AUTOMATIC VECTORIZATION OF TREE TRAVERSALS

Youngjoon Jo, Michael Goldfarb and Milind Kulkarni

PACT, Edinburgh, U.K.
September 11th, 2013
Commodity processors and SIMD

- Commodity processors support SIMD (Single Instruction Multiple Data) instructions
- SIMD width getting wider
  - AVX is 256bit
  - Upcoming AVX-512 to be 512bit (2015)
- Using SIMD is an excellent way to improve performance
SIMD works great for regular loops

for (int i = 0; i < 4; i++) {
    c[i] = a[i] + b[i];
}
SIMD works great for regular loops

```c
for (int i = 0; i < 4; i++) {
    c[i] = a[i] + b[i];
}

__m128 vec_a = _mm_load_ps(a);
__m128 vec_b = _mm_load_ps(b);
__m128 vec_c = _mm_add_ps(vec_a, vec_b);
_mm_store_ps(c, vec_c);
```
But not so well on irregular codes

```c
void main() {
    Ray *rays[N] = // rays to trace
    Node *root = // root of tree
    for (int i = 0; i < N; i++) {
        recurse(rays[i], root);
    }
}

void recurse(Ray *r, Node *n) {
    if (truncate(r, n)) return;
    if (n->isLeaf()) {
        update(r, n);
    } else {
        recurse(r, n->left);
        recurse(r, n->right);
    }
}
```
void main() {
    Ray *rays[N] = // rays to trace
    Node *root = // root of tree
    for (int i = 0; i < N; i++) {
        recurse(rays[i], root);
    }
}

void recurse(Ray *r, Node *n) {
    if (truncate(r, n)) return;
    if (n->isLeaf()) {
        update(r, n);
    } else {
        recurse(r, n->left);
        recurse(r, n->right);
    }
}
Automatic vectorization desired

```c
void main() {
    Ray *rays[N] = // rays to trace
    Node *root = // root of tree
    for (int i = 0; i < N; i++) {
        recurse(rays[i], root);
    }
}

void recurse(Ray *r, Node *n) {
    if (truncate(r, n)) return;
    if (n->isLeaf()) {
        update(r, n);
    } else {
        recurse(r, n->left);
        recurse(r, n->right);
    }
}
```

Automatic vectorization techniques for irregular codes highly desired
Automatic vectorization desired

```c
void main() {
    Ray *rays[N] = // rays to trace
    Node *root = // root of tree
    for (int i = 0; i < N; i++) {
        recurse(rays[i], root);
    }
}

void recurse(Ray *r, Node *n) {
    if (truncate(r, n)) return;
    if (n->isLeaf()) {
        update(r, n);
    } else {
        recurse(r, n->left);
        recurse(r, n->right);
    }
}
```

Automatic vectorization techniques for tree traversals highly desired
Tree codes are important
Tree codes are important
Tree codes are important
Tree vectorization challenges

- Non trivial to find vectorizable computation

- Difficult to keep vectorizable computation together
Previous tree vectorization work

- Non trivial to find vectorizable computation
  - Manually transform code to packetize traversals
  - Process multiple traversals in packet simultaneously
  - Wald et. al. [Computer Graphics Forum 2001]

- Difficult to keep vectorizable computation together
Previous tree vectorization work

In situations like physical simulation, collision detection or raytracing in scenes, where rays bounce into multiple directions (spherical or bumpmapped surfaces), coherent ray packets break down very quickly to single rays or do not exist at all. In the above mentioned tasks, packet oriented SIMD computations is much less useful.

Havel and Herout
[IEEE Transactions on Visualization and Computer Graphics 2010]
Previous tree vectorization work

• Non trivial to find vectorizable computation
  • Manually transform code to packetize traversals
  • Process multiple points in packet simultaneously
  • Wald et. al. [Computer Graphics Forum 2001]

• Difficult to keep vectorizable computation together
  • Look to alternative sources of vectorization
  • Pixar [RT 2006]
    Dammertz et. al. [EGSR 2008]
  • Kim et. al. [SIGMOD 2010]
  • Chhugani et. al. [SC 2012]
Our previous tree locality work

• **Point blocking**
  Jo and Kulkarni [OOPSLA 2011]

• **Traversal splicing**
  Jo and Kulkarni [OOPSLA 2012]
Our solution

• Non trivial to find vectorizable computation
  • Manually transform code to packetize traversals
  • Automatically packetize traversals with point blocking and a novel layout transformation

• Difficult to keep vectorizable computation together
  • Look to alternative sources of vectorization
  • Exploit dynamic sorting of traversal splicing to dramatically enhance utilization
Contributions

• Show how tree traversal codes can be systematically transformed to
  • Expose SIMD opportunities
  • Enhance utilization

• Propose a novel layout transformation for efficient vectorization of tree codes

• Present a framework for automatically restructuring traversals and data layouts to enable vectorization
Contributions

• Show how tree traversal codes can be systematically transformed to expose SIMD opportunities.

• Enhance utilization.

• Propose a novel layout transformation for efficient vectorization of tree codes.

• Present a framework for automatically restructuring traversals and data layouts to enable vectorization.

Spoiler alert!

SIMTree can deliver speedups of up to 6.59, and 2.78 on average.
Outline

• Example & Abstract Model
• Point Blocking to Enable SIMD
• Traversal Splicing to Enhance Utilization
• Automatic Transformation
• Evaluation and Conclusion
void main() {
    Ray *rays[N] = // rays to trace
    Node *root = // root of tree
    for (int i = 0; i < N; i++) {
        recurse(rays[i], root);
    }
}

void recurse(Ray *r, Node *n) {
    if (truncate(r, n)) return;
    if (n->isLeaf()) {
        update(r, n);
    } else {
        recurse(r, n->left);
        recurse(r, n->right);
    }
}
void main() {
    Point *points[N] = // entities to traverse tree
    Node *root = // root of tree
    for (int i = 0; i < N; i++) {
        recurse(points[i], root);
    }
}

void recurse(Point *p, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}
void main() {
    Point *points[N] = // entities to traverse tree
    Node *root = // root of tree
    for (int i = 0; i < N; i++) {
        recurse(points[i], root);
    }
}

void recurse(Point *p, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}
void main() {
    Point *points[N] = // entities to traverse tree
    Node *root = // root of tree
    for (int i = 0; i < N; i++) {
        recurse(points[i], root);
    }
}

void recurse(Point *p, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}
void main() {
    Point *points[N] = // entities to traverse tree
    Node *root = // root of tree
    for (int i = 0; i < N; i++) {
        recurse(points[i], root);
    }
}

void recurse(Point *p, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}
void main() {
    Point *points[N] = // entities to traverse tree
    Node *root = // root of tree
    for (int i = 0; i < N; i++) {
        recurse(points[i], root);
    }
}

void recurse(Point *p, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}
An abstract model

```java
void main() {
    foreach (Point p : points) {
        foreach (Node n : p.oracleNodes()) {
            update(p, n);
        }
    }
}
```
Iteration space of traversal

```java
void main() {
  foreach (Point p : points) {
    foreach (Node n : p.oracleNodes()) {
      update(p, n);
    }
  }
}
```

Points → Nodes
Iteration space of traversal
Iteration space of traversal

Nodes

1 2 4 8 9 5 10 11 3 6 12 13 7 14 15

Points

A

B

C

D

E

F

G

H

1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
How to vectorize?

Nodes

1 2 4 8 9 5 10 11 3 6 12 13 7 14 15

Points

A

B

C

D

E

F

G

H
Outline

• Example & Abstract Model
• Point Blocking to Enable SIMD
• Traversal Splicing to Enhance Utilization
• Automatic Transformation
• Evaluation and Conclusion
Point blocking [OOPSLA 2011]
Point blocking [OOPSLA 2011]
void recurse(Point *p, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}
Point blocked code

```c
void recurse(Block *block, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}
```
void recurse(Block *block, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}

Function body
Point blocked code

```c
void recurse(Block *block, Node *n) {
    for (int i = 0; i = block->size; i++) {
        Point *p = block->p[i];
        if (truncate(p, n)) continue;
        if (n->isLeaf()) {
            update(p, n);
        } else {
            recurse(p, n->left);
            recurse(p, n->right);
        }
    }
}
```

Loop over points in block
Point blocked code

```c
void recurse(Block *block, Node *n) {
    for (int i = 0; i = block->size; i++) {
        Point *p = block->p[i];
        if (truncate(p, n)) continue;
        if (n->isLeaf()) {
            update(p, n);
        } else {
            recurse(p, n->left);
            recurse(p, n->right);
        }
    }
}
```
void recurse(Block *block, Node *n) {
    Block *nextBlock = // next level block
    for (int i = 0; i = block->size; i++) {
        Point *p = block->p[i];
        if (truncate(p, n)) continue;
        if (n->isLeaf()) {
            update(p, n);
        } else {
            nextBlock->add(p);
        }
    }
}
void recurse(Block *block, Node *n) {
    Block *nextBlock = // next level block
    for (int i = 0; i = block->size; i++) {
        Point *p = block->p[i];
        if (truncate(p, n)) continue;
        if (n->isLeaf()) {
            update(p, n);
        } else {
            nextBlock->add(p);
        }
    }
    if (nextBlock->size > 0) {
        recurse(nextBlock, n->left);
        recurse(nextBlock, n->right);
    }
}
Point blocking [OOPSLA 2011]
Point blocking [OOPSLA 2011]
Point blocking [OOPSLA 2011]

Nodes

Points

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

A
B
C
D
E
F
G
H
Analogous to packet SIMD
Analogous to packet SIMD

Breaks down when points diverge
Packet SIMD has poor utilization

Points: A, B, C, D, E, F, G, H

Nodes: 1, 2, 4, 8, 9, 5, 10, 11, 3, 6, 12, 13, 7, 14, 15

Full SIMD
Partial SIMD
Packet SIMD has poor utilization

![Diagram showing packet SIMD utilization]

- Points A through H
- Nodes 1 through 15
- Full SIMD vs. Partial SIMD

Youngjoon Jo
SIMD utilization

= Work in full SIMD / Total work
SIMD utilization

Work in full SIMD / Total work
= Circles in blue / Total circles
= 24 / 74 = 0.32
Use larger block size

Use block size larger than SIMD width and compact points!
Use larger block size

Points A B C D E F G H

Nodes 1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
Better utilization with larger block size

Nodes

1  2  4  8  9  5  10  11  3  6  12  13  7  14  15

Points

A  B  C  D  E  F  G  H

Full SIMD

Partial SIMD
Better utilization with larger block size

Circles in blue / Total circles = 64 / 74 = 0.86
SIMD utilization – Block size

![SIMD Utilization vs Block Size Graph](image_url)
Ideal utilization

Block size equal to total points yields **ideal** SIMD utilization
Use max block! Problem solved?
Large block has poor locality

![Graph showing SIMD utilization vs block size for different applications: Barnes-Hut, Point Correlation, Nearest Neighbor, Vantage Point, and Photon Mapping.]
Large block has poor locality

Need schedule with good utilization and good locality
Outline

• Example & Abstract Model
• Point Blocking to Enable SIMD
• Traversal Splicing to Enhance Utilization
• Automatic Transformation
• Evaluation and Conclusion
Traversal splicing [OOPSLA 2012]
Traversed splicing [OOPSLA 2012]

1. Designate splice nodes
Traversing splicing [OOPSLA 2012]

1. Designate splice nodes
2. Traverse up to splice node
Traversing splicing [OOPSLA 2012]

<table>
<thead>
<tr>
<th>Points</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>13</td>
<td>7</td>
<td>14</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Designate splice nodes
2. Traverse up to splice node
Traversing splicing [OOPSLA 2012]

1. Designate splice nodes
2. Traverse up to splice node
3. Resume at next node
Traversalsplicing [OOPSLA 2012]

1. Designate splice nodes
2. Traverse up to splice node
3. Resume at next node
Traversing splicing [OOPSLA 2012]

1. Designate splice nodes
2. Traverse up to splice node
3. Resume at next node
4. Repeat 2-3 until finished
Traversing splicing [OOPSLA 2012]

1. Designate splice nodes
2. Traverse up to splice node
3. Resume at next node
4. Repeat 2-3 until finished
Can change order of points

We can change the order of paused points, but how?

1. Designate splice nodes
2. Traverse up to splice node
3. Resume at next node
4. Repeat 2-3 until finished
Dynamic sorting

1. Designate splice nodes
2. Traverse up to splice node
3. Resume at next node
4. Repeat 2-3 until finished

Insight: points which reach same nodes are likely to have similar traversals in future

Dynamic sorting on traversal history
Dynamic sorting

1. Designate splice nodes
2. Traverse up to splice node
Dynamic sorting

1. Designate splice nodes
2. Traverse up to splice node
3. Reorder points at splice node
Dynamic sorting

1. Designate splice nodes
2. Traverse up to splice node
3. Reorder points at splice node
Dynamic sorting

1. Designate splice nodes
2. Traverse up to splice node
3. Reorder points at splice node
4. Resume at next node
Dynamic sorting

1. Designate splice nodes
2. Traverse up to splice node
3. Reorder points at splice node
4. Resume at next node
5. Repeat 2-4 until finished
Dynamic sorting

1. Designate splice nodes
2. Traverse up to splice node
3. Reorder points at splice node
4. Resume at next node
5. Repeat 2-4 until finished
Dynamic sorting enhances utilization

Full SIMD

Partial SIMD
Dynamic sorting enhances utilization

Circles in blue / Total circles = 48 / 74 = 0.65
SIMD utilization – splice depth

![Graph showing SIMD utilization vs. block size for different splice depths. The graph includes a legend for block sizes 2, 4, 6, 8, and 10, and a line for N/A. The x-axis represents block size with values ranging from 4 to 400,000, while the y-axis represents SIMD utilization with values ranging from 0 to 1. The graph indicates an increase in SIMD utilization as block size increases for all splice depths.](image-url)
SIMD utilization – splice depth

Block size: 512
Splice depth: 10

Nearest Neighbor
SIMD utilization
Dynamic sorting can automatically extract almost the maximum amount of SIMD utilization.
Outline

• Example & Abstract Model
• Point Blocking to Enable SIMD
• Traversal Splicing to Enhance Utilization
• Automatic Transformation
• Evaluation and Conclusion
Automatic transformation

- Point blocking
  Jo and Kulkarni [OOPSLA 2011]
- Traversal splicing
  Jo and Kulkarni [OOPSLA 2012]
Automatic transformation

- Our key addition for SIMD:
  Layout transformation from AoS (array of structures) to SoA (structure of arrays)
  - + Allows vector load/stores
  - + Packed data has better spatial locality
  - - More overhead in moving data
AoS to SoA layout

- Whole program AoS to SoA layout transformation difficult to automate with aliasing

- Limit scope to traversal code only
  - Copy in to SoA before traversal
  - Copy out to AoS after traversal

- Inter-procedural, flow-insensitive analysis
  - Determine which point fields should be SoA
  - Conservatively ensure correctness
AoS to SoA layout

```c
void recurse(Point *p, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}
```
AoS to SoA layout

```c
struct Point { float f1, f2, f3; }

void recurse(Point *p, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}
```
AoS to SoA layout

```c
struct Point { float f1, f2, f3; }
struct Node { Node *left, *right; Point *point; }

void recurse(Point *p, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}
```
AoS to SoA layout

```c
struct Point { float f1, f2, f3; }
struct Node { Node *left, *right; Point *point; }

void recurse(Point *p, Node *n) {
    if (truncate(p, n)) return;
    if (n->isLeaf()) {
        update(p, n);
    } else {
        recurse(p, n->left);
        recurse(p, n->right);
    }
}

bool truncate(Point *p, Node *n) {
    return p->f1 == n->point->f1;
}

void update(Point *p, Node *n) {
    p->f2 += n->point->f3;
}
```
Ensuring correctness

```c
struct Point { float f1, f2, f3; }
struct Node { Node *left, *right; Point *point; }

void recurse(Point *p, Node *n) {
  if (truncate(p, n)) return;
  if (n->isLeaf()) {
    update(p, n);
  } else {
    recurse(p, n->left);
    recurse(p, n->right);
  }
}

bool truncate(Point *p, Node *n) {
  return p->f1 == n->point->f1;
}

void update(Point *p, Node *n) {
  p->f2 += n->point->f3;
}
```
Ensuring correctness

```c
struct Point { float f1, f2, f3; }
struct Node { Node *left, *right; Point *point; }

bool truncate(Point *p, Node *n) {
    return p->f1 == n->point->f1;
}

void update(Point *p, Node *n) {
    p->f2 += n->point->f3;
}
```
Ensuring correctness

```c
struct Point { float f1, f2, f3; }
struct Node { Node *left, *right; Point *point; }

bool truncate(Point *p, Node *n) {
    return p->f1 == n->point->f1;
}

void update(Point *p, Node *n) {
    p->f2 += n->point->f3;
}
```
Ensuring correctness

```c
struct Point { float f1, f2, f3; }
struct Node { Node *left, *right; Point *point; }
```

<table>
<thead>
<tr>
<th>Point-access</th>
<th>Non-point-access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Write</td>
</tr>
<tr>
<td>f1</td>
<td>✓</td>
</tr>
<tr>
<td>f2</td>
<td></td>
</tr>
<tr>
<td>f3</td>
<td></td>
</tr>
</tbody>
</table>

```c
template bool truncate(Point *p, Node *n) {
    return p->f1 == n->point->f1;
}
```

```c
template void update(Point *p, Node *n) {
    p->f2 += n->point->f3;
}
```
Ensuring correctness

```c
struct Point { float f1, f2, f3; }
struct Node { Node *left, *right; Point *point; }

bool truncate(Point *p, Node *n) {
    return p->f1 == n->point->f1;
}

void update(Point *p, Node *n) {
    p->f2 += n->point->f3;
}
```
Ensuring correctness

struct Point { float f1, f2, f3; }
struct Node { Node *left, *right; Point *point; }

<table>
<thead>
<tr>
<th>Point-access</th>
<th>Non-point-access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Write</td>
</tr>
<tr>
<td>f1</td>
<td>✓</td>
</tr>
<tr>
<td>f2</td>
<td>✓</td>
</tr>
<tr>
<td>f3</td>
<td></td>
</tr>
</tbody>
</table>

bool truncate(Point *p, Node *n) {
    return p->f1 == n->point->f1;
}

void update(Point *p, Node *n) {
    p->f2 += n->point->f3;
}
Transforming SoA fields

```c
struct Point { float f1, f2, f3; }
struct Node { Node *left, *right; Point *point; }

bool truncate(Point *p, Node *n) {
    return p->f1 == n->point->f1;
}

void update(Point *p, Node *n) {
    p->f2 += n->point->f3;
}
```

<table>
<thead>
<tr>
<th>Point-access</th>
<th>Non-point-access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Write</td>
</tr>
<tr>
<td>f1</td>
<td>✓</td>
</tr>
<tr>
<td>f2</td>
<td>✓</td>
</tr>
<tr>
<td>f3</td>
<td></td>
</tr>
</tbody>
</table>
Transforming SoA fields

```c
struct Point { float f1, f2, f3; }
struct Node { Node *left, *right; Point *point; }

bool truncate(Block *block, int bi, Node *n) {
    return block->f1[bi] == n->point->f1;
}
void update(Block *block, int bi, Node *n) {
    block->f2[bi] += n->point->f3;
}
```
Correctness violation example

```c
struct Point { float f1, f2, f3; }
struct Node { Node *left, *right; Point *point; }
```

<table>
<thead>
<tr>
<th>Point-access</th>
<th>Non-point-access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Write</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
</tr>
<tr>
<td>f1 ✓</td>
<td>✓</td>
</tr>
<tr>
<td>f2 ✓</td>
<td>✓</td>
</tr>
<tr>
<td>f3</td>
<td></td>
</tr>
</tbody>
</table>

```c
bool truncate(Block *block, int bi, Node *n) {
    return block->f1[bi] == n->point->f1;
}
```

```c
void update(Block *block, int bi, Node *n) {
    block->f2[bi] += n->point->f2;
}
```
Ensuring correctness

```c
struct Point {
    float f1, f2, f3;
};
struct Node {
    Node *left, *right;
    Point *point;
};
```

<table>
<thead>
<tr>
<th></th>
<th>Point-access</th>
<th>Non-point-access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read</td>
<td>Write</td>
</tr>
<tr>
<td>f1</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>f2</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Sound analysis conservatively proves SoA transformation correct. Suffices to transform all of our benchmarks.
SIMTree

• Implementation of analysis and transformation in a source to source C++ compiler
• Based on ROSE compiler infrastructure
• Transforms code to apply point blocking, traversal splicing, and SoA layout
• Does not perform the vectorization itself
• https://engineering.purdue.edu/plcl/simtree/
Outline

• Example & Abstract Model
• Point Blocking to Enable SIMD
• Traversal Splicing to Enhance Utilization
• Automatic Transformation
• Evaluation and Conclusion
Evaluation

• Five benchmarks
  • Barnes-Hut, Point Correlation, Nearest Neighbor, Vantage Point, Photon Mapping

• Real and random inputs form 17 benchmark/inputs

• Two machines
  • Intel Xeon E5-4650
  • AMD Opteron 6282

• Automatic transformation with SIMTree

• Manual vectorization of transformed code with 4-way SIMD intrinsics for best performance
  • Auto vectorization of transformed code with icc gets 84% of best performance
Speedup on Xeon

Speedup

- Packet SIMD [CGF 2001]
- Block [OOPSLA 2011]
- Block+Splice [OOPSLA 2012]
- Block+SIMD
- Block+SIMD+Splice

Geometric means
Speedup on Xeon

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon</td>
<td>0.81</td>
<td>1.13</td>
<td>1.19</td>
<td>1.92</td>
<td>2.69</td>
</tr>
</tbody>
</table>
Dynamic sorting makes SIMD profitable

![Graph showing speedup for different configurations](image)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon</td>
<td>0.81</td>
<td>1.13</td>
<td>1.19</td>
<td>1.92</td>
<td>2.69</td>
</tr>
<tr>
<td>Opteron</td>
<td>0.83</td>
<td>1.15</td>
<td>1.27</td>
<td>1.78</td>
<td>2.86</td>
</tr>
</tbody>
</table>
Dynamic sorting makes SIMD profitable

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BH-Ran</td>
<td>0.81</td>
<td>1.13</td>
<td>1.19</td>
<td>1.92</td>
<td>2.69</td>
</tr>
<tr>
<td>BH-Plu</td>
<td>0.83</td>
<td>1.15</td>
<td>1.27</td>
<td>1.78</td>
<td>2.86</td>
</tr>
<tr>
<td>PC-Ran</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC-Cov</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC-Mni</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC-Geo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN-Ran</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN-Cov</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN-Mni</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN-Geo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP-Ran</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP-Cov</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP-Mni</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP-Geo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM-Dra</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM-Chr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM-Cab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geomean</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Instruction counts: Opteron

<table>
<thead>
<tr>
<th></th>
<th>Block</th>
<th>BlockSoA</th>
<th>BlockSoA + SIMD</th>
<th>Block + Splice</th>
<th>BlockSoA + Splice</th>
<th>BlockSoA + SIMD + Splice</th>
</tr>
</thead>
<tbody>
<tr>
<td>BH-Ran</td>
<td>1.27</td>
<td>1.62</td>
<td>1.20</td>
<td>1.08</td>
<td>1.38</td>
<td>0.64</td>
</tr>
<tr>
<td>BH-Plu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC-Ran</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC-Cov</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC-Mni</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC-Geo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN-Ran</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN-Cov</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN-Mni</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN-Geo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP-Ran</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP-Cov</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP-Mni</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP-Geo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM-Dra</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM-Chr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM-Cab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>geomean</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Cycles per instruction: Opteron

<table>
<thead>
<tr>
<th>Base</th>
<th>Block</th>
<th>BlockSoA</th>
<th>Block+Splice</th>
<th>BlockSoA+Splice</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.24</td>
<td>1.56</td>
<td>1.35</td>
<td>1.17</td>
<td>1.04</td>
</tr>
</tbody>
</table>
Conclusion

• Show how tree traversal codes can be systematically transformed to
  • Expose SIMD opportunities
  • Enhance utilization

• Propose a novel layout transformation for efficient vectorization of tree codes

• Present a framework for automatically restructuring traversals and data layouts to enable vectorization
Conclusion

• Show how tree traversal codes can be systematically transformed to
  • Expose SIMD opportunities
  • Enhance utilization

• Propose a novel layout transformation for efficient vectorization of tree codes

• Present a framework for automatically restructuring traversals and data layouts to enable vectorization

SIMTree is open source!
https://engineering.purdue.edu/plcl/simtree/
AUTOMATIC VECTORIZATION OF TREE TRAVERSALS

Youngjoon Jo, Michael Goldfarb and Milind Kulkarni

PACT, Edinburgh, U.K.
September 11th, 2013