SIMD Vectorization for the WHT

Drexel University
Computer Science Dept.

Michael Andrews <mjand@drexel>
Jeremy Johnson <jjohnson@drexel.edu>
Motivation

- Wanted to add support for SIMD instructions
  - SSE / SSE2 codelet generation
  - Include in search mechanism
- WHT 1.8 package is difficult to modify and extend
- Decided to reorganize package to make it easier to modify and extend in the future
  - Example: Parallel, Fixed Point
Approach

- Reorganized package with an emphasis on modularization
  - Modeled after FFTW package layout
- Vectorized codelet generator
  - Right most codelets
    \[
    \begin{cases}
    n > 2v & (W_2 \otimes I_{(n/2)})(I_2 \otimes W_{(n/2)}) \\
    \text{otherwise} & W_{2v}
    \end{cases}
    \]
  - Interleaved codelets
    \[(I_k \otimes (W_n \otimes I_v))\]
Package Maintainace

• Updated to standard build system
  – GNU autoconf, automake and libtool
  – 900 % less make code (by word)
  – 160 % less configure code (by word)
• Replaced custom parser with bison parser
• Replaced PCL performance measurement with PAPI
  – Includes parameters to control sampling based on statistical significance
• Modularized adding new codelets
Adding a new codelet

- **Split / Small**
  - Write implementation to package interface
  - Add to small codelet registry
  - Add configure parameters
- **Unrolled**
  - Write code generator
  - Add to make_codelets script
Case Study – Interleaved (small)

1) Write codelet implementation, assuming standard interface.

```c
/* interleave.c */
Wht *
interleave_init(char *name, size_t n, int params[], size_t np)
{
    Wht *W;
    size_t k;

    k = params[0];

    /* “Derive” from small “base class” */
    W = small_init(name, n, params, np);
    W->accept = interleave_accept;
    W->nk = k;

    /* Configure parameter selected by user */
    if (k > (1 << WHT_MAX_INTERLEAVE))
        wht_error("not configured for codelets of size %zd interleaved by %zd", n, k);

    W->attr[interleave_by] = k;

    return W;
}
```
Case Study – Interleaved (small)

• 2) Trivially, add implementation file to build process

/* wht/Makefile.am */
-- SNIP --
libwht_la_SOURCES = direct.c interleave.c vector.c small.c split.c \
    utils.c wht.c scanner.l parser.y
-- SNIP --

• 3) Add new codelet implementation to registry

/* registry.h */

static const small_init_entry
smalls_registry[] = {
-- SNIP --
/* identifier, number of parameters, dispatcher */
    { "smallil", 1, (small_init_fp) &interleave_init},
-- SNIP --
}
Case Study – Interleaved (small)

• 4) Add new configure parameters

    /* configure.ac */

    -- SNIP --

    AC_ARG_ENABLE([max_interleave],
        [AC_HELP_STRING([--enable-interleave=K],
            [interleave 2^K loop iterations in codelet]],
        [max_interleave=$enableval],
        [max_interleave=$MAX_INTERLEAVE])

    if test $max_interleave -gt $MAX_INTERLEAVE_PKG; then
        AC_MSG_ERROR([--enable-interleave larger than package was built for])
    fi

    AC_DEFINE_UNQUOTED(WHT_MAX_INTERLEAVE,$max_interleave,
        [interleave 2^K loop iterations in codelet])

    -- SNIP --

• 5) Run autogen.sh to rebuild build toolchain
Case Study – Interleaved (unrolled)

1) Create codelet generator
   - Input as command line parameters
   - Output to codelet to stdout
   - Required codelet interface
     
     ```
     typedef void (*codelet)(Wht *W, long S, wht_value *x);
     void apply_small2_il2(Wht *W, long S, wht_value *x)
     ```

2) Add to codelet creation script make_codelets.pl

```-- SNIP --
for ($i=1;$i<=$small;$i++) {
  for ($j=1;$j<=$interleave;$j++) {
    $k = 2**$j;
    push(@codelets,($i, "smallil", [ $k ], "whtgen", "-n $i -i $k", "s$_i\_il\_$k.c");
  }
}
-- SNIP --
```
SIMD Codelets

- Vectorized codelets (right most leaf)

\[
  \begin{cases}
    N > 2v & (W_2 \otimes I_{N/2})(I_2 \otimes W_{N/2}) \\
    \text{otherwise} & W_{2v}
  \end{cases}
\]

- \text{W}_2v\text{ implementation is SIMD instruction dependent}

Let

- \( n \): transform size
- \( x \): input vector
- \( y \): output vector
- \( v \): SIMD vector size
- \( a \): initial offset for input
- \( s \): access stride for input
- \( b \): initial offset for output
- \( t \): access stride for output

\[
\text{wht}(n,x,a,s,y,b,t,v)
\]

- if \( n = 2 \)
  - return \( \text{wht}_2(x,a,s,y,b,t,v) \)
- else if \( n = 2^v \)
  - return \( \text{wht}_{2v}(n,x,a,s,y,b,t,v) \)
- else
  - \( z = y \)
  - \( m = n/2 \)
  - \( \text{wht}(m,x,a,s,z,b,t,v) \)
  - \( \text{wht}(m,x,a+(m/v),s,z,b+(m/v),t,v) \)
  - for \( i \) in 0 .. \((m/v)-1\) do
    - \( \text{wht}_2(z,a+i,(m/v)*s,y,b+i,(m/v)*t,v) \)
  - done
#include "wht.h"

void apply_small2_v2_a(Wht *W, long S, wht_value *x)
{
#if (1 <= WHT_MAX_UNROLL) && (2 == WHT_VECTOR_SIZE)
    wht_vector2 ta1;
    wht_vector2 ta2;
    wht_vector2 ta3;
    wht_vector2 ta4;
    wht_vector2 ta5;
    wht_vector2 ta6;
    wht_vector2 ta7;
    wht_vector2 ta8;
    wht_vector2 ta9;
    wht_vector2 ta10;
    vload2(ta1,x[0]);
    vload2(ta2,x[2]);
    vadd2(ta3,ta1,ta2);
    vsub2(ta4,ta1,ta2);
    vshuf2(ta5,ta3,ta4,0,0);
    vshuf2(ta6,ta3,ta4,1,1);
    vadd2(ta7,ta5,ta6);
    vsub2(ta8,ta5,ta6);
    vshuf2(ta9,ta7,ta8,0,0);
    vshuf2(ta10,ta7,ta8,1,1);
    vstore2(ta9,x[0]);
    vstore2(ta10,x[2]);
#else
    wht_error("initialization guards should prevent this message");
#endif
}
Performance Improvement

Optimal Plan determined by new Dynamic Programming implementation

Baseline Plan determined by WHT 1.8-- Cycle counts measured with PAPI

Improvement by a factor of 1.6 to 2.0
Performance Improvement

Optimal Plan determined by new Dynamic Programming implementation

Baseline Plan determined by WHT 1.8-- Cycle counts measured with PAPI

No significant improvement
Observations

- GCC never utilizes vector instructions without intrinsics macros
- ICC ONLY utilizes vector load / stores without intrinsics macros
  - Auto vectorization in ICC requires loops, we unroll loops
- Pentium IV prefers iterative algorithms
  - Increases time to compute Dynamic Programming
- Opteron prefers algorithms with 1 level recursion
Performance of Vectorized Codelets

Intel Pentium 4
3.2 GHz
16 Kb L1 Data Cache 8 Way
2048 Kb L2 Data Cache 8 Way
64 Byte Cache Line
gcc 3.4.6
papi 3.2.1

Performance increases by a factor of
1.5 to 2.0 after the size of codelet is
greater than $2^4$ on Pentium and
1.2 to 2.3 after size $2^5$ on Opteron

All optimal plans choosen by DP
utilize a right most vectorized codelet

AMD Opteron
1.8 GHz
64 Kb L1 Data Cache 2 Way
1024 Kb L2 Data Cache 8 Way
64 Byte Cache Line
gcc 3.4.4
papi 3.2.1
The split[n,4] plan was chosen to measure performance improvement due to interleaving. A codelet of size 4 is necessary to interleave by at most 16.

Though this suggests that some codelets perform better when interleaved, the DP algorithm rarely chooses them on this architecture.
Performance of Interleaved Codelets

split[n, 4] interleaved by k vs. split[n,4] interleaved by k AND vectorized by 2

Again, vectorization improves performance by around a factor of 1.5 to 2.0
Performance of Interleaved Codelets

Interleaving by 2 and 4 provides most substantial improvement, around a factor of 1.4 - 1.6.

The DP algorithm occasionally chooses codelets interleaved by 8 on this architecture, and ALWAYS chooses vectorized and interleaved codelets.
Performance of Interleaved Codelets

The split\([n, 4]\) plan was chosen to measure performance improvement due to interleaving. A codelet of size 4 is necessary to interleave by at most 16.

Though this suggests that NO codelets perform better when interleaved, the DP algorithm OFTEN chooses them on this architecture.

AMD Opteron
1.8 GHz
64 Kb L1 Data Cache 2 Way
1024 Kb L2 Data Cache 8 Way
64 Byte Cache Line
gcc 3.4.4
papi 3.2.1
Performance of Interleaved Codelets

split[n, 4] interleaved by k vs. split[n,4] interleaved by k AND vectorized by 2

Though this suggests that ALL codelets perform better when vectorized, the DP algorithm RARELY chooses them on this architecture.
Performance of Interleaved Codelets

split[n, 4] vs. split[n,4] interleaved by k AND vectorized by 2

No real performance improvement here reflects overall lack of performance improvement on this architecture.
## Appendix A: Plans chosen by DP

### Intel Pentium 4 3.2 GHz
- 16 Kb L1 Data Cache 8 Way
- 2048 Kb L2 Data Cache 8 Way
- 64 Byte Cache Line
- gcc 3.4.6
- papi 3.2.1

### WHT 1.8

<table>
<thead>
<tr>
<th>Size</th>
<th>Cycles</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>691</td>
<td>small[2]</td>
</tr>
<tr>
<td>3</td>
<td>859</td>
<td>small[3]</td>
</tr>
<tr>
<td>4</td>
<td>868</td>
<td>small[4]</td>
</tr>
<tr>
<td>5</td>
<td>1689</td>
<td>small[5]</td>
</tr>
<tr>
<td>6</td>
<td>3356</td>
<td>small[6]</td>
</tr>
<tr>
<td>7</td>
<td>5254</td>
<td>split[ smallil2[3], small[4] ]</td>
</tr>
<tr>
<td>8</td>
<td>13376</td>
<td>split[ smallil2[2], small[6] ]</td>
</tr>
<tr>
<td>9</td>
<td>26014</td>
<td>split[ smallil2[3], small[6] ]</td>
</tr>
<tr>
<td>10</td>
<td>61169</td>
<td>split[ small[4], small[6] ]</td>
</tr>
<tr>
<td>12</td>
<td>257189</td>
<td>split[ smallil2[3], split[ smallil2[3], small[6] ] ]</td>
</tr>
<tr>
<td>13</td>
<td>566345</td>
<td>split[ smallil8[4], split[ smallil2[3], small[6] ] ]</td>
</tr>
<tr>
<td>15</td>
<td>2885659</td>
<td>split[ smallil8[3], split[ smallil2[3], split[ smallil2[3], small[6] ] ] ]</td>
</tr>
</tbody>
</table>

### WHT 2.0 No Vector

<table>
<thead>
<tr>
<th>Size</th>
<th>Cycles</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>624</td>
<td>small[2]</td>
</tr>
<tr>
<td>3</td>
<td>703</td>
<td>small[3]</td>
</tr>
<tr>
<td>4</td>
<td>881</td>
<td>small[4]</td>
</tr>
<tr>
<td>5</td>
<td>1589</td>
<td>small[5]</td>
</tr>
<tr>
<td>6</td>
<td>3134</td>
<td>small[6]</td>
</tr>
<tr>
<td>7</td>
<td>5699</td>
<td>split[ small[4], small[3] ]</td>
</tr>
<tr>
<td>8</td>
<td>10474</td>
<td>split[ small[4], small[4] ]</td>
</tr>
<tr>
<td>9</td>
<td>25960</td>
<td>split[ small[4], small[3], small[2] ]</td>
</tr>
<tr>
<td>12</td>
<td>243855</td>
<td>split[ smallil8[4], small[4], small[4] ]</td>
</tr>
<tr>
<td>13</td>
<td>528930</td>
<td>split[ smallil4[5], small[4], small[4] ]</td>
</tr>
<tr>
<td>14</td>
<td>1176058</td>
<td>split[ smallil2[6], small[4], small[4] ]</td>
</tr>
<tr>
<td>15</td>
<td>2565392</td>
<td>split[ smallil8[4], smallil8[4], small[4], small[3] ]</td>
</tr>
</tbody>
</table>

### WHT 2.0 With Vector

<table>
<thead>
<tr>
<th>Size</th>
<th>Cycles</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>611</td>
<td>small[1]</td>
</tr>
<tr>
<td>2</td>
<td>616</td>
<td>smallv2[2]</td>
</tr>
<tr>
<td>3</td>
<td>656</td>
<td>smallv2[3]</td>
</tr>
<tr>
<td>4</td>
<td>700</td>
<td>smallv2[4]</td>
</tr>
<tr>
<td>5</td>
<td>912</td>
<td>smallv2[5]</td>
</tr>
<tr>
<td>6</td>
<td>1497</td>
<td>smallv2[6]</td>
</tr>
<tr>
<td>7</td>
<td>3039</td>
<td>smallv2[7]</td>
</tr>
<tr>
<td>8</td>
<td>8460</td>
<td>split[ small[4], smallv2[4] ]</td>
</tr>
<tr>
<td>9</td>
<td>18085</td>
<td>split[ smallilv4[3], small[3], smallv2[3] ]</td>
</tr>
<tr>
<td>10</td>
<td>39769</td>
<td>split[ smallilv2[4], smallil[3], smallv2[3] ]</td>
</tr>
<tr>
<td>11</td>
<td>88254</td>
<td>split[ smallilv2[4], smallv2[4], smallv2[3] ]</td>
</tr>
<tr>
<td>12</td>
<td>166465</td>
<td>split[ smallilv2[4], smallv2[4], smallv2[4] ]</td>
</tr>
<tr>
<td>13</td>
<td>361364</td>
<td>split[ smallilv8[5], small[4], smallv2[4] ]</td>
</tr>
<tr>
<td>14</td>
<td>768870</td>
<td>split[ smallilv4[6], small[4], smallv2[4] ]</td>
</tr>
<tr>
<td>15</td>
<td>1670406</td>
<td>split[ smallil2[7], small[4], smallv2[4] ]</td>
</tr>
</tbody>
</table>
### Appendix A: Plans chosen by DP

**Opteron**

**WHT 1.8**

<table>
<thead>
<tr>
<th>Size</th>
<th>Cycles</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>451</td>
<td>small[2]</td>
</tr>
<tr>
<td>3</td>
<td>471</td>
<td>small[3]</td>
</tr>
<tr>
<td>4</td>
<td>668</td>
<td>small[4]</td>
</tr>
<tr>
<td>5</td>
<td>1418</td>
<td>small[5]</td>
</tr>
<tr>
<td>6</td>
<td>3602</td>
<td>small[6]</td>
</tr>
<tr>
<td>7</td>
<td>6769</td>
<td>small[7]</td>
</tr>
<tr>
<td>8</td>
<td>5654</td>
<td>split[ small[3], small[5] ]</td>
</tr>
<tr>
<td>9</td>
<td>12728</td>
<td>split[ small[3], small[6] ]</td>
</tr>
<tr>
<td>10</td>
<td>24655</td>
<td>split[ small[4], small[6] ]</td>
</tr>
<tr>
<td>11</td>
<td>45982</td>
<td>split[ small[3], small[3], small[5] ]</td>
</tr>
<tr>
<td>12</td>
<td>103484</td>
<td>split[ small[3], split[ small[3], small[6] ] ]</td>
</tr>
<tr>
<td>13</td>
<td>230368</td>
<td>split[ small[4], small[6] ]</td>
</tr>
<tr>
<td>14</td>
<td>577190</td>
<td>split[ small[3], small[3], small[5] ]</td>
</tr>
<tr>
<td>15</td>
<td>1263494</td>
<td>split[ small[3], small[3], small[5] ]</td>
</tr>
</tbody>
</table>

**WHT 2.0 No Vector**

<table>
<thead>
<tr>
<th>Size</th>
<th>Cycles</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>431</td>
<td>small[2]</td>
</tr>
<tr>
<td>3</td>
<td>452</td>
<td>small[3]</td>
</tr>
<tr>
<td>4</td>
<td>661</td>
<td>small[4]</td>
</tr>
<tr>
<td>5</td>
<td>1122</td>
<td>split[ small[3], small[2] ]</td>
</tr>
<tr>
<td>6</td>
<td>1439</td>
<td>split[ small[3], small[3] ]</td>
</tr>
<tr>
<td>7</td>
<td>2665</td>
<td>split[ small[4], small[3] ]</td>
</tr>
<tr>
<td>8</td>
<td>4762</td>
<td>split[ small[4], small[4] ]</td>
</tr>
<tr>
<td>9</td>
<td>9661</td>
<td>split[ small[3], small[3], small[3] ]</td>
</tr>
<tr>
<td>10</td>
<td>20734</td>
<td>split[ small[4], small[3], small[3] ]</td>
</tr>
<tr>
<td>11</td>
<td>44527</td>
<td>split[ small[4], small[4], small[3] ]</td>
</tr>
<tr>
<td>12</td>
<td>91116</td>
<td>split[ small[3], small[3], small[3], small[3] ]</td>
</tr>
<tr>
<td>13</td>
<td>205586</td>
<td>split[ small[4], small[3], small[3], small[3] ]</td>
</tr>
<tr>
<td>14</td>
<td>629961</td>
<td>split[ small[4], small[4], small[3], small[3] ]</td>
</tr>
<tr>
<td>15</td>
<td>1341949</td>
<td>split[ small[4], small[4], small[3], small[3] ]</td>
</tr>
</tbody>
</table>

**WHT 2.0 With Vector**

<table>
<thead>
<tr>
<th>Size</th>
<th>Cycles</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>431</td>
<td>small[2]</td>
</tr>
<tr>
<td>3</td>
<td>452</td>
<td>small[3]</td>
</tr>
<tr>
<td>4</td>
<td>621</td>
<td>small[v2][4]</td>
</tr>
<tr>
<td>5</td>
<td>803</td>
<td>small[v2][5]</td>
</tr>
<tr>
<td>6</td>
<td>1439</td>
<td>split[ small[3], small[3] ]</td>
</tr>
<tr>
<td>7</td>
<td>2665</td>
<td>split[ small[4], small[3] ]</td>
</tr>
<tr>
<td>8</td>
<td>4762</td>
<td>split[ small[4], small[4] ]</td>
</tr>
<tr>
<td>9</td>
<td>9661</td>
<td>split[ small[3], small[3], small[3] ]</td>
</tr>
<tr>
<td>10</td>
<td>20734</td>
<td>split[ small[4], small[3], small[3] ]</td>
</tr>
<tr>
<td>11</td>
<td>44527</td>
<td>split[ small[4], small[4], small[3] ]</td>
</tr>
<tr>
<td>12</td>
<td>91116</td>
<td>split[ small[3], small[3], small[3], small[3] ]</td>
</tr>
<tr>
<td>13</td>
<td>205586</td>
<td>split[ small[4], small[3], small[3], small[3] ]</td>
</tr>
<tr>
<td>14</td>
<td>587544</td>
<td>split[ small[vil][16][4], small[4], small[3], small[3] ]</td>
</tr>
<tr>
<td>15</td>
<td>1250362</td>
<td>split[ small[vil][16][5], small[4], small[3], small[3] ]</td>
</tr>
</tbody>
</table>

**AMD Opteron**

1.8 GHz

- 64 Kb L1 Data Cache 2 Way
- 1024 Kb L2 Data Cache 8 Way
- 64 Byte Cache Line

gcc 3.4.4
papi 3.2.1