Detecting SIMDization Opportunities through Static/Dynamic Dependence Analysis

Olivier Aumage, Denis Barthou, Christopher Haine,
Tamara Meunier

University of Bordeaux / INRIA Runtime

August 27, 2013

SIMD: Key for performance

SIMD instructions are essential for reaching high levels of performance.

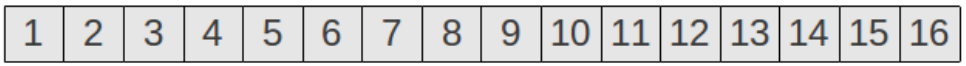
SSE vectors : 4 x 32-bit floats

128 bits

AVX vectors : 8 x 32-bit floats

256 bits

MIC vectors : 16 x 32-bit floats



512 bits

Hand vectorization

Explicit vectorization is complex and time consuming.

- Assembly instructions or intrinsics are not portable
- Language extensions such as GCC vector extensions only offer limited subset of arithmetic operations

s1213 of TSVC benchmark

```
for (int i = 1; i < LEN-2; i++) {
    b[i  ] = a[i+1]*d[i  ];
    a[i+1] = b[i  ]+c[i+1];
}
```

Hand vectorization

Explicit vectorization is complex and time consuming.

- Assembly instructions or intrinsics are not portable
- Language extensions such as GCC vector extensions only offer limited subset of arithmetic operations

s1213 of TSVC benchmark

```
for (int i = 1; i < LEN-2; i++) {
    b[i  ] = a[i+1]*d[i  ];
    a[i+1] = b[i  ]+c[i+1];
}
```

```
for (int i = 4; i < LEN-4; i+=4) {
    rA = _mm_loadu_ps(&a[i+1]);
    rC = _mm_loadu_ps(&c[i+1]);
    rD = _mm_load_ps(&d[i]);
    rB = _mm_mul_ps(rA,rD);
    rA = _mm_add_ps(rB,rC);
    _mm_store_ps(&b[i],rB);
    _mm_storeu_ps(&a[i+1],rA);
}
```

Compilers vectorization

Automatic vectorization can fail:

- lack of semantic
- data reshaping
- reduction
- ...

Hints about what caused vectorization to fail are hard to understand.

Examples with GCC 4.6.3:

```
tsc.c:1986: note: not vectorized, possible dependence between
            data-refs b[D.18123_4] and b[i_46] (->s1213)
tsc.c:121: note: not vectorized: complicated access pattern.
tsc.c:5476: note: not vectorized: relevant stmt not supported:
            *D.16720_10 = D.16721_11;
```

Our Contribution

A new tuning approach for vectorization based on binary code:

- Giving hints to the user about issues that hinder vectorization

  ex: loop transformation, resheduling, ...

- Based on static and dynamic dependence analysis

- Implemented in MAQAO

  performance tuning tool, based on binary code analysis

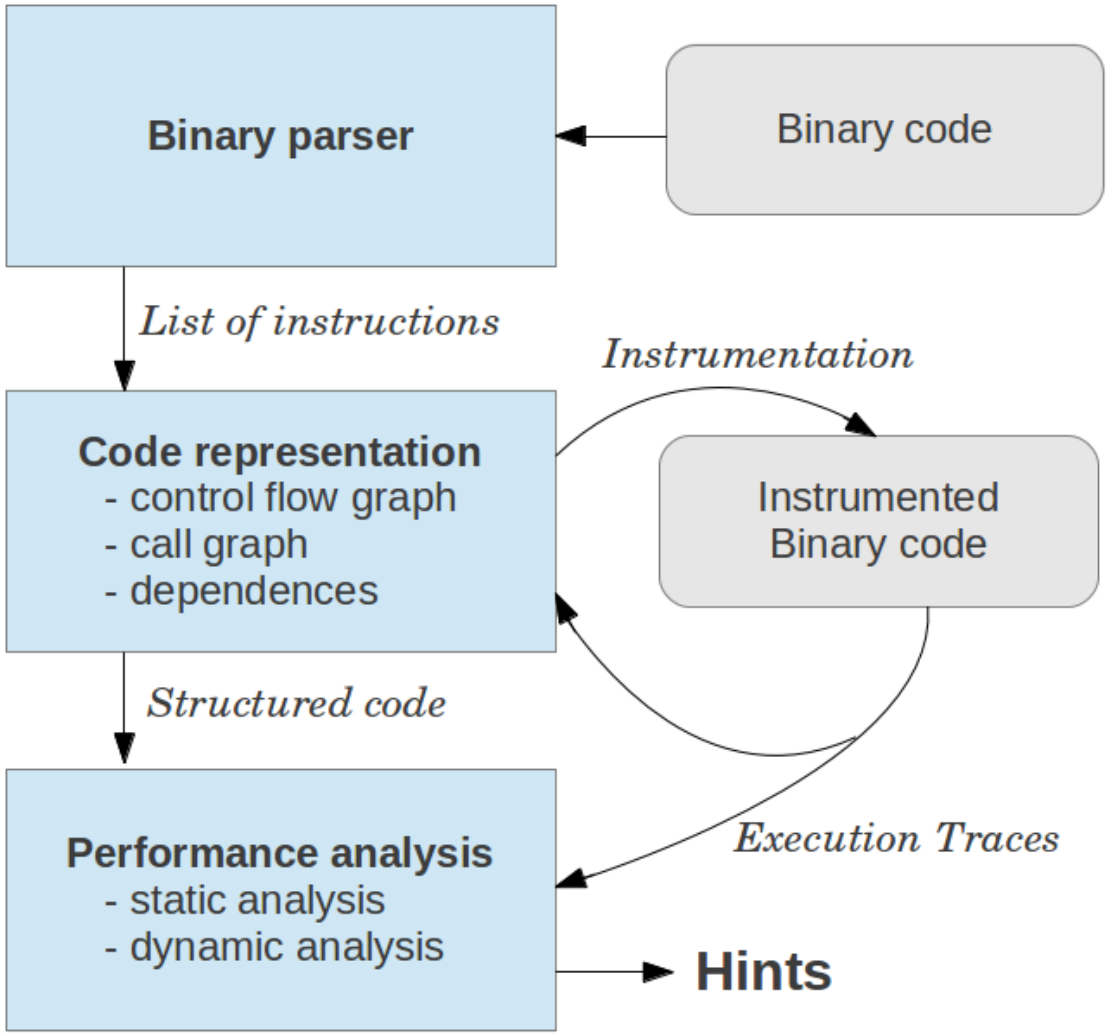
- Tested on TSVC benchmark

  Maleki, S., Gao, Y., Garzarn, M.J., Wong, T., Padua, D.A.: *An evaluation of vectorization compilers. In: International Conference on Parallel Architectures and Compilation Techniques* (PACT) (2011)

  151 functions

Outline

MAQAO: Performance tuning tool



2 - Register Dependence Analysis

Done statically, corresponding to static single assignment (SSA) form analysis.

Differentiate dependences used in address computation from those used in actual computation (to vectorize).
Address computation part: Information about indirections and control.

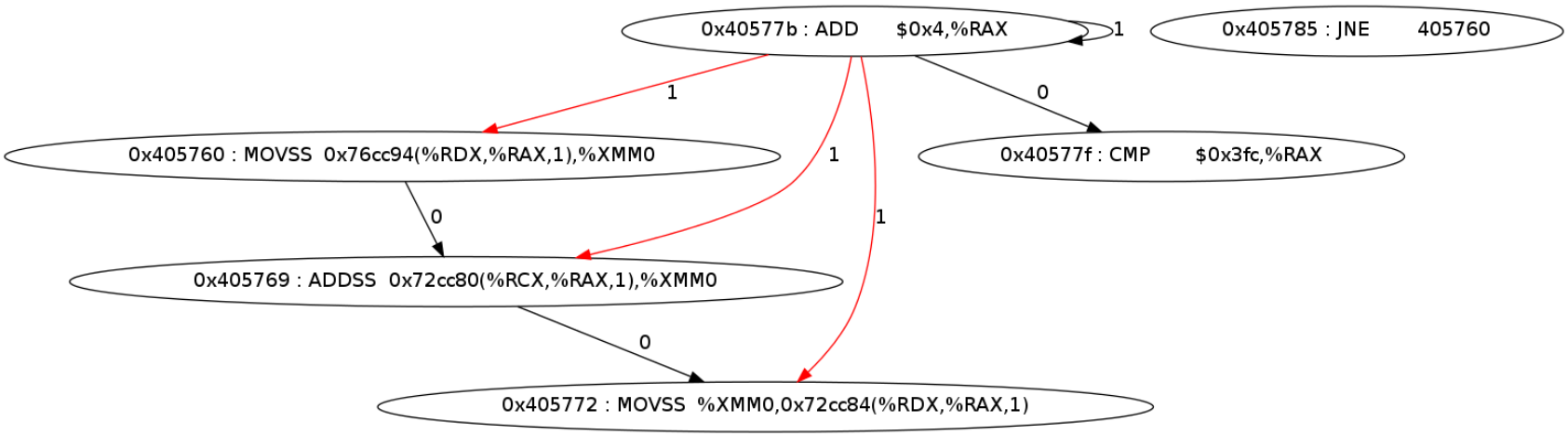


Figure: s119 of TSVC

2 - Register Dependence Analysis

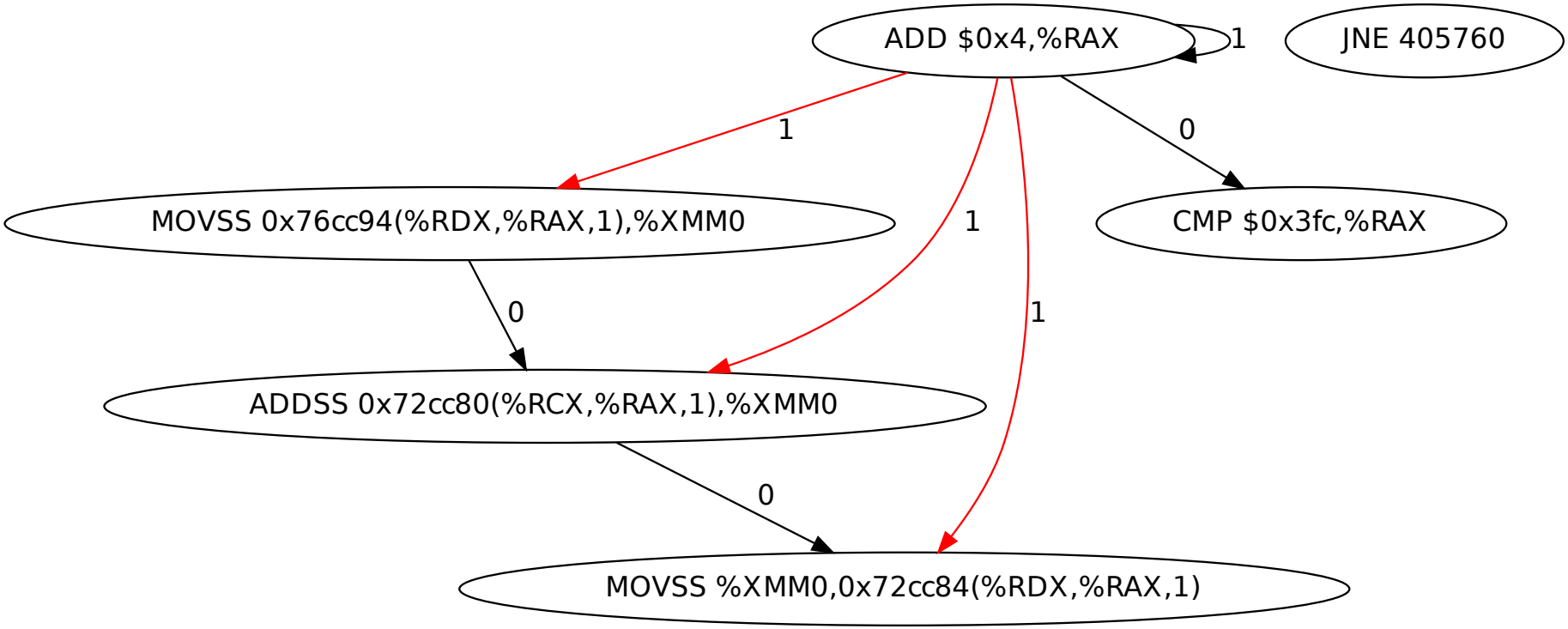


Figure: s119 of TSVC

3 - Memory dependences : Collecting Memory Traces

Method: We use MAQAO for collecting traces

- All loads and stores in innermost loops are traced
- Addresses are compressed on the fly

```
for (int i = 1; i < 256; i++)
  for (int j = 1; j < 256; j++)
    aa[i][j] = aa[i-1][j-1]+bb[i][j];
```

code of s119

```
for i0 = 0 to 254
  for i1 = 0 to 254
    read 0x72cc80 + 1024*i0 + 4*i1
```

trace for aa[i-1][j-1]

3 - Memory dependences : Collecting Memory Traces

Method: We use MAQAO for collecting traces

- All loads and stores in innermost loops are traced
- Addresses are compressed on the fly

```
for (int i = 1; i < 256; i++)
  for (int j = 1; j < 256; j++)
    aa[i][j] = aa[i-1][j-1]+bb[i][j];
```

code of s119

```
for i0 = 0 to 254
  for i1 = 0 to 254
    write 0x72d084 + 1024*i0 + 4*i1
```

trace for aa[i][j]

3 - Memory Dependence Analysis

Distance vector is computed from memory traces:

```
for i0 = 0 to 254
  for i1 = 0 to 254
    read 0x72cc80 + 1024*i0 + 4*i1
```

```
for j0 = 0 to 254
  for j1 = 0 to 254
    write 0x72d084 + 1024*j0 + 4*j1
```

Write and read access the same memory location for $i0 = j0 + 1$ and $i1 = j1 + 1$.

$\rightarrow$ **distance vector = 1, 1**

3 - Memory Dependence Analysis

Distance vector is computed from memory traces:

```
for i0 = 0 to 254
  for i1 = 0 to 254
    read 0x72cc80 + 1024*i0 + 4*i1
```

```
for j0 = 0 to 254
  for j1 = 0 to 254
    write 0x72d084 + 1024*j0 + 4*j1
```

Write and read access the same memory location for
$i0 = j0 + 1$ and $i1 = j1 + 1$.
$\rightarrow$ **distance vector = 1, 1**

General case:

- no dependence if no intersection
- " * " if unmatching boundaries or strides

3 - Memory Dependence Analysis

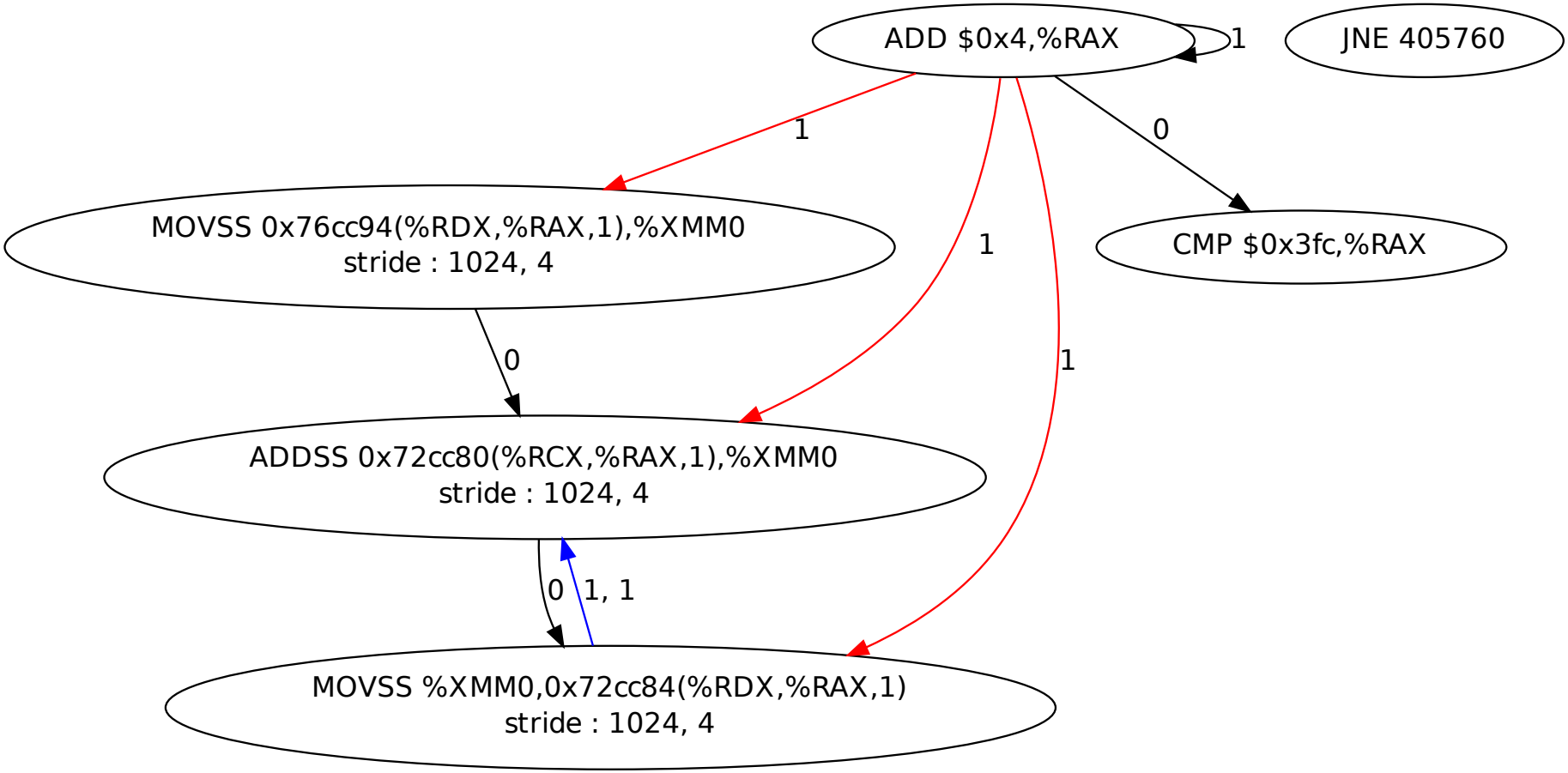


Figure: s119

4 - SIMDization Analysis and Hints

The code is SIMDizable if for the computational part of its graph:

- There is no cycle
- Or for each cycle:
  - The cumulative weight is greater than the width of the SIMD vectors.
  - Or all the instructions of the cycle are one of the following types: add, mul, max, min.
    $\rightarrow$ Reduction

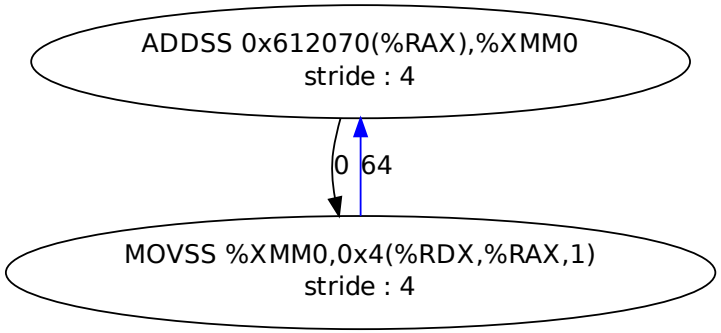


Figure: subgraph of s424

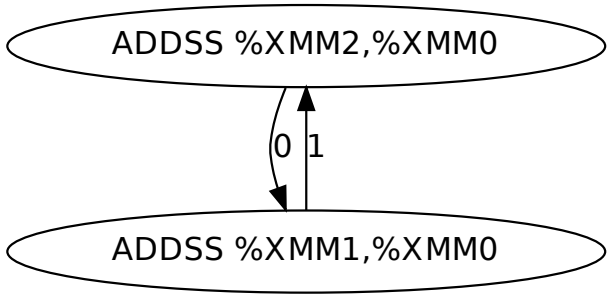


Figure: subgraph of s221

4 - Code Transformations Hints

Hints for code transformations required for SIMDization are based on dependence graph, strides and control flow graph.

- Data alignment

  first address offset is not a multiple of the vector

4 - Code Transformations Hints

Hints for code transformations required for SIMDization are based on dependence graph, strides and control flow graph.

- Data alignment

  first address offset is not a multiple of the vector

- Rescheduling

  dependences of distance 1 on scalars, with no cycle

4 - Code Transformations Hints

Hints for code transformations required for SIMDization are based on dependence graph, strides and control flow graph.

- Data alignment

  first address offset is not a multiple of the vector

- Rescheduling

  dependences of distance 1 on scalars, with no cycle

- Loop transformations, loop reversal

  strides in the wrong way, or negative strides

4 - Code Transformations Hints

Hints for code transformations required for SIMDization are based on dependence graph, strides and control flow graph.

- Data alignment

  first address offset is not a multiple of the vector

- Rescheduling

  dependences of distance 1 on scalars, with no cycle

- Loop transformations, loop reversal

  strides in the wrong way, or negative strides

- Data reshaping required

  the smallest stride does not correspond to the data length

4 - Code Transformations Hints

Hints for code transformations required for SIMDization are based on dependence graph, strides and control flow graph.

- Data alignment

  first address offset is not a multiple of the vector

- Rescheduling

  dependences of distance 1 on scalars, with no cycle

- Loop transformations, loop reversal

  strides in the wrong way, or negative strides

- Data reshaping required

  the smallest stride does not correspond to the data length

- Versioning required

  control or indirection - traces depend on data

4 - Code Transformations Hints

Hints for code transformations required for SIMDization are based on dependence graph, strides and control flow graph.

- Data alignment

  first address offset is not a multiple of the vector
- Rescheduling

  dependences of distance 1 on scalars, with no cycle
- Loop transformations, loop reversal

  strides in the wrong way, or negative strides
- Data reshaping required

  the smallest stride does not correspond to the data length
- Versioning required

  control or indirection - traces depend on data
- Idiom Recognition

  simple pattern matching on dependence graph (dot product, mem copy)

4 - Code Transformations Hints : Example

ICC 13.0.1 Output:

```
tsc.c(1420): (col. 4) remark: loop was not vectorized:
    existence of vector dependence.
```

GCC 4.6.3 Output:

```
tsc.c:1419: note: not vectorized: complicated access pattern.
tsc.c:1420: note: not vectorized, possible dependence between
    data-refs bb[D.18342_9][i_45] and bb[j_46][i_45]
```

s126:

```
int k = 1;
for(int i = 0; i < LEN2; i++){
  for(int j = 1; j < LEN2; j++){
    bb[j][i] = bb[j-1][i] +
        array[k-1] * cc[j][i];
    ++k;
  }
  ++k;
}
```

4 - Code Transformations Hints : Example

ICC 13.0.1 Output:

```
tsc.c(1420): (col. 4) remark: loop was not vectorized:
    existence of vector dependence.
```

GCC 4.6.3 Output:

```
tsc.c:1419: note: not vectorized: complicated access pattern.
tsc.c:1420: note: not vectorized, possible dependence between
    data-refs bb[D.18342_9][i_45] and bb[j_46][i_45]
```

s126:

```
int k = 1;
for(int i = 0; i < LEN2; i++){
  for(int j = 1; j < LEN2; j++){
    bb[j][i] = bb[j-1][i] +
        array[k-1] * cc[j][i];
    ++k;
  }
  ++k;
}
```

Maqao Output:

```
Loop at line 1420 of tsc.c, function s126:
vectorizable with reduction (add)
uncontiguous data (stride: 1024):
    need of interchange or transpose
code template:
- load (i, i+1, i+2, i+3)          line 1421
- load (j, j+256, j+512, j+768)
  and mul                          line 1421
- add                              line 1421
- store (j, j+256, j+512, j+768)   line 1421
```

5 - Evaluation on TSVC Benchmark

MAQAO vs GCC 4.6.3 and Intel ICC 13.0.1 compilers, on Intel Core i5-2540M:

Tool	**Maqao**	**GCC**	**ICC**
Detected vectorizable cases	123	46	104
Corresponding MAQAO hint			
- Reduction	30	15	24
- Idiom	8	3	7
- Data alignment issue	11	4	4
- Code restructuration	53	6	39
- Loop interchange or data transpose	9	4	7
- Rescheduling	9	1	1
- Control	23	0	17

Related Works

- Auto-vectorizing compilers for C code (GCC, ICC, IBM, ...) or higher level (Scout)
- Hybrid compile-time/run-time approach (profile guided optimization)
  - E. Park, L.N.P., Cavazos, J., Cohen, A., Sadayappan, P.: *Predictive modeling in a polyhedral optimization space.* Conf. on Code Generation and Optimization (2011)
  - Nuzman, D., Dyshel, S., Rohou, E., Rosen, I., Williams, K., Yuste, D., Cohen, A., Zaks, A.: *Vapor SIMD: Auto-vectorize once, run everywhere.* Conf. on Code Generation and Optimization (2011)
  - Tournavitis, G., Wang, Z., Franke, B., OBoyle, M.F.: *Towards a holistic approach to auto-parallelization: integrating profile-driven parallelism detection and machine-learning based mapping.* Conf. on Programming Language Design and Implementation (2009)
- Approach similar to us, but less information in input (no compressed trace, no dependence graph)
  Holewinski, J., Ramamurthi, R., Ravishankar, M., Fauzia, N., Pouchet, L.N., Rountev, A., Sadayappan, P.: *Dynamic trace-based analysis of vectorization potential of applications.* Conf. on Programming Language Design and Implementation (2012)

Conclusion

- Providing hints about transformations necessary for SIMDization

  combining static (for registers) and dynamic (for memory) dependence analysis

- Promising results on TSVC benchmark

- Plan to expand our work:
  - going further on the analysis (loop distribution, reroll)
  - implementing for various architectures (currently: ARM)
  - predicting performance gain