Scaling and Quality of Modularity Optimization Methods for Graph Clustering

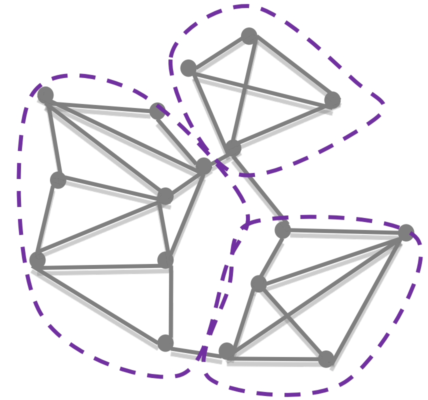
Sayan Ghosh[+], Mahantesh Halappanavar[+], Antonino Tumeo[+], Ananth Kalyanaraman[*]

[+]Pacific Northwest National Laboratory, Richland, WA
[*]Washington State University, Pullman, WA

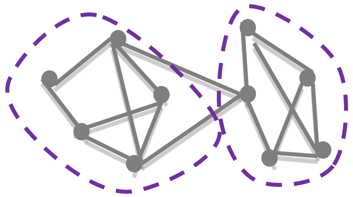
Graph Challenge Innovation award
2019 IEEE High Performance Extreme Computing Conference

Graph Clustering (Community Detection)

- Problem: Given $G(V,E,\omega)$, identify tightly knit groups of vertices that strongly correlate to one another within their group, and sparsely so, outside.



Input :
- $V = \{1,2,\ldots n\}$
- E: a set of M edges
- $\omega(e)$: weight of edge e (non-negative)
- $m = \Sigma_{\forall e \in E}\ \omega(e)$



Output :
- A partitioning of V into k mutually disjoint clusters $P = \{C_1, C_2, \ldots C_k\}$

- Modularity (Newman, 2004): A statistical measure for assessing the quality of a given community-wise partitioning P of the vertices V

Notation	Definition
$C(i)$	Cluster containing vertex i
$e_{i\text{->}C(i)}$	Number of edges from *i* to vertices in *C*(*i*)
a_C	Sum of the degree of all vertices in cluster *C*

$$Q = \frac{1}{2m}\sum_{\forall i \in V} e_{i\rightarrow C(i)} - \sum_{\forall C \in P}\left(\frac{a_C}{2m}\cdot\frac{a_C}{2m}\right)$$

Fraction of intra-cluster edges

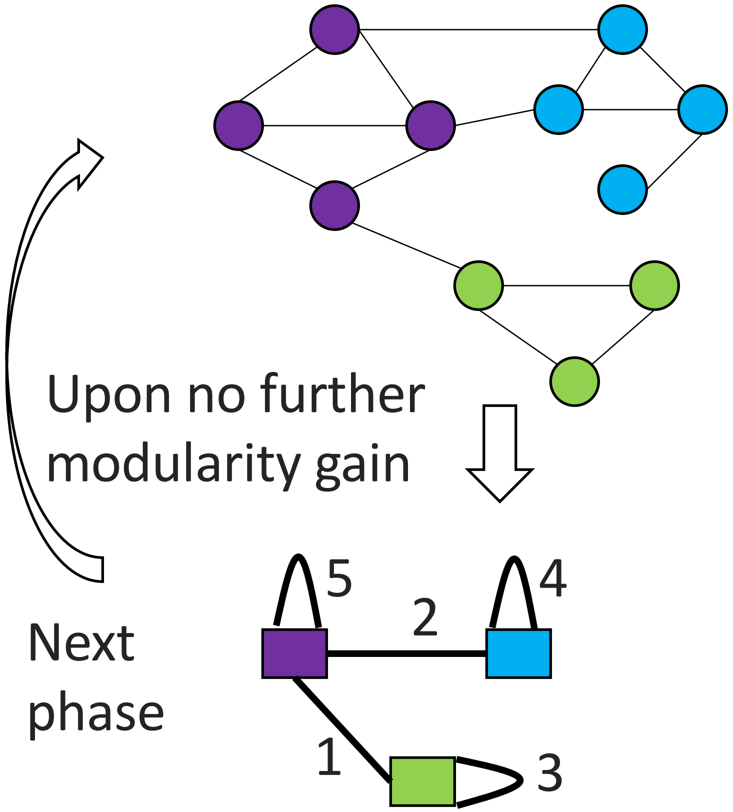
Equivalent fraction in a random graph

Louvain method (Blondel et al. 2008)

Input: G(V,E,ω)

Goal: Compute a partitioning of V that maximizes modularity (Q)

Initialize: Every vertex starts in its own community (i.e., *C(i)={i}*)



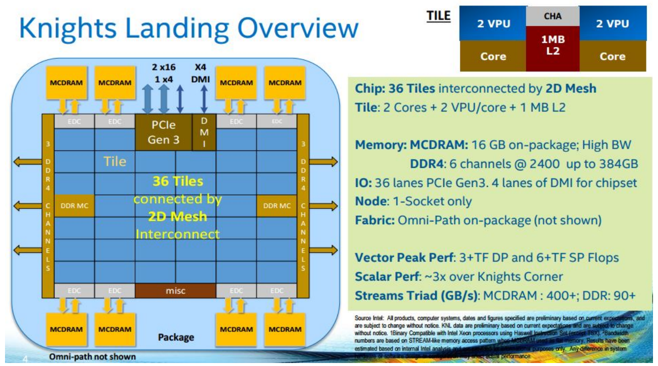
Multi-phase multi-iterative heuristic

Within each iteration:

- **For** every vertex $i \in V$:
  1. Let *C(i)* : current community of *i*
  2. Compute modularity gain (ΔQ) for moving *i* into each of *i*'s neighboring communities
  3. Let C_{max} : neighboring community with largest ΔQ
  4. If ($\Delta Q>0$) { Set $C(i) = C_{max}$ }

Update on Distributed Louvain implementation : Vite

- Ported Vite to Intel KNL processors on the ALCF Theta supercomputer
  - KNL is dead, long live KNL: KNL served as a precursor to modern multicore systems with HBM
  - Used KNL 16 GB MCDRAM as addressable memory – allocated some heavily used C++ containers on MCDRAM
- Implemented a balanced graph distribution to reduce overall communication (by 80%) at the expense of extra I/O



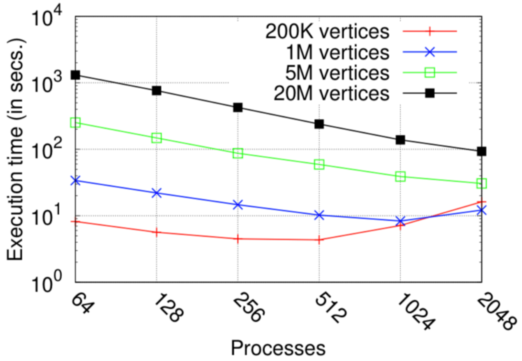


Fig. 1. Distributed Louvain clustering strong scaling results of highOverlap-highBlockSize challenge datasets on ALCF Theta.

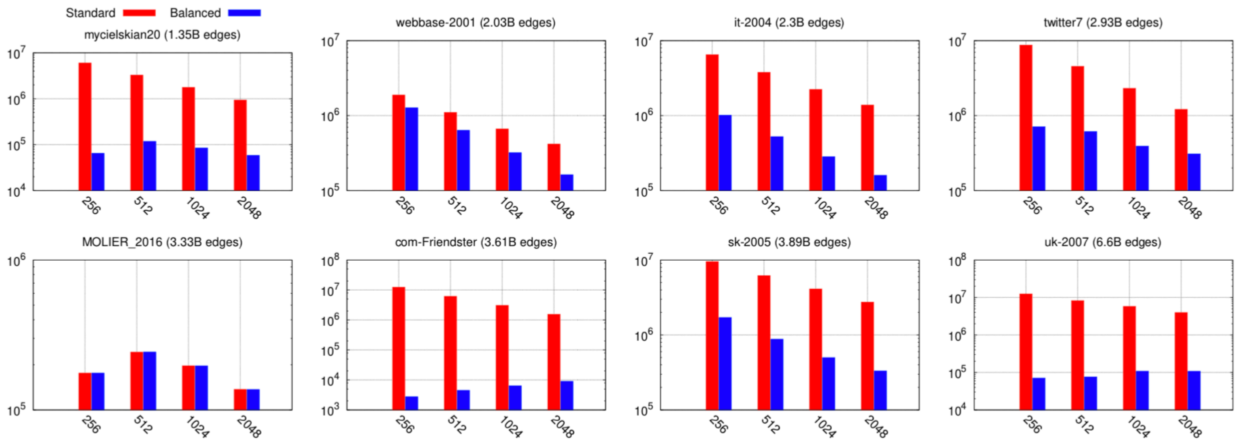


Fig. 2. Graph distribution characteristics of standard and edge-balanced vertex-based distribution. *Y-axis*: Standard Deviation (#Edges/process), *X-axis*: #Processes.

High standard deviation in #Edges/process means more imbalance

Vite results on ALCF Theta for billion-edge graphs

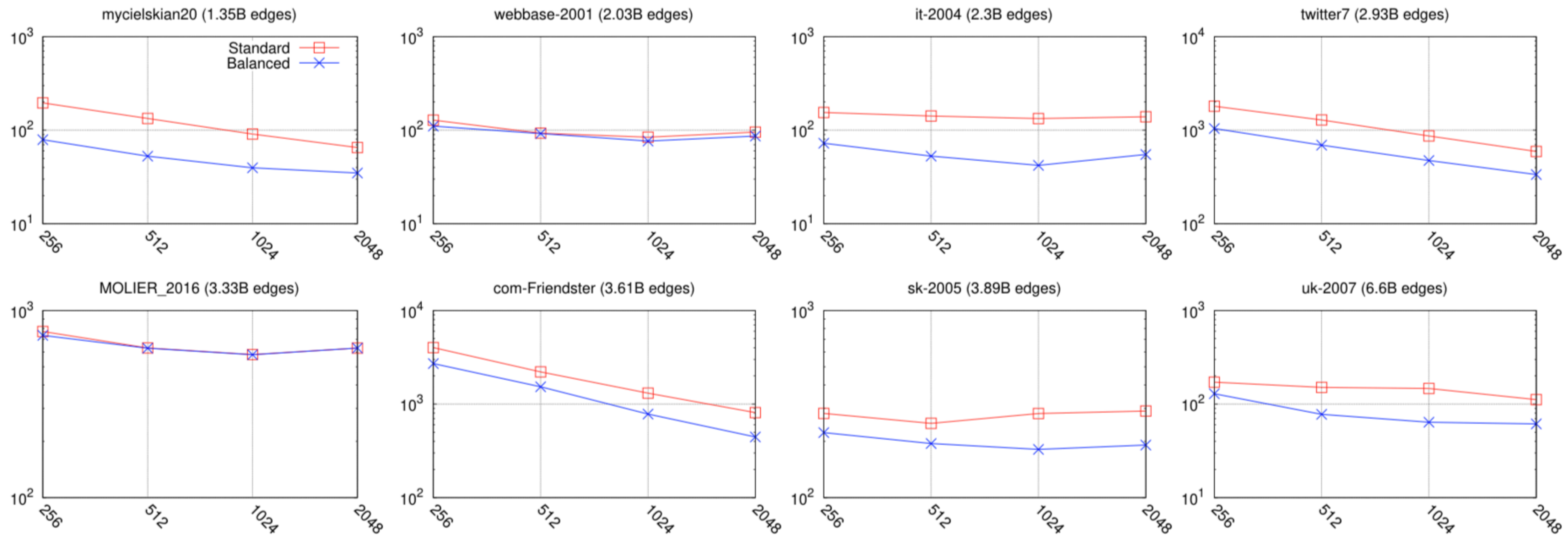


Fig. 3. Distributed-memory Louvain execution times of real-world graphs with over a billion edges, using the standard and edge-balanced graph distribution. *Y-axis*: Total execution time (in secs.), *X-axis*: #Processes.

0-80% improvement in end-to-end execution time using the balanced distribution

Thanks

- US DoE ExaGraph project
- Battelle PNNL
- NSF award CCF 1815467 to Washington State University
- Argonne Leadership Computing Facility

Code: https://github.com/Exa-Graph/vite