Impact of AVX-512 Instructions on Graph Partitioning Problems

Md Maruf Hossain,
Erik Saule

University of North Carolina at Charlotte

{mhossa10,esaule}@uncc.edu

This work is supported by the National Science Foundation CCF-1652442
and was made possible by a computing allocation given by TACC through XSEDE.
Explore the Intel vector architectures (Cascade Lake and Skylake).
Vectorized parallel Louvain Method (PLM).
Apply vector operation on the parallel greedy graph coloring algorithm.
Compare with the state-of-the-art existing algorithm.
Show the quality of the algorithms are preserved.
Motivations

- Intel AVX-512 vectorization offers potential speedups.
- Vectorization provide proper utilization of the large 512 bits registers.

- Make sure of the higher usages of the memory bandwidth and floating-point-operations.
- Energy efficient.

scalar floating point operation

vectorized floating point operation
Louvain Method: Move phase

Greedy optimization method that extracts communities from large networks.

Modularity difference moving $u$ from community $C$ to community $D$:

$$\Delta mod(u, C \rightarrow D) = \frac{\omega(u, D/ \{u\})-\omega(u, C/ \{u\})}{\omega(E)} + \frac{(vol(C/ \{u\})-vol(D/ \{u\}))*vol(u)}{2*\omega(E)^2}$$
Louvain Method: Coarsening

Source: Blondel, Guillaume, Lambiotte, Lefebvre. 2008.
1: for \( v \in N(u) \) do
2:     if \( u \neq v \) then
3:         affinity[\( \zeta[v] \)] \leftarrow \text{affinity}[\zeta[v]] + \omega[u][v]
4:     if \( \zeta[v] \) not in neigh_comm then
5:         neigh_comm \leftarrow \zeta[v]
6:     end if
7: end if
8: end for
1: best ← 0
2: C ← \zeta[u]
3: for \(D \in \text{neigh\_comm}\) do
4:     if \(D \neq C\) then
5:         \[\Delta \text{mod}(u, C \rightarrow D) = \frac{\text{affinity}[D] - \text{affinity}[C]}{\omega(E)} + \frac{(\text{vol}(C/\{u\}) - \text{vol}(D/\{u\})) \times \text{vol}(u)}{2 \times \omega(E)^2}\]
6:     if \(\Delta \text{mod}(u, C \rightarrow D) > \text{best}\) then
7:         best ← \(\Delta \text{mod}(u, C \rightarrow D)\)
8:         possible\_next\_comm ← D
9:     end if
10: end if
11: end for
1: for $u \in V$ do
2:     $affinity[x] = 0, \forall x$
3: for $v \in N(u)$ do
4:     $affinity[\zeta[v]] + = w[u][v]$
5: end for
6: make move decision
7: end for
One Vertex Per Lane (OVPL)

1: for $i \leftarrow 1$ to $\text{min\_deg}($vertex\_block$)$ do
2:   for $u \in$ vertex\_block do
3:     $v \leftarrow N[u][i]$
4:     if $u \neq v$ then
5:       affinity$[u][\zeta[v]] \leftarrow$ affinity$[u][\zeta[v]] + \omega[u][v]$
6:     if $\zeta[v]$ not in neigh\_comm$[u]$ then
7:       neigh\_comm$[u] \leftarrow \zeta[v]$
8:   end if
9: end if
10: end for
11: end for
12: for $i \leftarrow \text{min\_deg}($vertex\_block$) + 1$ to $\text{max\_deg}($vertex\_block$)$ do
13:   for $u \in$ vertex\_block do
14:     if $\text{deg}(u) \geq i$ then
15:       $v \leftarrow N[u][i]$
16:     if $u \neq v$ then
17:       affinity$[u][\zeta[v]] \leftarrow$ affinity$[u][\zeta[v]] + \omega[u][v]$
18:     if $\zeta[v]$ not in neigh\_comm$[u]$ then
19:       neigh\_comm$[u] \leftarrow \zeta[v]$
20: end if
21: end if
22: end for
One Vertex Per Lane (OVPL)

Graph Coloring

OVPL Blocking

Physical Memory
If more than one neighbor belong to same community, the vectorized modularity calculation might lead to conflicts.

We propose two mechanisms to handle the conflict,

- Identify neighbors of different community with _mm512_conflict_epi32 from AVX–512CD.
- Compute the affinity of one community using _mm512_mask_reduce_add_ps from AVX–512F.
Reduce-Scatter using Conflict Detection

- **N** is the neighbors
- **C** is the communities of the **N**
- **M** is the mask (from Conflict Detection)
- **RN** is the remaining neighbors

```
const __m512i set0 = __mm512_set1_epi32(0x00000000);
/// Load at most 16 neighbor vertices.
__m512i N = __mm512_loadu_si512((__m512i *) &pnt_outEdges[i]);
/// Gather community of the neighbor vertices.
__m512i C = __mm512_mask_i32gather_epi32(set0, self_loop_mask, N, &zeta[0], 4);
/// Detect conflict of the community
__m512i C_conflict = __mm512_conflict_epi32(C);
/// Calculate mask **M** by comparing **C_conflict** with **set0**
const __mmask16 M = __mm512_mask_cmpeq_epi32_mask(self_loop_mask, C_conflict, set0);
```
- **N** is the neighbors
- **C** is the communities
- **M** is the mask (from compress)
- **RN** is the remaining neighbors
- **RC** is the remaining communities

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>28</td>
<td>16</td>
<td>93</td>
<td>44</td>
<td>11</td>
<td>72</td>
<td>50</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>M</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RN</td>
<td>93</td>
<td>44</td>
<td>11</td>
<td>50</td>
<td>23</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RC</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Quality of Louvain Method
Performance of Louvain Method

Performance on Cascade Lake (48 threads)  
Performance on Skylake (36 threads)
Performance of OVPL on Graphs with Regular Degrees
We investigate the impact of AVX-512 instructions of the Cascade Lake and Skylake. OVPL shows efficiency for graphs with balanced and high average degree. ONPL also shows good performance but it requires to handle reduce-scatter explicitly. In future, compiler extension could enable the techniques without intrinsic