Vector Seeker
Finding Vector Potential

G. Carl Evans, David Padua
University of Illinois
With
Seth Abraham, Bob Kuhn
Intel
Outline

1. Introduction
   1. Motivation
   2. Current Compilers

2. Vector Seeker
   1. Ideas
   2. Tool
Motivation

• Vector extensions are relatively inexpensive and popular
• Compilers perform unevenly.
• Current hardware implementations have limitations.
• Manual vectorization is not portable and profitability is hard to determine ahead of time.
Compilers have limitations

# Media Bench II Applications

<table>
<thead>
<tr>
<th>Appl</th>
<th>XLC</th>
<th>ICC</th>
<th>GCC</th>
<th>XLC</th>
<th>ICC</th>
<th>GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Automatic</td>
<td>Manual</td>
<td></td>
</tr>
<tr>
<td>JPEG Enc</td>
<td>-</td>
<td>1.33</td>
<td>-</td>
<td>1.39</td>
<td>2.13</td>
<td>1.57</td>
</tr>
<tr>
<td>JEPG Dec</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.14</td>
<td>1.13</td>
</tr>
<tr>
<td>H263 Enc</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.25</td>
<td>2.28</td>
<td>2.06</td>
</tr>
<tr>
<td>H263 Dec</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.31</td>
<td>1.45</td>
<td>-</td>
</tr>
<tr>
<td>MPEG2 Enc</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.06</td>
<td>1.96</td>
<td>2.43</td>
</tr>
<tr>
<td>MPEG2 Dec</td>
<td>-</td>
<td>-</td>
<td>1.15</td>
<td>1.37</td>
<td>1.45</td>
<td>1.55</td>
</tr>
<tr>
<td>MPEG4 Enc</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.44</td>
<td>1.81</td>
<td>1.74</td>
</tr>
<tr>
<td>MPEG4 Dec</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.12</td>
<td>-</td>
<td>1.18</td>
</tr>
</tbody>
</table>

Table shows whole program speedups measured against unvectorized application.

Dependence Analysis

• Dependence analysis is the foundation.
• It computes relations between statement instances.
• These relations can be used to transform programs:
  + for locality (tiling),
  + parallelism (vectorization, parallelization),
  + communication (message aggregation),
  + reliability (automatic checkpoints),
  + power …
Example of use of dependence

Consider the loop

```c
for (i=1; i<n; i++) {
    for (j=1; j<n; j++) {
        a[i][j]=a[i][j-1]+a[i-1][j];
    }
}
```
Example of use of dependence

Compute dependences (part 1)

for (i=1; i<n; i++) {
    for (j=1; j<n; j++) {
        a[i][j]=a[i] [j-1]+a[i-1][j];
    }
}

j=1  a[1][1] = a[1][0] + a[0][1]

j=2  a[1][2] = a[1][1] + a[0][2]

j=3  a[1][3] = a[1][2] + a[0][3]

j=4  a[1][4] = a[1][3] + a[0][4]

i=2

a[2][1] = a[2][0] + a[1][1]

a[2][2] = a[2][1] + a[1][2]

a[2][3] = a[2][2] + a[1][3]

Example of use of dependence

Compute dependences (part 2)

for (i=1; i<n; i++) {
  for (j=1; j<n; j++) {
    a[i][j] = a[i][j-1] + a[i-1][j];
  }
}

\[
\begin{array}{c}
  j=1 \\
  a[1][1] = a[1][0] + a[0][1] \\
  a[2][1] = a[2][0] + a[1][1]
\end{array}
\]

\[
\begin{array}{c}
  j=2 \\
  a[1][2] = a[1][1] + a[0][2] \\
  a[2][2] = a[2][1] + a[1][2]
\end{array}
\]

\[
\begin{array}{c}
  j=3 \\
  a[1][3] = a[1][2] + a[0][3] \\
  a[2][3] = a[2][2] + a[1][3]
\end{array}
\]

\[
\begin{array}{c}
  j=4 \\
  a[1][4] = a[1][3] + a[0][4] \\
\end{array}
\]
Example of use of dependence

```cpp
for (i=1; i<n; i++) {
    for (j=1; j<n; j++) {
        a[i][j]=a[i][j-1]+a[i-1][j];
    }
}
```

1 2 3 4 …

1 2 3 4 1,1

or
Example of use of dependence.

- Find parallelism

```c
for (i=1; i<n; i++) {
  for (j=1; j<n; j++) {
    a[i][j] = a[i][j-1] + a[i-1][j];
  }
}
```
Example of use of dependence

• Transform the code

```c
for (i=1; i<n; i++) {
    for (j=1; j<n; j++) {
        a[i][j]=a[i][j-1]+a[i-1][j];
    }
}
```

```c
for k=4; k<2*n; k++) forall (i=max(2,k-n):min(n,k-2)) a[i][k-i]=...
```
Goals

• Our goal is to develop tools to help compiler designers and programmers to deal with vectorization.

• These tools would help answer questions such as:
  + Compiler writers
    ▪ What vectorization opportunities are missed by the compiler?
  + Programmers
    ▪ Where are there potential opportunities for manual transformations?
  + Researchers:
    ▪ What opportunity is there for vector parallelism, or how much headroom is there?
      – Across the board
      – For certain classes of applications
Vectorization

• Dependence is the core
• Alias analysis
• Loop Transformations
  ✦ Splitting, Fusion, Peeling, Interchange
• Memory Layout Transformations
  ✦ Stride
• Pattern Matching
  ✦ Reductions
Idea - Tracing

Inspired by


• Kumar used tracing to find maximum parallelism
• Trace the sequential program and consider an instruction executed as soon possible on an ideal parallel machine
• Report
  - At each virtual time
    - Number of instructions executed
Kumar’s Ideal Machine

• Execute one instruction in unit time
• Execute each instruction as soon as all the following
  ✤ conditional branches in the sequential program proceeding the instruction have been determined
  ✤ all inputs to the instruction are available
• Complete knowledge of all data flow
• Output values are available for lifetime of program
Anti and Output Dependence

• Anti-dependence
  1. \( B = 3 \)
  2. \( A = B + 1 \)
  3. \( B = 7 \)

1. \( B = 3 \)
N. \( B^2 = B \)
2. \( A = B^2 + 1 \)
3. \( B = 7 \)

• Output dependence
  1. \( B = 3 \)
  2. \( A = B + 1 \)
  3. \( B = 7 \)

1. \( B^2 = 3 \)
2. \( A = B^2 + 1 \)
3. \( B = 7 \)

• Both removed by having all output values for lifetime of program
for (i = 0; i < N; i++)
    A[i] = B[i] + C[i];
for(i = 0; i < N; i++)
    A[i] = B[i] + C[i];

Simple Loop Dynamic Dependence Graph

Results

1

1

2

LD B[0]  LD C[0]

2


3

B[0] + C[0]

3


4

ST A[0]

4

ST A[1]

5

Center for Exascale Simulation of Plasma-Coupled Combustion
Our Idea - Tracing

• Use tracing to find maximum vector parallelism
• Trace the sequential program and consider an instruction executed as soon possible on an ideal vector machine

• Report
  + At each instruction location
    - At each virtual time
      – Number of times the instruction is executed with a dependence on a vector location
        » Why? What is a vector location? More on this later
Vector Seeker’s Ideal Machine

• Execute one vector instruction in unit time with all instructions from sequential program with the following conditions
  ➕ All input values have been computed
  ➕ All come from the same source instruction
  ➕ All have a dependence chain to a vector location

• Execute all serial instructions before any vector instructions
  ➕ Have no dependence chain to a vector location

• Complete knowledge of all data flow

• Output values are available for all time
for(i = 0; i < N; i++)
    A[i] = B[i] + C[i];
for (i = 0; i < N; i++)
    A[i] = B[i] + C[i];
for(i = 0; i < N; i++)
    A[i] = B[i] + C[i];
for(i = 0; i < N; i++)
    A[i] = B[i] + C[i];

Results
What is a Vector Location?

We need to determine what memory to treat as vector memory and what to prune.

- **Automatic:** Treat all heap allocated memory as vector locations

- **Manual:** Instrument code using
  - `void _tracer_array_memory(void *start, size_t length);`
  - `void _tracer_array_memory_clear(void *start);`
  - Used in the manual testing to find more vector parallelism
Vector Seeker Implementation

• Vector Seeker is built as a tool using Intel Pin
• Instrument the instructions in the sequential execution.
• Report the size of vectors found for every instruction operating on vector locations
Vector Seeker Data Structures

• Shadow Memory (SM[A])
  + Map from memory addresses and registers to depths in the trace
  + Initialized as follows all memory and registers initialized to ⊥ unless it is a vector location then initialized to 1

• Result Vector (RV[I][D])
  + Map indexed first by instruction line then by depth to number of dynamic executions
  + This stores how many times a particular instruction can be executed in parallel at a particular depth.
Greedy Instrument

\[ D \leftarrow \max(SM[\text{operand 1}], SM[\text{operand 2}], \ldots) \]

\[ I \leftarrow \text{instruction} \]

if \( I \) is a vector allocation then

for all addresses \( A \) in allocation do \( SM[A] = D + 1 \)

else if \( I \) is a vector deallocation then

for all addresses \( A \) in vector deallocation do \( SM[A] = \bot \)

else if \( I \) = simple load or store with address \( A \) then

\[ SM[A] \leftarrow D \]

else if \( D \neq \bot \) then

\[ SM[\text{destination address}] \leftarrow D + 1 \]

\[ RV[I][D + 1] \leftarrow RV[I][D + 1] + 1 \]

else

\[ SM[\text{destination address}] \leftarrow \bot \text{ or 1 if vector location} \]
Example Vector Seeker Tool Run

Code snippet from microtest.cpp
...
20   for(int i = 0; i < 5; i++)
21   {
23   }
24   for(int i = 0; i < 5; i++)
25   {
26     tmp += A[i]*B[i];
27   }
28   ...

Tool Run – Summary Version
#start instruction log
V:microtest.cpp,22:5
P:microtest.cpp,26:5
#end instruction log

Tool Run - Standard
#start instruction log
microtest.cpp,22
 0x400923:SSE:mulsd xmm0, xmm1
<2,5>
microtest.cpp,26
 0x400965:SSE:mulsd xmm0, xmm1
<3,5>
 0x40096e:SSE:addsd xmm0, xmm1
<4,1><5,1><6,1><7,1><8,1>
Example Vector Seeker Tool Run

Code snippet from microtest.cpp

```cpp
... 
20   for(int i = 0; i < 5; i++)
21   {
23   }
24   for(int i = 0; i < 5; i++)
25   {
26     tmp += A[i]*B[i];
27   }
28   ...
```

Tool Run – Summary Version
#start instruction log
V: microtest.cpp, 22: 5
P: microtest.cpp, 26: 5
#end instruction log

Tool Run - Standard
#start instruction log
microtest.cpp, 22
0x400923: SSE: mulsd xmm0, xmm1
<5>
microtest.cpp, 26
0x400965: SSE: mulsd xmm0, xmm1
<5>
0x40096e: SSE: addsd xmm0, xmm1
<1><1><1><1><1>

Source File and Line Number
Example Vector Seeker Tool Run

Code snippet from microtest.cpp

```cpp
... 20   for(int i = 0; i < 5; i++)
21   {
23   } 24   for(int i = 0; i < 5; i++)
25   {
26     tmp += A[i]*B[i];
27   }
28   ...
```

Tool Run – Summary Version

```
#start instruction log
V: microtest.cpp, 22: 5
P: microtest.cpp, 26: 5
#end instruction log
```

Tool Run - Standard

```
#start instruction log
microtest.cpp, 22
0x400923: SSE: mulsd xmm0, xmm1
<5>
microtest.cpp, 26
0x400965: SSE: mulsd xmm0, xmm1
<5>
0x40096e: SSE: addsd xmm0, xmm1
<1><1><1><1><1>
```

Vector Size

Source File and Line Number
Example Vector Seeker Tool Run

**Code snippet from microtest.cpp**

```cpp
... 20   for(int i = 0; i < 5; i++)
21   {
23   }
24   for(int i = 0; i < 5; i++)
25   {
26     tmp += A[i]*B[i];
27   }
28   ...
```

**Full/Partial Vector**

**Tool Run – Summary Version**

```txt
#start instruction log
V:microtest.cpp,22:5
P:microtest.cpp,26:5
#end instruction log
```

**Source File and Line Number**

**Tool Run - Standard**

```txt
#start instruction log
microtest.cpp,22
0x400923:SSE:mulsd xmm0, xmm1 <5>
microtest.cpp,26
0x400965:SSE:mulsd xmm0, xmm1 <5>
0x40096e:SSE:addsd xmm0, xmm1 <1><1><1><1><1>
```

**Vector Size**
Example Vector Seeker Tool Run

Code snippet from microtest.cpp

```cpp
... 20   for(int i = 0; i < 5; i++)
21   {
23   }
24   for(int i = 0; i < 5; i++)
25   {
26     tmp += A[i]*B[i];
27   }
28   ...
...```

Tool Run – Summary Version

```plaintext
#start instruction log
V:microtest.cpp,22:5
P:microtest.cpp,26:5
#end instruction log
```

Tool Run – Standard

```plaintext
#start instruction log
microtest.cpp,22
0x400923:SSE:mulsd xmm0, xmm1 <5>
microtest.cpp,26
0x400965:SSE:mulsd xmm0, xmm1 <5>
0x40096e:SSE:addsd xmm0, xmm1 <1><1><1><1><1>
```

Machine Instruction

Full/Partial Vector

Source File and Line Number
Vector Seeker and Reductions

- Reductions have true dependencies and thus are not found directly by Vector Seeker.
- In most cases they are easily inferred by a human from the pattern as seen in microtest.cpp.
- Automatic detection may be possible if needed.
for(i = 0; i < N; i++)
    sum += B[i];

Simple Reduction Pruned Graph

1
   LD B[0]
   sum + B[0]

2
   LD B[1]
   sum + B[1]

3
   LD B[2]
   sum + B[2]

...
Questions?