SML/NJ) and MLton.
- MLton is an open-source, whole-program, optimizing Standard ML compiler [1].
- Modern compilers, in addition to supporting the complete specification of a language, sometimes offer additional features exclusive to their compilers.
- SML/NJ offers one such feature that allows creation of vectors using a special syntax, vector expressions.

MOTIVATION

- Vectors are homogenous, immutable sequences with constant-time access.
- In MLton, the only way to create vectors is by calling functions from the Vector structure of the SML Basis Library.
- Direct pattern matching on vectors is not currently possible.
- Maintaining consistency between SML/NJ and MLton (two leading SML compilers).

GOALS

- Allow creation of vectors using new syntax in MLton.
- Find opportunities to optimize vectors created using vector expressions.
- Use vector expressions internally and observe code size benefits.
- Establish a foundation upon which vector patterns can be supported.

MLton COMPILER OVERVIEW

<table>
<thead>
<tr>
<th>Source</th>
<th>Intermediate Language</th>
<th>Optimization Passes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front End</td>
<td>AST</td>
<td></td>
</tr>
<tr>
<td>Elaborate</td>
<td>CoreML</td>
<td>CoreMSimplify</td>
</tr>
<tr>
<td>Defunctorize</td>
<td>XML</td>
<td>XMLSimplify</td>
</tr>
<tr>
<td>Monomorphise</td>
<td>SXML</td>
<td>SXMLSimplify</td>
</tr>
<tr>
<td>LclosureConvert</td>
<td>SSA</td>
<td>SSA2Simplify</td>
</tr>
<tr>
<td>ToSSA</td>
<td>SSA2</td>
<td>SSA2Simplify</td>
</tr>
<tr>
<td>ToSSA2</td>
<td>SSA</td>
<td>SSA2Simplify</td>
</tr>
<tr>
<td>ToMachine</td>
<td>Machine</td>
<td>RSSA2Simplify</td>
</tr>
<tr>
<td>Codegen</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: Overview of MLton's architecture

- The rectangles denote the phases of the compiler that were modified significantly.

IMPLEMENTATION

FRONT END

- The scanner was modified to include a new token Hash-bucket ( # ). This token is unique and used only for the vector expression syntax.
- The parser makes use of the above token and identifies vector expressions and their constituent element expressions.

DEFUNCTORIZE (Early translation)

- The defunctorize pass was modified to translate vector expressions into simpler expressions.
- Vector expressions are turned into XML primitives that perform the following task:
- Create an uninitialized array → Update the values of all positions → Cast array to a vector
- The functionality and new syntax were validated using a toy example.

DEFUNCTORIZE (Late translation)

- New primitive (Vector_vector) was introduced to represent vector expression creation.
- Translation of vector expressions was moved to the SSA Simplify pass to allow optimization passes to work with vector expressions.
- Closure Convert pass was modified to "understand" vector expressions.
- Defunctorize pass was modified to propagate vector expressions deeper into the compiler.

RESULTS AND CONCLUSIONS

- MLton supports new syntax for the creation of vector expressions (Figure 1).
- MLton has a new internal primitive for creating vectors using vector expressions.
- Vector expressions are treated as reusable constants. This reduces the amount of memory used in programs involving repetitive creation of vector expressions.
- MLton uses vector expressions internally for creation of small vectors of sizes 0 through 6.

PERFORMANCE COMPARISON

<table>
<thead>
<tr>
<th>Property</th>
<th>Vector expression (Test A)</th>
<th>Vector Tabulate (Test B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Running Time (ms)</td>
<td>2,424</td>
<td>2,467</td>
</tr>
<tr>
<td>GC - # of copies</td>
<td>734</td>
<td>917</td>
</tr>
<tr>
<td>GC - bytes copied</td>
<td>9,976,528</td>
<td>12,405,176</td>
</tr>
<tr>
<td>Total allocated bytes</td>
<td>128,014,368</td>
<td>160,014,368</td>
</tr>
</tbody>
</table>

Figure 3: Test example (-A) Vector expression (B) Vector tabulate

- Figure 3 shows an example program that creates a vector and prints out its element 500,000 times.
- Test A - Vector created using vector expression
- Test B - Vector created using vector tabulate

The following data was gathered from the example. All values are averaged over 20 runs.

(Fixed Certificate Collector)

| Size of the MLton executable reduced by 0.8 MB approximately (32803 KB → 32005 KB).

FUTURE WORK

- Optimize Vector sub operations on vectors created using vector expressions aggressively.
- Implement ‘Loop Invariant Code Motion’ to move vector expressions outside the body of loops.
- Find further optimizations on vector expressions.
- Extend support for pattern matching on vector expressions (vector patterns).

REFERENCES

1. MLton (http://mlton.org).