Impact of AVX-512 Instructions on Graph Partitioning Problems

Md Maruf Hossain,
Erik Saule

University of North Carolina at Charlotte

{*mhossa10,esaule*}*@uncc.edu*

UNC CHARLOTTE

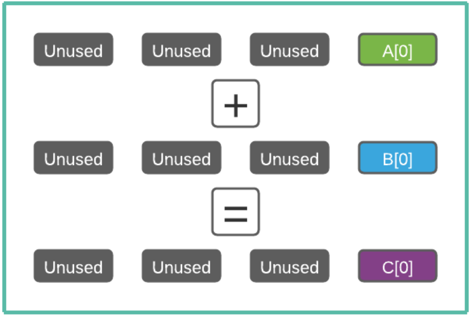
This work is supported by the National Science Foundation CCF-1652442
and was made possible by a computing allocation given by TACC through XSEDE.

Overview

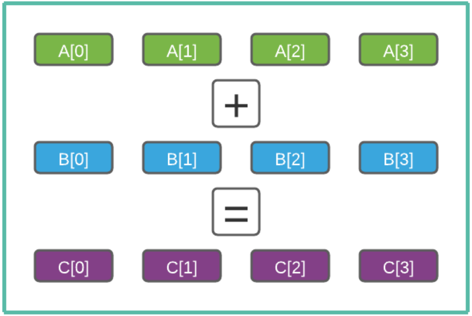
- Explore the Intel vector architectures(Cascade Lake and Skylake).
- Vectorized parallel Louvain Method(PLM).
- Apply vector operation on the parallel greedy graph coloring algorithm.
- Compare with the state-of-the-art existing algorithm.
- Show the quality of the algorithms are preserved.

Motivations

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



scalar floating point operation



vectorized floating point operation

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

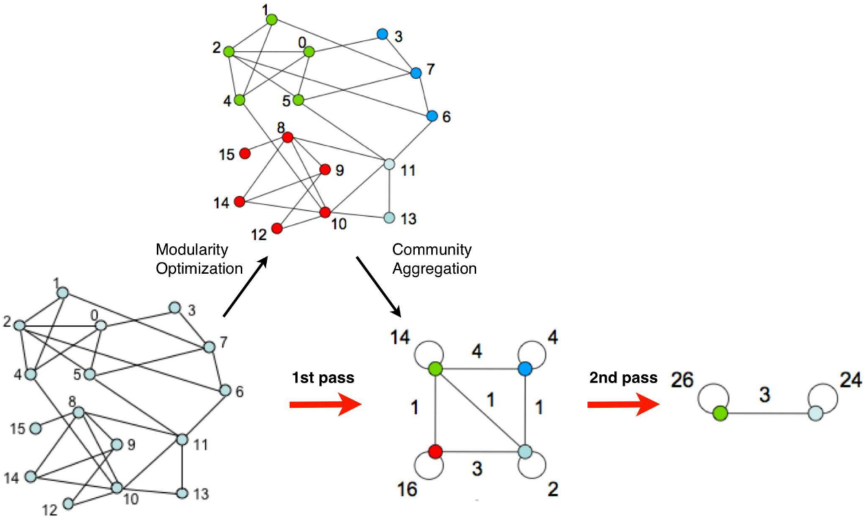
Louvain Method: Move phase

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.

Affinity Calculation

for $v \in N(u)$ **do**

  if $u \neq v$ **then**

    affinity$[\zeta[v]] \leftarrow$ affinity$[\zeta[v]] + \omega[u][v]$

    if $\zeta[v]$ not in neigh_comm **then**

      neigh_comm $\leftarrow \zeta[v]$

    end if

  end if

end for

Community Assignment

best $\leftarrow 0$
$C \leftarrow \zeta[u]$
for $D \in$ neigh_comm **do**
    if $D \neq C$ **then**
        $\Delta mod(u, C \rightarrow D) = \frac{\texttt{affinity}[D] - \texttt{affinity}[C]}{\omega(E)} + \frac{(vol(C/\ \{u\}) - vol(D/\ \{u\})) * vol(u)}{2 * \omega(E)^2}$
        if $\Delta mod(u, C \rightarrow D) >$ best **then**
            best $\leftarrow \Delta mod(u, C \rightarrow D)$
            $possible\_next\_comm \leftarrow D$
        end if
    end if
end for

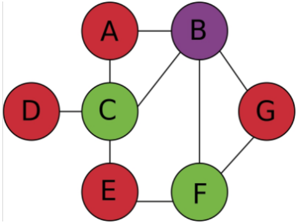
Vectorization

for $u \in V$ **do**
  $\mathit{affinity}[x] = 0, \forall x$
  for $v \in N(u)$ **do**
    $\mathit{affinity}[\zeta[v]] += w[u][v]$
  end for
  make move decision
end for

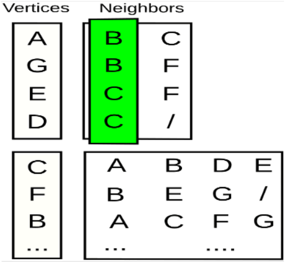
One Vertex Per Lane (OVPL)

```
for i ← 1 to min_deg(vertex_block) do
    for u ∈ vertex_block do
        v ← N[u][i]
        if u ≠ v then
            affinity[u][ζ[v]] ← affinity[u][ζ[v]] + ω[u][v]
            if ζ[v] not in neigh_comm[u] then
                neigh_comm[u] ← ζ[v]
            end if
        end if
    end for
end for
for i ← min_deg(vertex_block) + 1 to max_deg(vertex_block) do
    for u ∈ vertex_block do
        if deg(u) ≥ i then
            v ← N[u][i]
            if u ≠ v then
                affinity[u][ζ[v]] ← affinity[u][ζ[v]] + ω[u][v]
                if ζ[v] not in neigh_comm[u] then
                    neigh_comm[u] ← ζ[v]
                end if
            end if
```

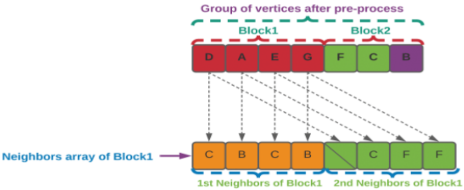
One Vertex Per Lane (OVPL)



Graph Coloring



OVPL Blocking



Physical Memory

One Neighbor Per Lane (ONPL)

- 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

N	28	16	93	44	11	72	50	23
C	1	1	4	5	3	1	3	2
M	0	1	0	0	0	3	16	0
RN	16	72	50	X	X	X	X	X

```
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);
```

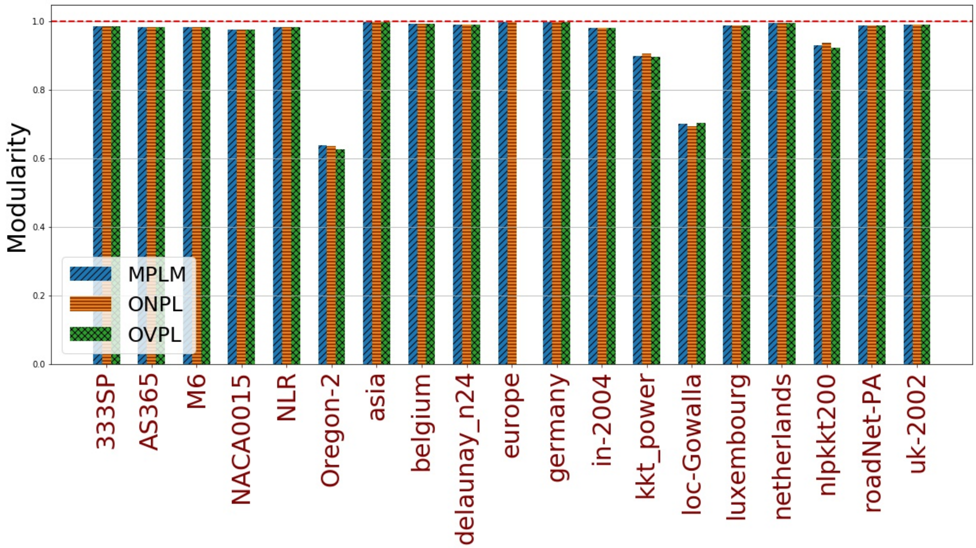
Reduce-Scatter using Horizontal Reduce

- **N** is the neighbors
- **C** is the communities
- **M** is the mask (from compress)
- **RN** is the remaining neighbors
- **RC** is the remaining communities

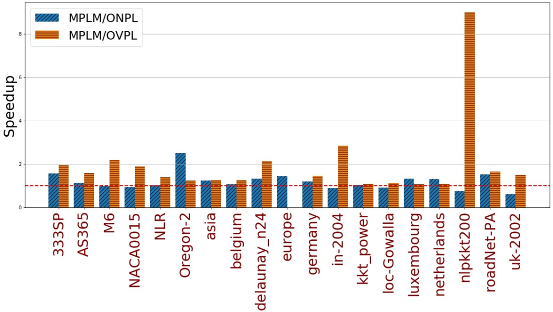
N	28	16	93	44	11	72	50	23
C	1	1	4	5	3	1	3	2
M	1	1	0	0	0	1	0	0
RN	93	44	11	50	23	X	X	X
RC	4	5	3	3	2	X	X	X

```
/// Load at most 16 neighbor vertices.
__m512i N = _mm512_loadu_si512((__m512i *) &pnt_outEdges[i]);
/// Gather community of the neighbor vertices.
__m512i C_vec = _mm512_mask_i32gather_epi32(set0, self_loop_mask, N, &zeta[0], 4);
/// Count the number of valid neighbors(u!=v)
sint vertex_cnt = _mm_popcnt_u32((unsigned)self_loop_mask);
/// It will compress communities to left that is not processed yet
C = _mm512_mask_compress_epi32(set0, self_loop_mask, C_vec);
/// Create a mask for Communities that is not processed
__mmask16 comm_mask = pow(2, vertex_cnt) - 1;
index * comm_not_processed = (index *)&C;
/// Handle the first community
__m512i first_comm = _mm512_set1_epi32(comm_not_processed[0]);
/// Calculate Mask for the first community
const __mmask16 M = _mm512_mask_cmpeq_epi32_mask(comm_mask, first_comm, C);
```

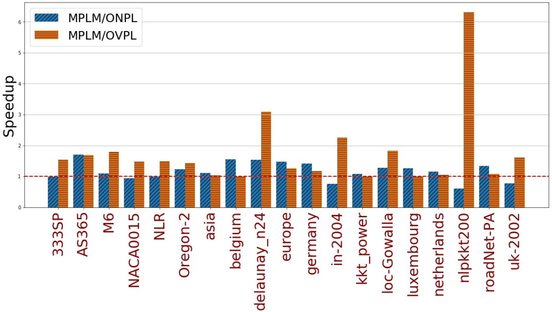
Quality of Louvain Method



Performance of Louvain Method

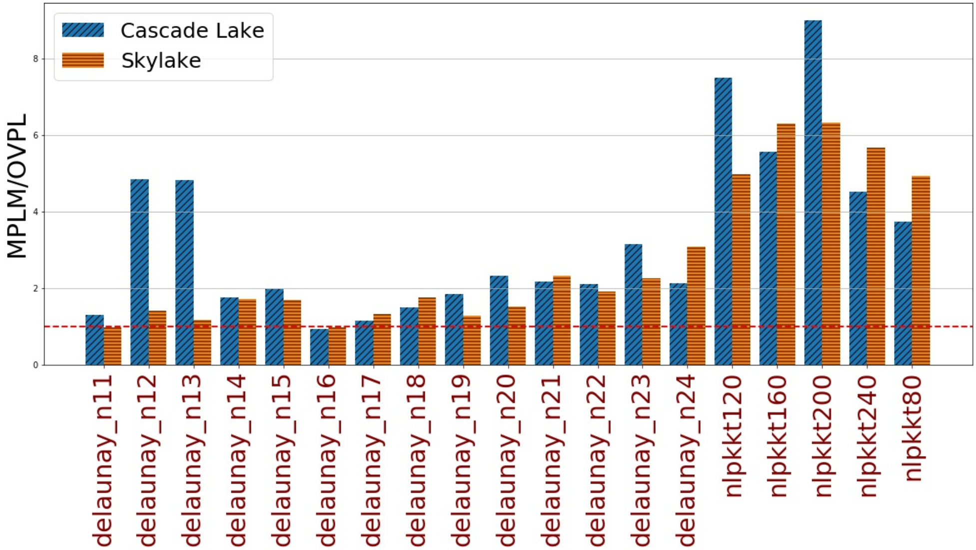


Performance on Cascade Lake (48 threads)



Performance on Skylake (36 threads)

Performance of OVPL on Graphs with Regular Degrees



Conclusion

- 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