Dynamic SIMD Scheduling

Florian Wende

SC15 MIC Tuning BoF
November 18th, 2015
Dynamic Work Assignment: The Idea

Irregular SIMD execution

- Caused by branching: control flow varies across SIMD lanes
- SIMD lanes run out of work even though there is still work to do (next iteration)

```java
for (i=1; i<N; i++) {
    ... 
    while (pred) {
        <work>
        pred=...
    }
    ...
}
```

Static SIMD

Scalar loop execution
Dynamic Work Assignment: The Idea

Irregular SIMD execution
- Caused by branching: control flow varies across SIMD lanes
- SIMD lanes run out of work even though there is still work to do (next iteration)
- Can we pre-schedule successive iterations on idle SIMD lanes (dynamically)?

Scalar loop execution

```
for (i=1; i<N; i++) {
  ...
  while (pred) {
    <work>
    pred=...
  }
  ...
}
```

Dynamic SIMD vs. Static SIMD

- **Dynamic SIMD**
  - Even partitioning of all work + pre-scheduling within partitions
  - Greedy schedule (not shown here)

- **Scalar loop execution**

```
for (i=1; i<N; i++) {
  ...
  while (pred) {
    <work>
    pred=...
  }
  ...
}
```
Dynamic Work Assignment: Implementation

Dynamic SIMD

\[
\text{lane\_alife\_any}=\text{true}, \quad \text{acquire\_work\_any}=\text{true}
\]

\[
\text{for (ii}=0; \text{ ii}<\text{VL}; \text{ ii}++) \quad /\text{ VL: Vector Length}
\]

\[
\text{lane\_alife[ii]}=\text{true}, \quad \text{lane\_acquire\_work[ii]}=\text{true}, \quad i[ii]=ii
\]

\[
\text{while (lane\_alife\_any)}
\]

\[
\text{lane\_alife\_any}=\text{false}
\]

\[
\text{if (acquire\_work\_any)}
\]

\[
\text{Acquire work}
\]

\[
\text{for (ii}=0; \text{ ii}<\text{VL}; \text{ ii}++)
\]

\[
\text{if (lane\_alife[ii])}
\]

\[
\text{Do work}
\]

wende@zib.de
Dynamic Work Assignment: Implementation

Dynamic SIMD

\[
\text{lane\_alife\_any=true, acquire\_work\_any=true}
\]

for (ii=0; ii<VL; ii++) // VL: Vector Length

\[
\text{lane\_alife[ii]=true, lane\_acquire\_work[ii]=true, i[ii]=ii}
\]

while (lane\_alife\_any)

\[
\text{lane\_alife\_any=false}
\]

if (acquire\_work\_any)

\[
\text{acquire\_work\_any=false}
\]

for (ii=0; ii<VL; ii++)

\[
\text{if (lane\_acquire\_work[ii])}
\]

\[
\text{lane\_acquire\_work[ii]=false}
\]

\[
\text{if (i[ii] < N) load\_data(i[ii])}
\]

\[
\text{else lane\_alife[ii]=false}
\]

\[
\text{lane\_alife\_any=reduce\_or(lane\_alife[ii])}
\]

for (ii=0; ii<VL; ii++)

\[
\text{if (lane\_alife[ii])}
\]

Do work
Dynamic Work Assignment: Implementation

Dynamic SIMD

```c
lane_alife_any=true, acquire_work_any=true
for (ii=0; ii<VL; ii++) // VL: Vector Length
   lane_alife[ii]=true, lane_acquire_work[ii]=true, i[ii]=ii

while (lane_alife_any)
   lane_alife_any=false
   if (acquire_work_any)
      acquire_work_any=false
      for (ii=0; ii<VL; ii++)
         if (lane_acquire_work[ii])
            lane_acquire_work[ii]=false
            if (i[ii] < N) load_data(i[ii])
            else lane_alife[ii]=false
            lane_alife_any=reduce_or(lane_alife[ii])

   for (ii=0; ii<VL; ii++)
      if (lane_alife[ii])
         <work(i[ii])>
         pred=...
         if (!pred) store_data(i[ii]), i[ii]+=VL, lane_acquire_work[ii]=true
         acquire_work_any=reduce_or(lane_acquire_work[ii])
```

Static SIMD

Dynamic SIMD

wende@zib.de

Dynamic SIMD Scheduling

SC15 IXPUG MIC Tuning BoF
The kernel (synthetic)

Input:
\[ x_1[] \text{ uniform over } [0.0, 2.0] \]
\[ x_2[] \text{ uniform over } [0.0, 1.0] \]

Output:
\[ y[] \]

#define \text{N} \ (8 \times 1024 \times 1024)
#define \text{D} \ (50)

\textbf{for} (i=0; i<\text{N}; i++) \\
\quad \text{int } k=0 \\
\quad \text{int } k_{max}=(\text{int})(\text{D} \times x_2[i]) \\
\quad \text{double } temp_y=0.0 \\
\quad \text{double } temp_x1=x_1[i] \\
\quad \textbf{while} (k < k_{max}) \\
\quad \quad temp_y=\sqrt{temp_x1 + temp_y} \\
\quad \quad \text{if } (temp_y > 1.0) \text{ temp}_y=\log(temp_y) \\
\quad \quad k++ \\
\quad y[i]=temp_y
Platforms / Software
- Xeon E5-2680v3 @ 1.9GHz (AVX base frequency)
- Xeon Phi 5120D
- Intel C++ compiler 16.0, explicit vectorization with OpenMP 4.0

Runtimes for the loop execution in seconds (initial performance)

<table>
<thead>
<tr>
<th></th>
<th>Haswell</th>
<th>Xeon Phi</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D=50, N=8<em>1024</em>1024, x1[]$ and $x2[]$ at random $VL=4$</td>
<td>6.75 s</td>
<td>36.0 s</td>
</tr>
<tr>
<td>SIMD Static, Auto-Vectorization</td>
<td>4.63 s</td>
<td>11.7 s</td>
</tr>
<tr>
<td>SIMD Static, Explicit Vectorization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD Dynamic$_{EvenPartitioning}$ Explicit Vect.</td>
<td>3.90 s</td>
<td>10.1 s</td>
</tr>
</tbody>
</table>
Dynamic Work Assignment: Optimizations

Gaining performance over initial implementation

- Local (private) copies in case of repeated memory accesses
- Replace masked call to \( \log() \) (maybe SVML in general) by unmasked call + conditional assignment

\[
\begin{align*}
\text{if (pred) } & y = \log(...) & \rightarrow & \text{double } \text{temp}_y = \log(...) \\
& \text{if (pred) } & y = \text{temp}_y
\end{align*}
\]

- \#pragma vector aligned + \#pragma omp simd to tell the compiler all memory accesses are aligned: OpenMP 4.0 aligned clause has some limitations
- \texttt{safelen(n)} clause with values \( n > \) native vector length
### Platforms / Software
- Xeon E5-2680v3 @ 1.9GHz (AVX base frequency)
- Xeon Phi 5120D
- Intel C++ compiler 16.0, explicit vectorization with OpenMP 4.0

### Runtimes for the loop execution in seconds (final performance)

<table>
<thead>
<tr>
<th></th>
<th>Haswell</th>
<th></th>
<th>Xeon Phi</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VL=4</td>
<td>VL=8</td>
<td>VL=8</td>
<td>VL=16</td>
</tr>
<tr>
<td>$D=50, N=8<em>1024</em>1024, x1[]$ and $x2[]$ at random</td>
<td>6.75 s</td>
<td>-</td>
<td>36.0 s</td>
<td>-</td>
</tr>
<tr>
<td>SIMD Static, Auto-Vectorization</td>
<td>4.63 s</td>
<td>-</td>
<td>11.7 s</td>
<td>-</td>
</tr>
<tr>
<td>SIMD Static, Explicit Vectorization</td>
<td>3.08 s</td>
<td>(26%)</td>
<td>2.68 s</td>
<td>(45%)</td>
</tr>
<tr>
<td>SIMD Dynamic $^{\text{EvenPartitioning}}$, Explicit Vect.</td>
<td>9.83 s</td>
<td>(+3%)</td>
<td>9.67 s</td>
<td>(+5%)</td>
</tr>
</tbody>
</table>

Fastest version on KNC with Intrinsics: ~8 s
Platforms / Software

- Xeon E5-2680v3 @ 1.9GHz (AVX base frequency)
- Xeon Phi 5120D
- Intel C++ compiler 16.0, explicit vectorization with OpenMP 4.0

### Runtimes for the loop execution in seconds (final performance)

<table>
<thead>
<tr>
<th></th>
<th>Haswell</th>
<th>Xeon Phi</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>D=50, N=8<em>1024</em>1024, x1[] and x2[] at random</strong></td>
<td><strong>VL=4</strong></td>
<td><strong>VL=8</strong></td>
</tr>
<tr>
<td>SIMD Static, Auto-Vectorization</td>
<td>6.75 s</td>
<td>-</td>
</tr>
<tr>
<td>SIMD Static, Explicit Vectorization</td>
<td>4.63 s</td>
<td><strong>11.7 s</strong></td>
</tr>
<tr>
<td>SIMD Dynamic_EvenPartitioning_ Explicit Vect.</td>
<td>3.08 s (+26%)</td>
<td><strong>2.68 s</strong> (+45%)</td>
</tr>
</tbody>
</table>

Fastest version on KNC with Intrinsics: ~8 s

wendes@zib.de
## Dynamic Work Assignment: Test Sample, Performance

### Platforms / Software

- Xeon E5-2680v3 @ 1.9GHz (AVX base frequency)
- Xeon Phi 5120D
- Intel C++ compiler 16.0, explicit vectorization with OpenMP 4.0

### Runtimes for the loop execution in seconds (final performance)

<table>
<thead>
<tr>
<th></th>
<th>Haswell (VL=4)</th>
<th>Haswell (VL=8)</th>
<th>Xeon Phi (VL=8)</th>
<th>Xeon Phi (VL=16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D=50, N=8<em>1024</em>1024, x1[]$ and $x2[]$ at random</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD Static, Auto-Vectorization</td>
<td>6.75 s</td>
<td>-</td>
<td>36.0 s</td>
<td>-</td>
</tr>
<tr>
<td>SIMD Static, Explicit Vectorization</td>
<td><strong>4.63 s</strong> (+26%)</td>
<td>-</td>
<td><strong>11.7 s</strong> (+45%)</td>
<td>-</td>
</tr>
<tr>
<td>SIMD Dynamic $^\text{EvenPartitioning}$, Explicit Vect.</td>
<td>3.08 s</td>
<td><strong>2.68 s</strong> (+45%)</td>
<td>9.83 s (+3%)</td>
<td><strong>9.67 s</strong> (+5%)</td>
</tr>
</tbody>
</table>

Fastest version on KNC with Intrinsics: ~8 s

---

wend@zib.de  Dynamic SIMD Scheduling  SC15 IXPUG MIC Tuning BoF
Dynamic SIMD scheduling can help gaining the overall application performance

- **Speedup over static SIMD scheduling:** 1.5x and more for our sample
- Can be implemented with high-level language constructs (also in Fortran)
- Kind of optimization not carried out by the compiler yet

Where to use?

- Iteration schemes with varying convergence criteria
- Dynamic work creation

The End
Backup
Dynamic Work Assignment: Vector Expand Operation

Dynamic SIMD (Greedy schedule)

```c
for (ii=0; ii<VL; ii++)
    if (lane_acquire_work[ii]) i[ii] = icurrent++
```

SIMD execution requires vector expand operation!

```
icurrent has value 8
vector_expand(i[], lane_acquire_work[], [8,9,10,11])
icurrent now has value 10
```

Vector expand (and compress) operations will be available with AVX512!
Currently, we mimic this instruction in software
Dynamic Work Assignment: Test Sample, Performance

Platforms / Software
- Xeon E5-2680v3 @ 1.9GHz (AVX base frequency)
- Xeon Phi 5120D
- Intel C++ compiler 16.0, explicit vectorization with OpenMP 4.0

## Runtimes for the loop execution in seconds (final performance)

<p>|</p>
<table>
<thead>
<tr>
<th>D=50, N=8<em>1024</em>1024, x1[] and x2[] at random</th>
<th>VL=4</th>
<th>VL=8</th>
<th>VL=8</th>
<th>VL=16</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMD Static, Auto-Vectorization</td>
<td>6.75 s</td>
<td>-</td>
<td>36.0 s</td>
<td>-</td>
</tr>
<tr>
<td>SIMD Static, Explicit Vectorization</td>
<td>4.63 s</td>
<td>-</td>
<td>11.7 s</td>
<td>-</td>
</tr>
<tr>
<td>SIMD Dynamic_EvenPartitioning_ Explicit Vect.</td>
<td>3.08 s (+26%)</td>
<td>2.68 s (+45%)</td>
<td>9.83 s (+3%)</td>
<td>9.67 s (+5%)</td>
</tr>
<tr>
<td>SIMD Dynamic_Greedy_ Explicit Vect.</td>
<td>3.44 s (+18%)</td>
<td>3.07 s (+32%)</td>
<td>10.1 s (+2%)</td>
<td>11.0 s (-7%)</td>
</tr>
<tr>
<td>SIMD Dynamic_EvenPartitioning_ Explicit Vect.</td>
<td>2.81 s (+38%)</td>
<td>-</td>
<td>7.97 s (+26%)</td>
<td>-</td>
</tr>
<tr>
<td>SIMD Dynamic_Greedy_ Explicit Vect.</td>
<td>2.93 s (+38%)</td>
<td>-</td>
<td>8.08 s (+27%)</td>
<td>-</td>
</tr>
</tbody>
</table>

wende@zib.de

Dynamic SIMD Scheduling

SC15 IXPUG MIC Tuning BoF