AMD

GRAPH COLORING ON THE GPU AND SOME TECHNIQUES TO IMPROVE LOAD IMBALANCE

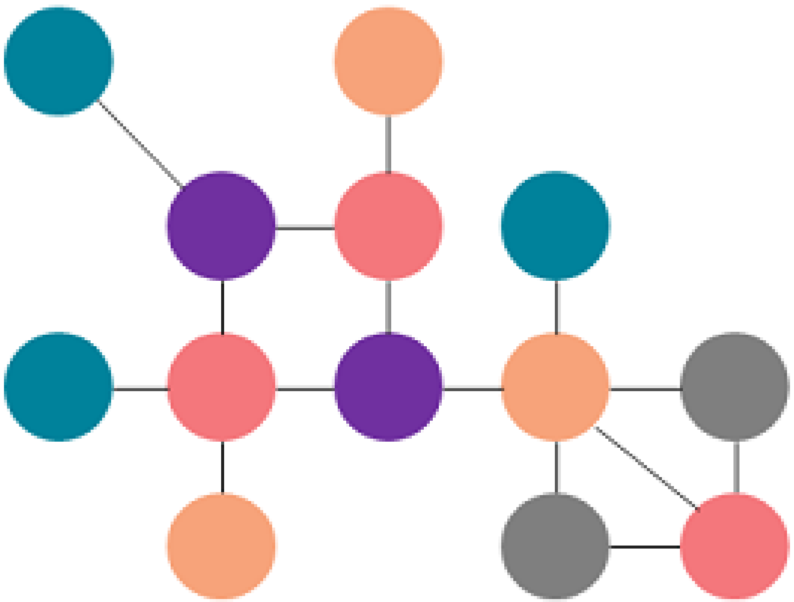
SHUAI CHE, GREGORY RODGERS, BRAD BECKMANN, STEVE REINHARDT
AMD
SPEAKER: DIBYENDU DAS

GRAPH COLORING

- Graph coloring is a key building block for many graph applications
- Graph coloring presents load imbalance across GPU threads
- Its program behavior changes over time in different iterations
  - Load distribution across threads
  - Static approach usually is not effective

GRAPH COLORING

- Label a graph so that no adjacent vertices have the same color
  - We do not study optimal coloring in this work



BASELINE COLORING ALGORITHM

- **Randomization-based approach (baseline)**
  - Assign vertices with random values
  - Repeat the following steps until all the vertices are colored
    - Each thread checks if a vertex is a local maximum using random numbers
    - **If** the vertex is a local maximum, assign the vertex a new color
    - **else** ignore the vertex and evaluate it in the following iteration

BASELINE COLORING ALGORITHM

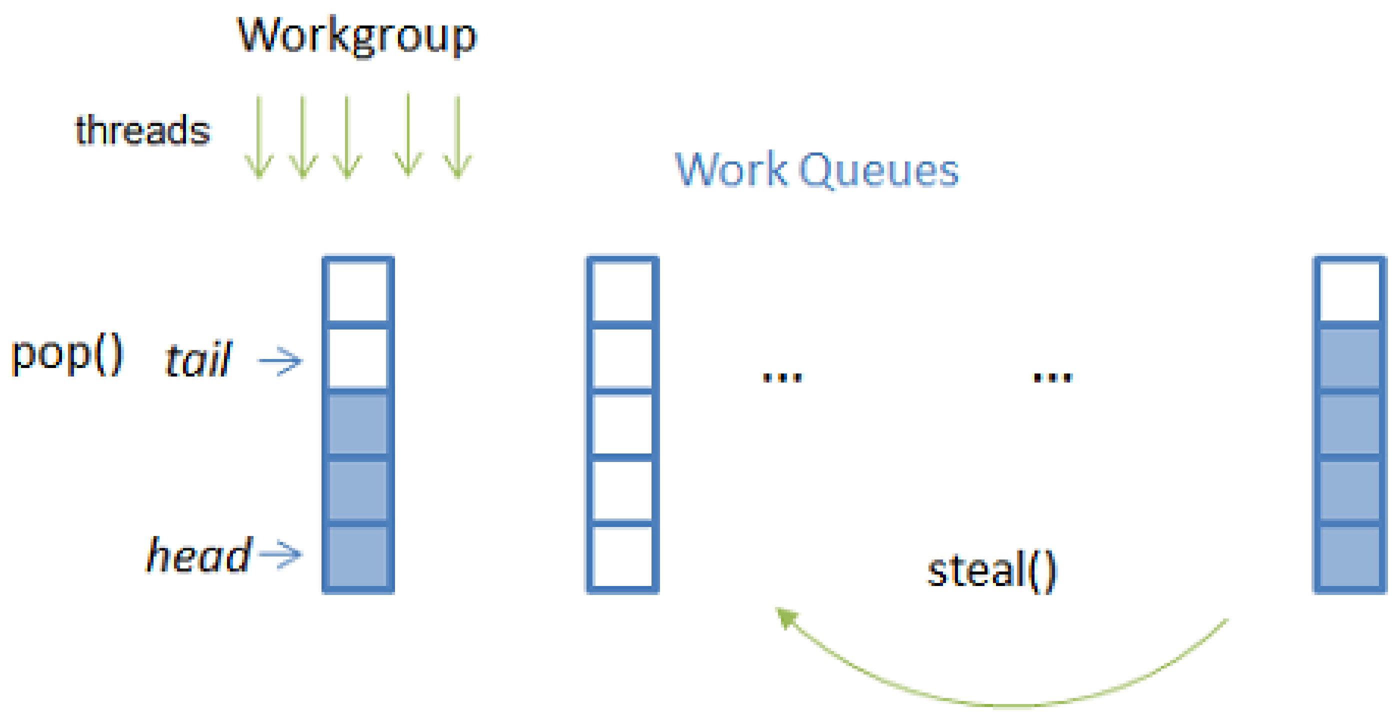
- Issues of the baseline algorithm
  - Different vertices have different degrees
  - Load Imbalance across GPU threads. Short running threads have to wait for long running threads, wasting compute resources and power
- We first apply workstealing to balance workloads across workgroups
  - Each workgroup is associated with a work queue
  - Each workgroup consists of multiple threads, each of which processes a vertex and its neighborlist
  - The workstealing algorithm uses a similar approach used by Tsigas and Cedermann (*GPU Computing Gems*)

WORKSTEALING

- Two basic operations in workstealing

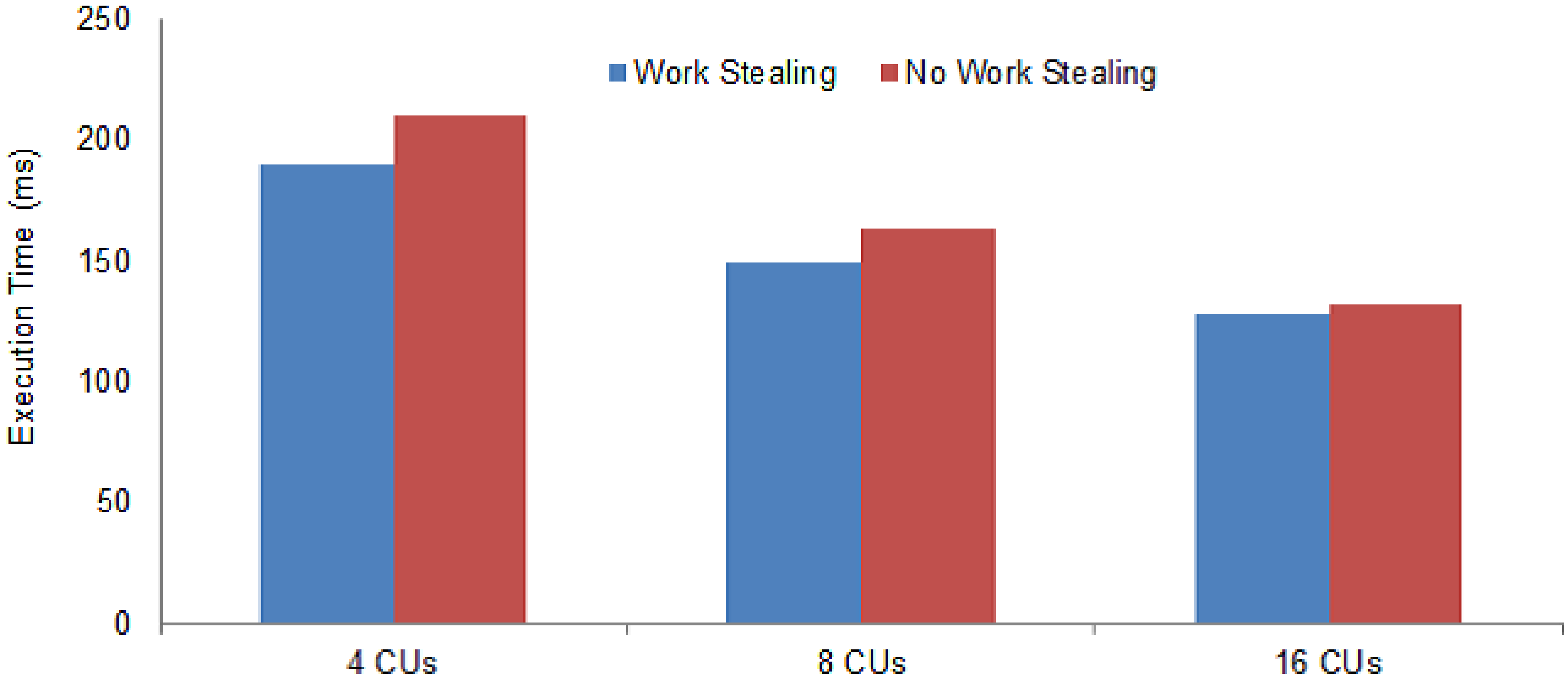
  Pop dequeues an element from the tail of the local queue

  Steal dequeues an element from the head of a remote queue, when the local queue is empty



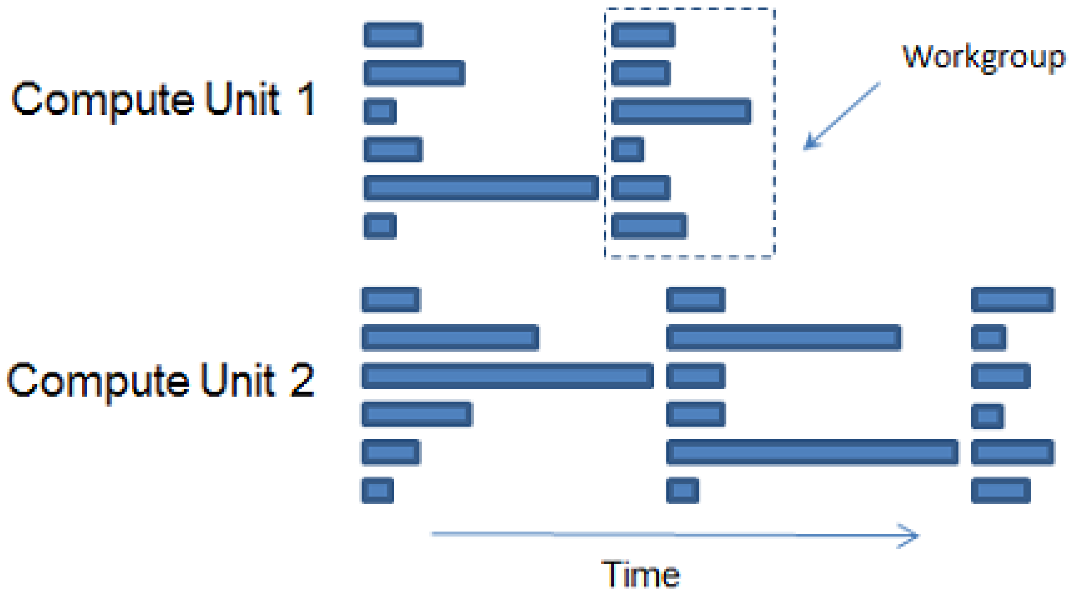
PERFORMANCE OF WORKSTEALING

- Less than 10% performance improvement



WORKSTEALING

- Work stealing in the workgroup granularity only partially resolves the overall load imbalance problem
- Significant imbalance exists within a workgroup, especially for unstructured graphs (e.g., power-law graphs)



A HYBRID APPROACH

- Vertex degree can be a heuristic to estimate the running time of a thread to process a vertex and its neighborlist
- We color large-degree vertices first, so that they will not be evaluated in the following iterations. Load imbalance across threads will be improved.

HYBRID ALGORITHM

Phase 1 (degree-based coloring)

Precalculate degrees of all the vertices

Repeat the following steps until a *switching condition* is met

Each thread checks if a vertex is a local maximum using **vertex-degree values**

If the vertex is a local maximum, assign the vertex a new color

else ignore the vertex and evaluate it in the following iteration

Phase 2 (randomization-based coloring)

Repeat the following steps until all the vertices are colored

Each thread checks if a vertex is a local maximum using **random numbers**

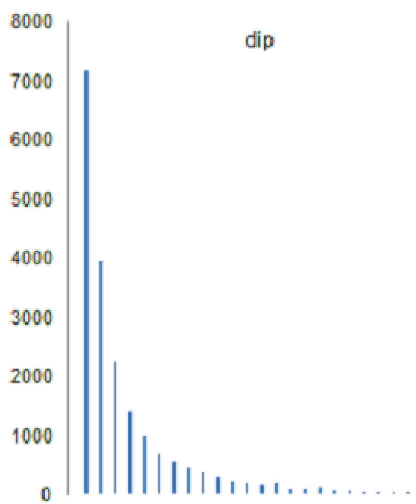
If the vertex is a local maximum, assign the vertex a new color

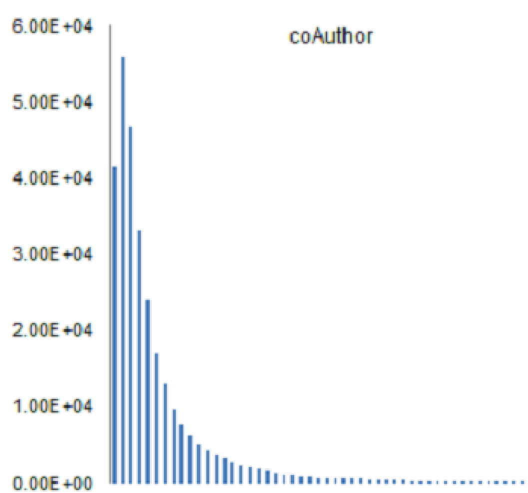
else ignore the vertex and evaluate it in the following iteration

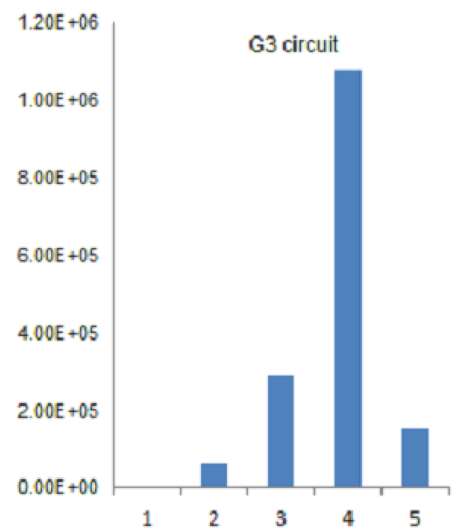
Note: for Phase 1, we only color a vertex if and only if it is a local maximum and it is the only local maximum in the neighborhood

HYBRID ALGORITHM

- **Degree-based coloring** will get diminishing benefits because more and more vertices will have smaller, same degrees (e.g. dip and coauthor). Thus, we switch to **randomization-based coloring**
- Switch condition:
  - No. of colorable of vertices using the degree-based approach is less than a threshold
    For example, no. of colorable vertices is not big enough to fit all the GPU cores
  - For many unstructured graphs, most of the large-degree vertices can be colored in only a few iterations.

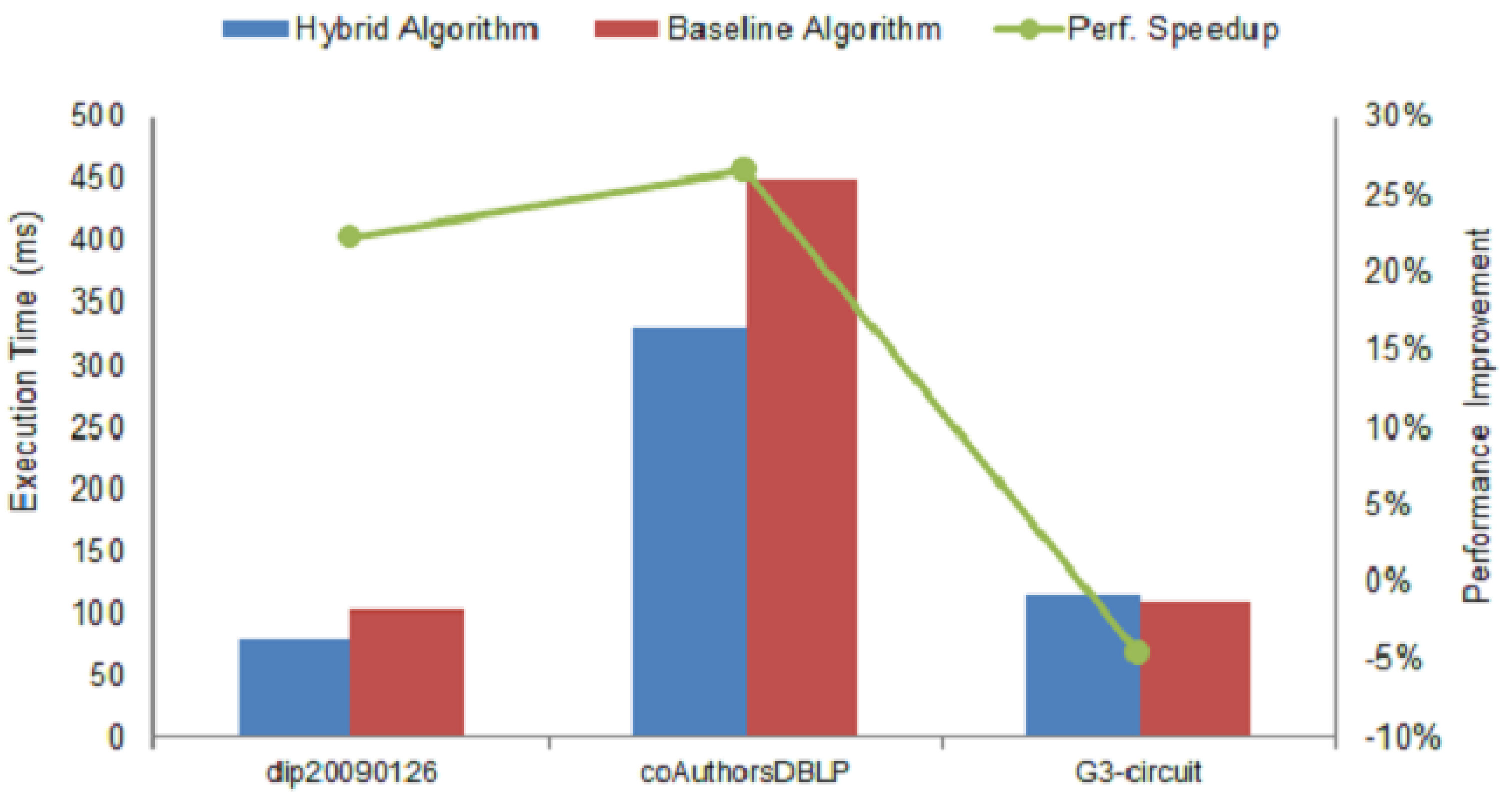






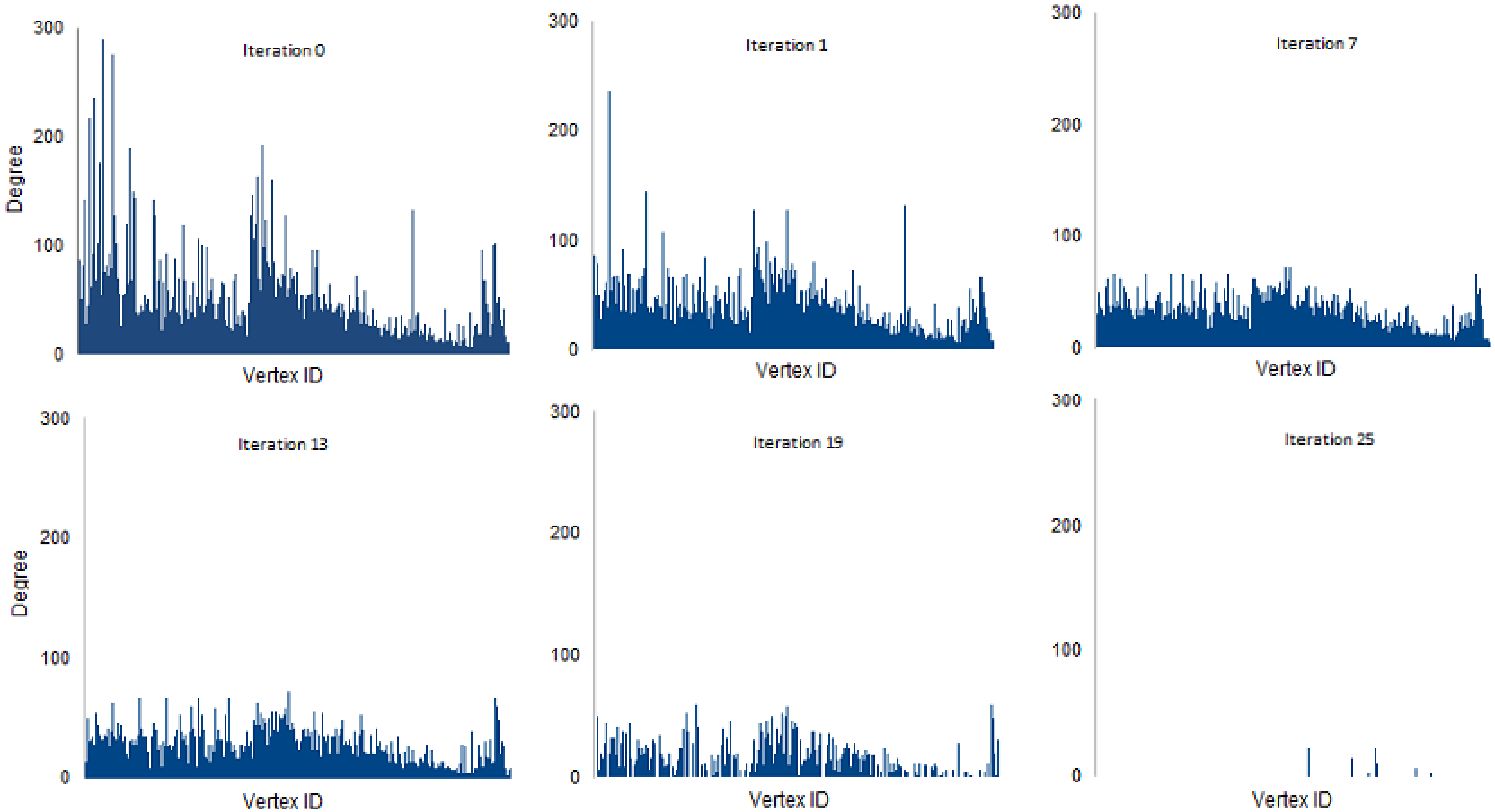
PERFORMANCE BENEFITS

- The hybrid algorithm is 23% faster than the baseline, randomization-based approach for dip20090126, and 27% faster for coAuthorDBLP
- The hybrid algorithm is especially effective to color unstructured graphs



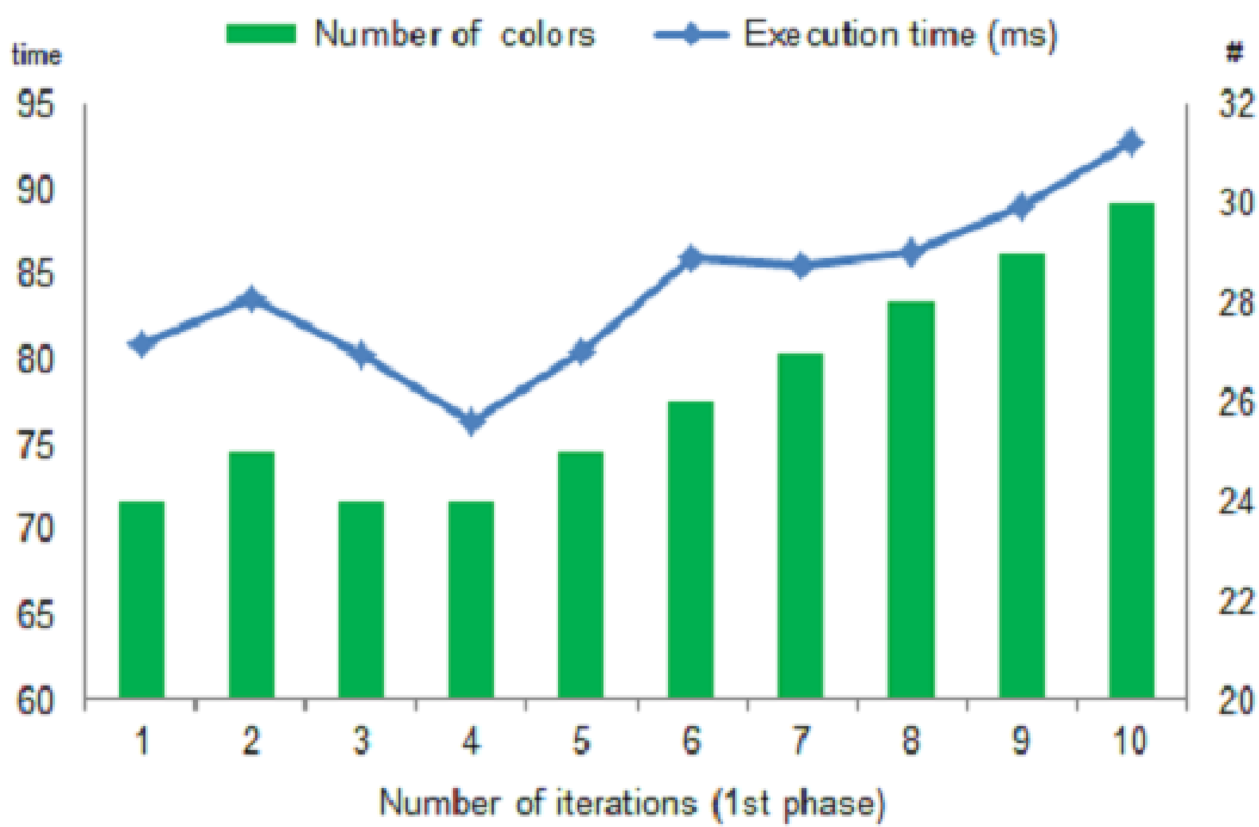
ACTIVE VERTICES ACROSS ITERATIONS

- High-degree vertices are colored in the first few iterations. Load imbalance is improved for the following iterations.



IMPACT OF PHASE CHANGE

- The best case: switching at the 4th iteration for dip
- 15% performance difference between the best and worst cases
- It is an open research question to determine the optimal switch point.
  – Currently, some threshold value is used



CONCLUSION AND FUTURE WORK

- This paper shows the cause of SIMD load imbalance when performing coloring
- We show workstealing offer only limited performance improvement, due to significant imbalance within a workgroup
- We propose a hybrid 2-phase graph coloring algorithm with the combination of degree and randomization-based strategies
- Future work includes:
  - Extension to multiple machine nodes
  - Evaluation with different data layouts and inputs
  - Integration of this algorithm into other graph applications (e.g., independent set)