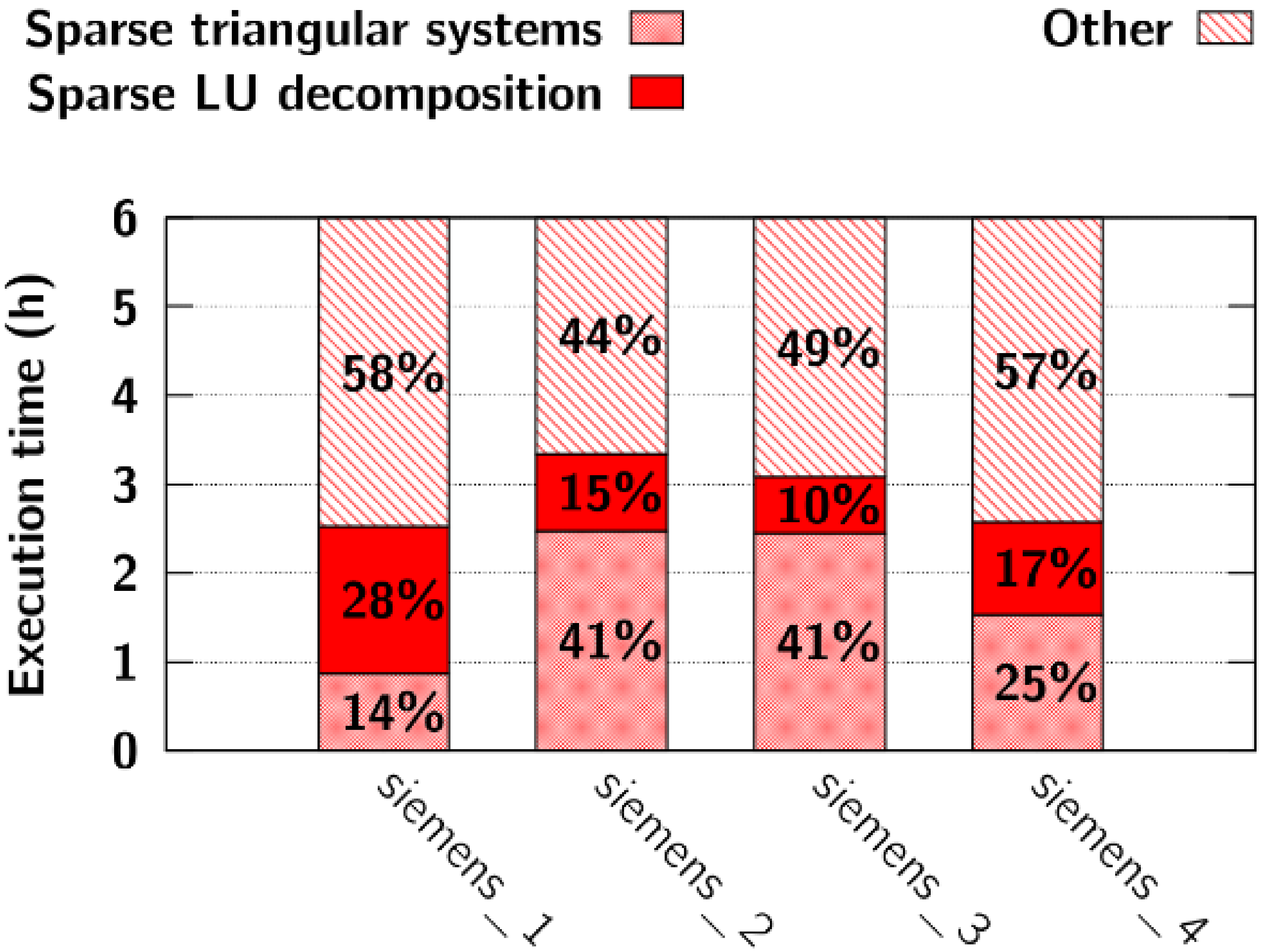
Trading between task-level and data-level parallelism to solve sparse triangular linear systems

Andrea Picciau

Circuits and Systems Group
Imperial College London

12 October 2015



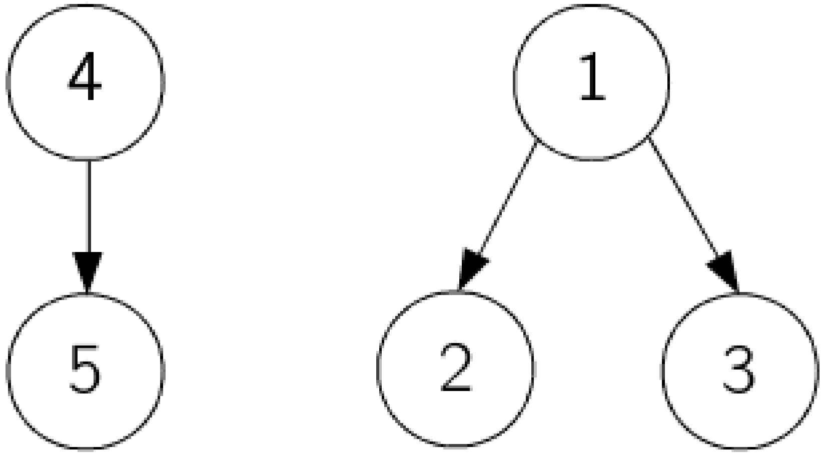
Data dependencies in sparse triangular systems are represented using directed acyclic graphs

$$\begin{cases} x_1 = b_1 \\ l_{2,1}x_1 + x_2 = b_2 \\ l_{3,1}x_1 + x_3 = b_3 \\ x_4 = b_4 \\ l_{5,4}x_4 + x_5 = b_5 \\ l_{6,3}x_3 + l_{6,5}x_5 + x_6 = b_6 \\ l_{7,6}x_6 + x_7 = b_7 \\ l_{8,6}x_6 + x_8 = b_8 \end{cases}$$

Data dependencies in sparse triangular systems are represented using directed acyclic graphs

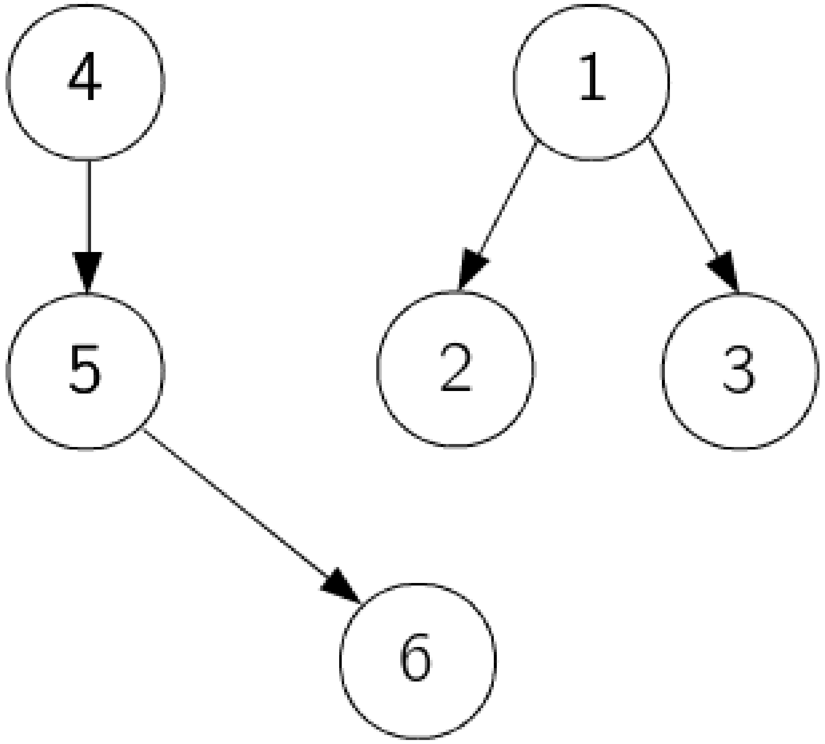
$$\begin{cases} x_1 = b_1 \\ l_{2,1}x_1 + x_2 = b_2 \\ l_{3,1}x_1 + x_3 = b_3 \\ x_4 = b_4 \\ l_{5,4}x_4 + x_5 = b_5 \\ l_{6,3}x_3 + l_{6,5}x_5 + x_6 = b_6 \\ l_{7,6}x_6 + x_7 = b_7 \\ l_{8,6}x_6 + x_8 = b_8 \end{cases}$$

Data dependencies in sparse triangular systems are represented using directed acyclic graphs



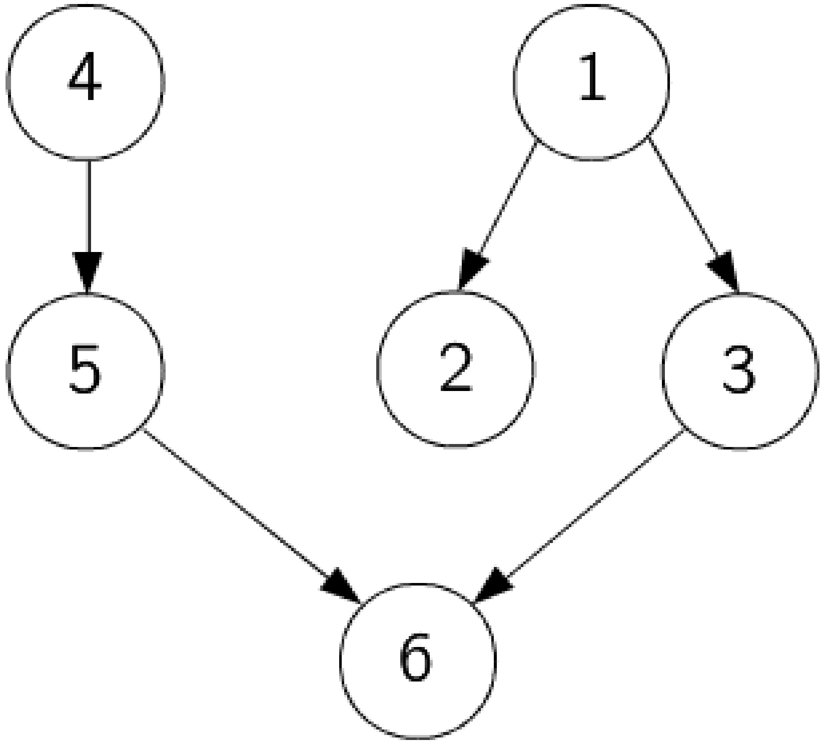
$$\begin{cases} x_1 = b_1 \\ l_{2,1}x_1 + x_2 = b_2 \\ l_{3,1}x_1 + x_3 = b_3 \\ x_4 = b_4 \\ l_{5,4}x_4 + x_5 = b_5 \\ l_{6,3}x_3 + l_{6,5}x_5 + x_6 = b_6 \\ l_{7,6}x_6 + x_7 = b_7 \\ l_{8,6}x_6 + x_8 = b_8 \end{cases}$$

Data dependencies in sparse triangular systems are represented using directed acyclic graphs



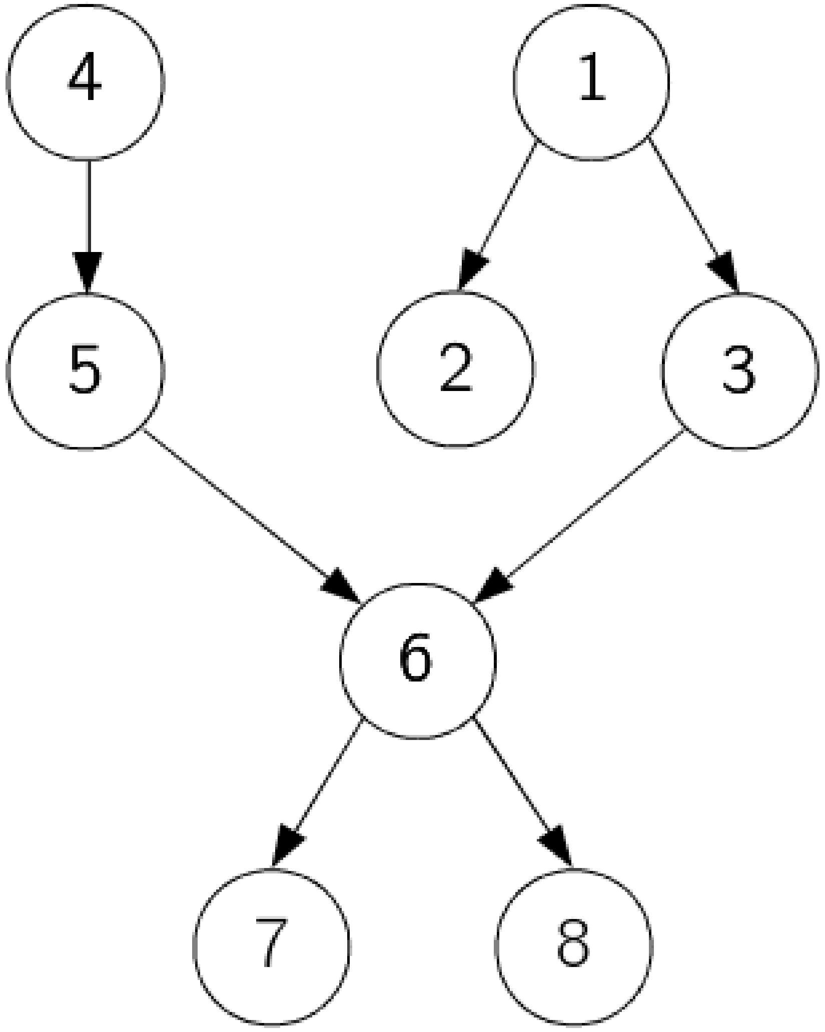
$$\begin{cases} x_1 = b_1 \\ l_{2,1}x_1 + x_2 = b_2 \\ l_{3,1}x_1 + x_3 = b_3 \\ x_4 = b_4 \\ l_{5,4}x_4 + x_5 = b_5 \\ l_{6,3}x_3 + l_{6,5}x_5 + x_6 = b_6 \\ l_{7,6}x_6 + x_7 = b_7 \\ l_{8,6}x_6 + x_8 = b_8 \end{cases}$$

Data dependencies in sparse triangular systems are represented using directed acyclic graphs



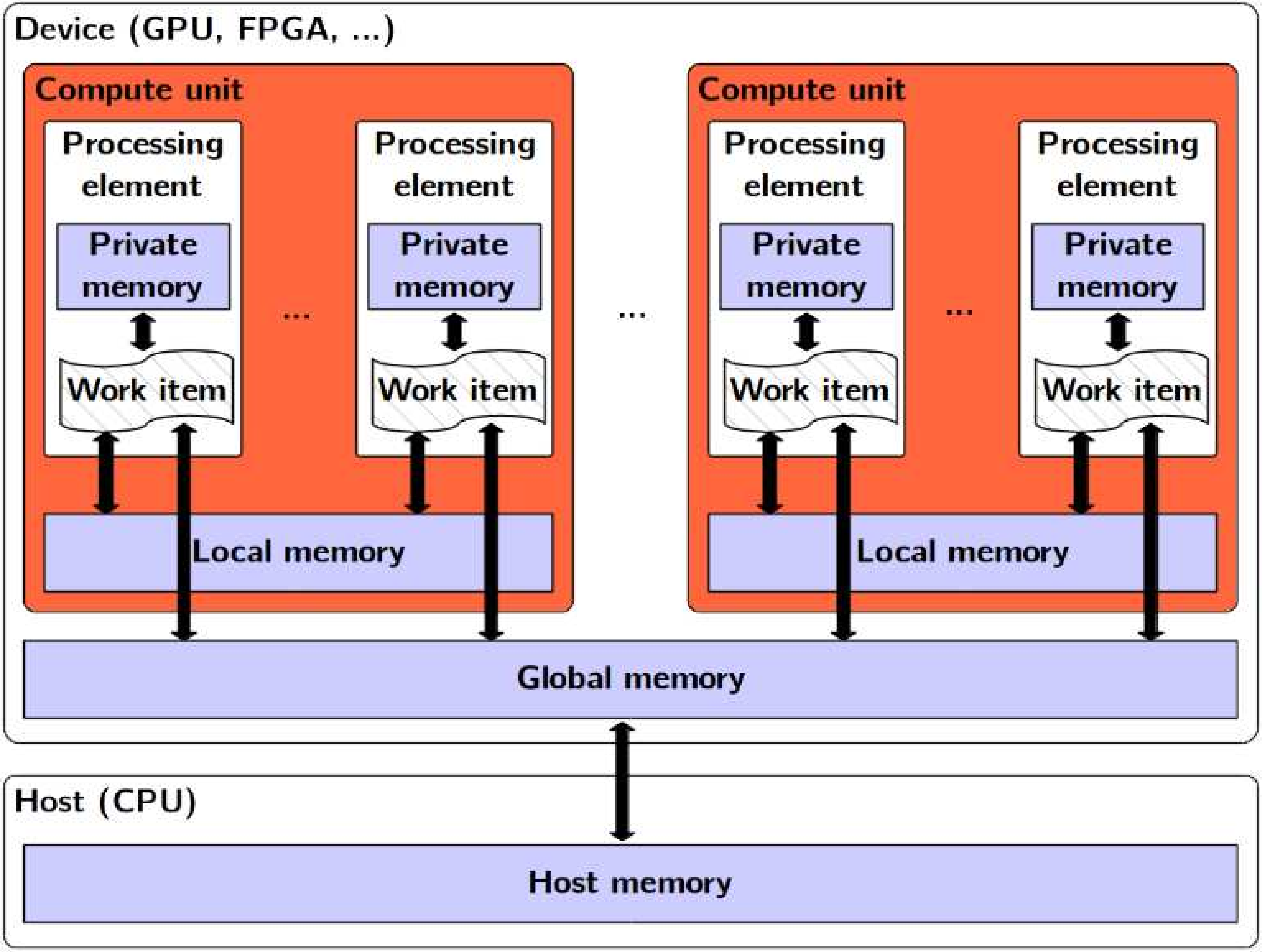
$$\begin{cases} x_1 = b_1 \\ l_{2,1}x_1 + x_2 = b_2 \\ l_{3,1}x_1 + x_3 = b_3 \\ x_4 = b_4 \\ l_{5,4}x_4 + x_5 = b_5 \\ l_{6,3}x_3 + l_{6,5}x_5 + x_6 = b_6 \\ l_{7,6}x_6 + x_7 = b_7 \\ l_{8,6}x_6 + x_8 = b_8 \end{cases}$$

Data dependencies in sparse triangular systems are represented using directed acyclic graphs

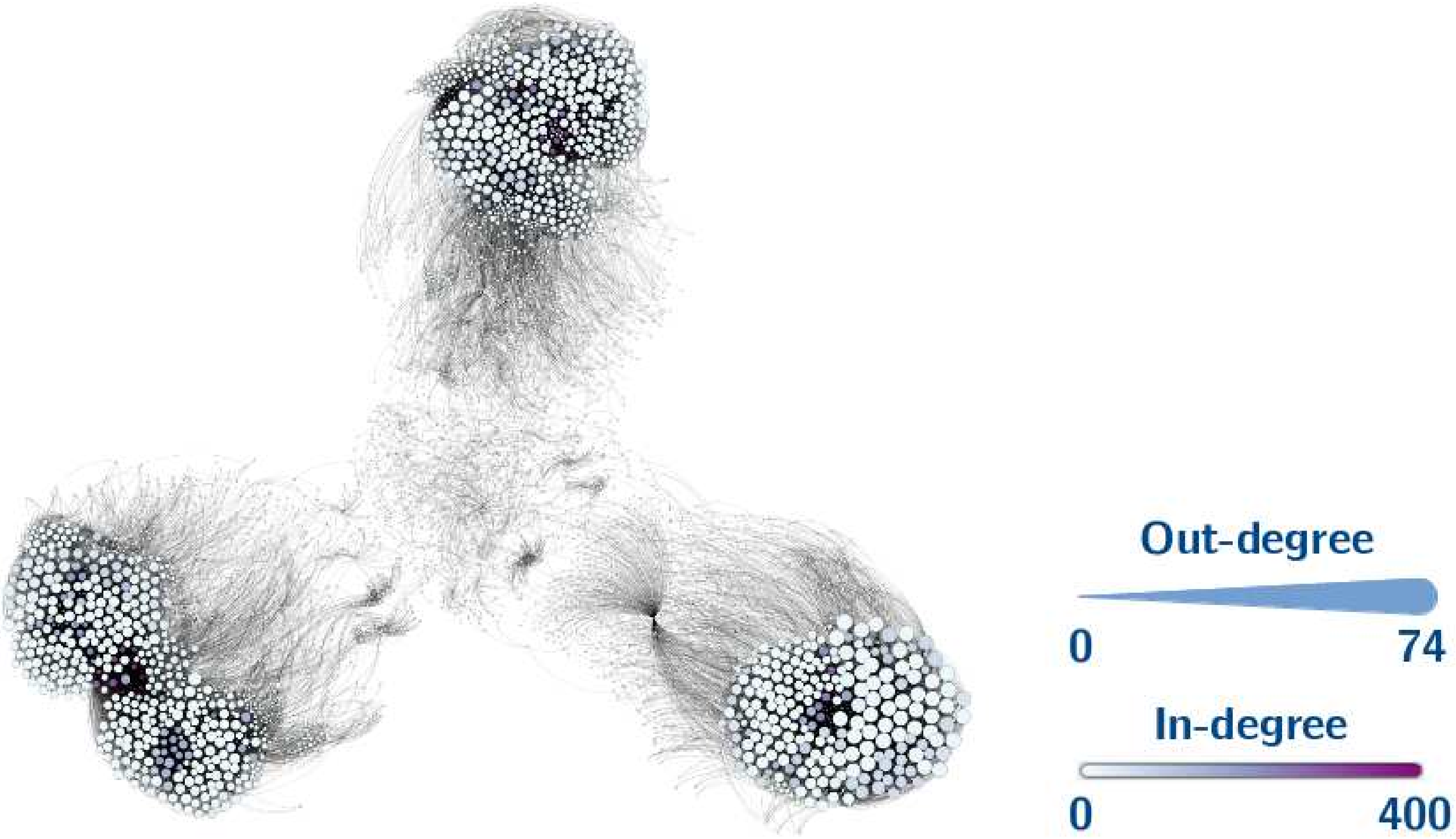


$$\begin{cases} x_1 = b_1 \\ l_{2,1}x_1 + x_2 = b_2 \\ l_{3,1}x_1 + x_3 = b_3 \\ x_4 = b_4 \\ l_{5,4}x_4 + x_5 = b_5 \\ l_{6,3}x_3 + l_{6,5}x_5 + x_6 = b_6 \\ l_{7,6}x_6 + x_7 = b_7 \\ l_{8,6}x_6 + x_8 = b_8 \end{cases}$$

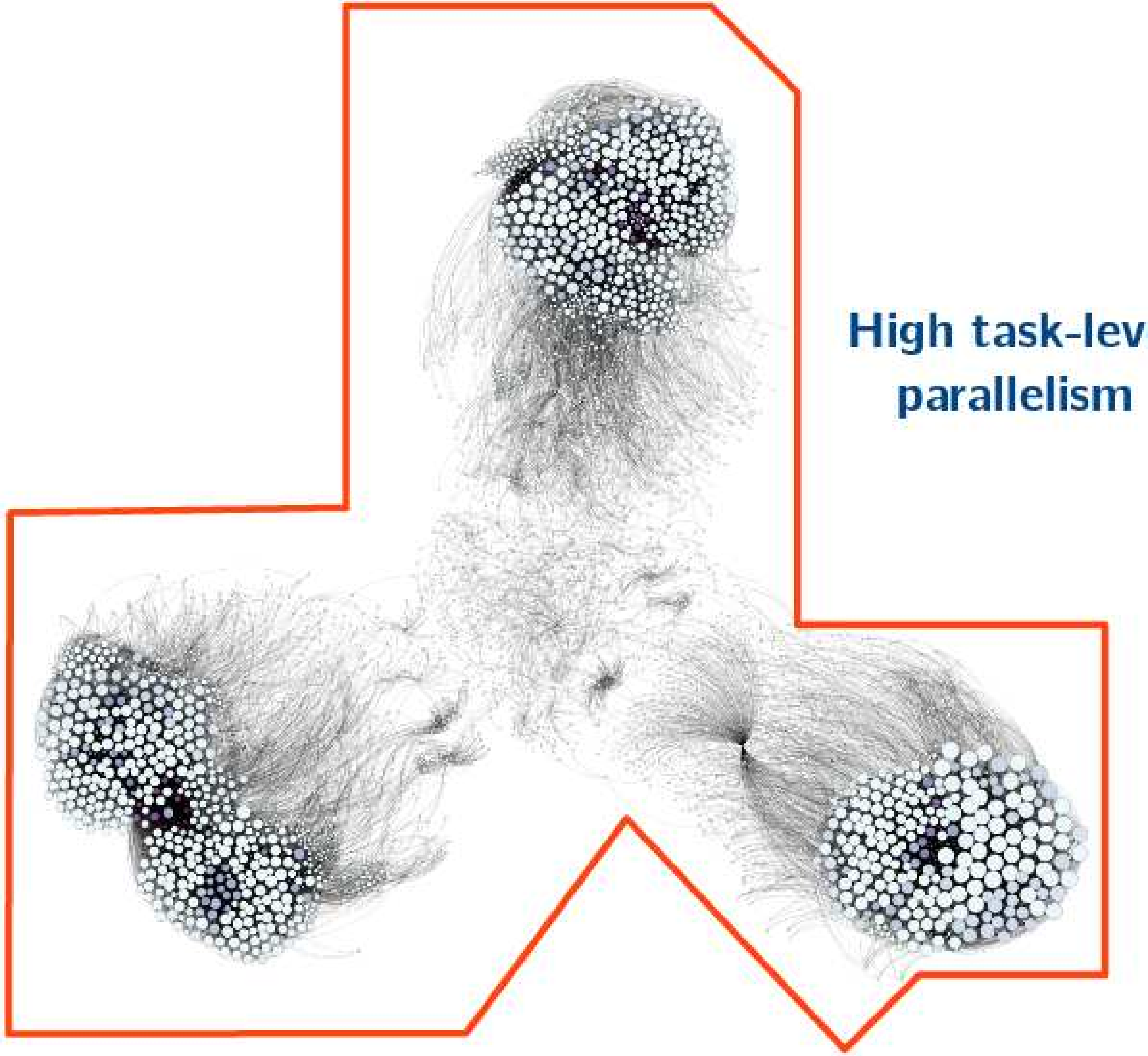
GPUs are composed of compute units, each one containing a local memory

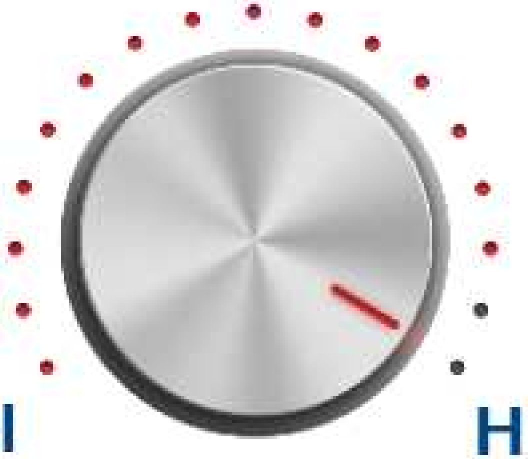


Our approach is to partition the graph into sub-graphs and process each one inside a compute unit



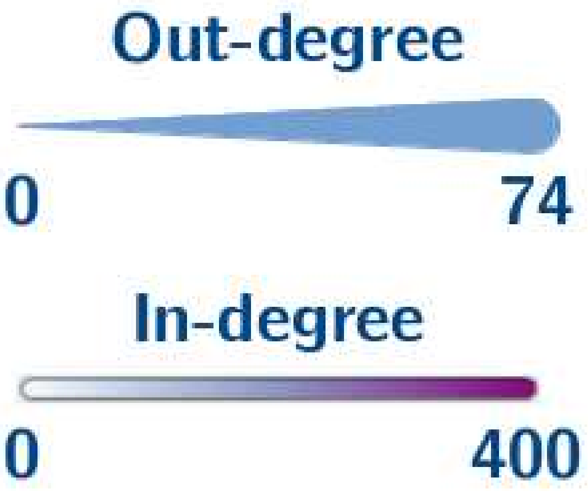
Our approach is to partition the graph into sub-graphs and process each one inside a compute unit



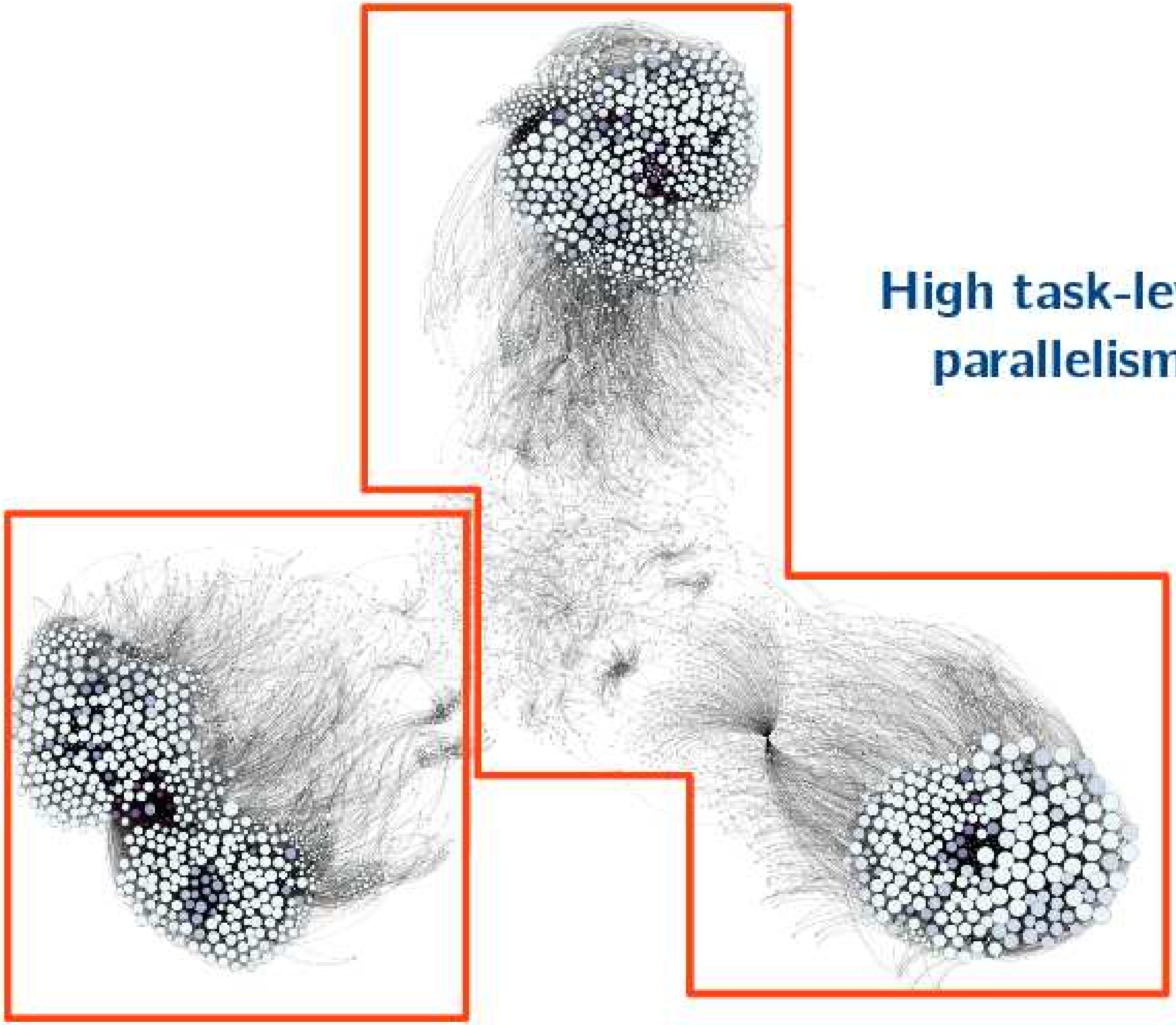


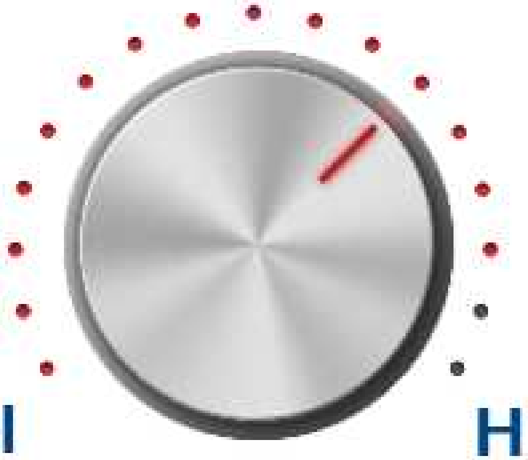
High task-level parallelism

High data-level parallelism



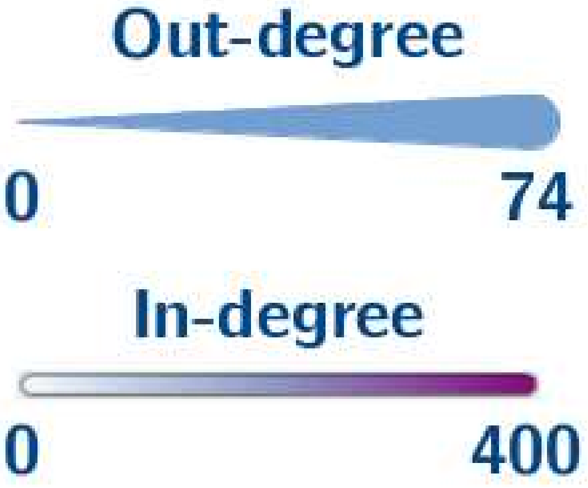
Our approach is to partition the graph into sub-graphs and process each one inside a compute unit



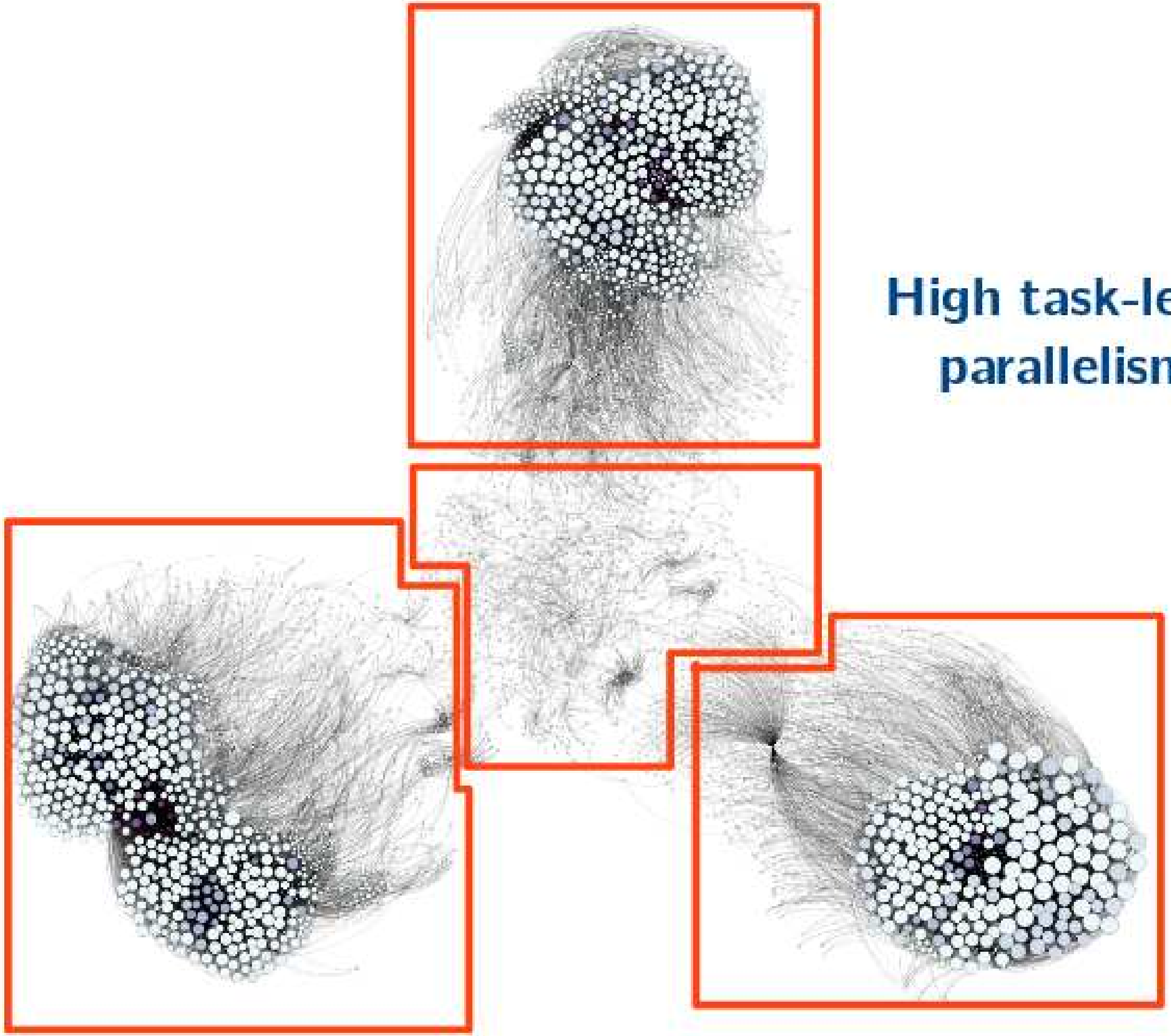


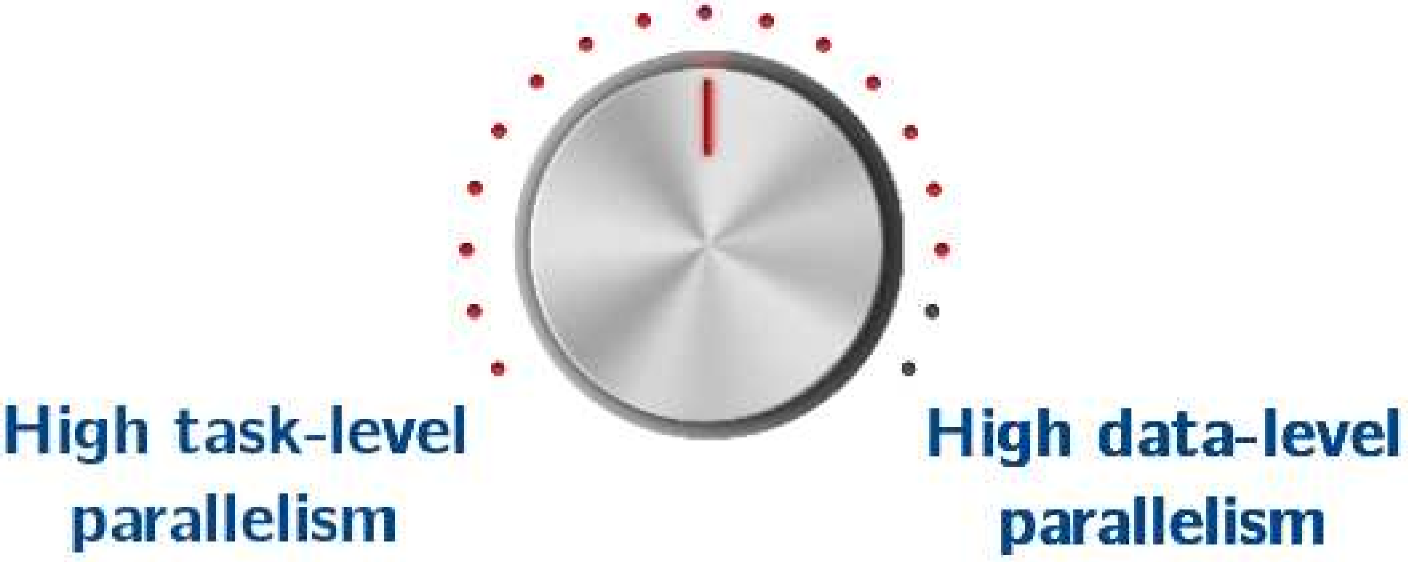
High task-level parallelism

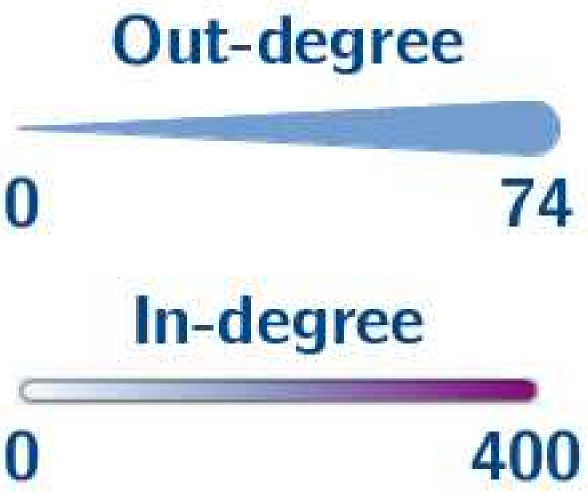
High data-level parallelism



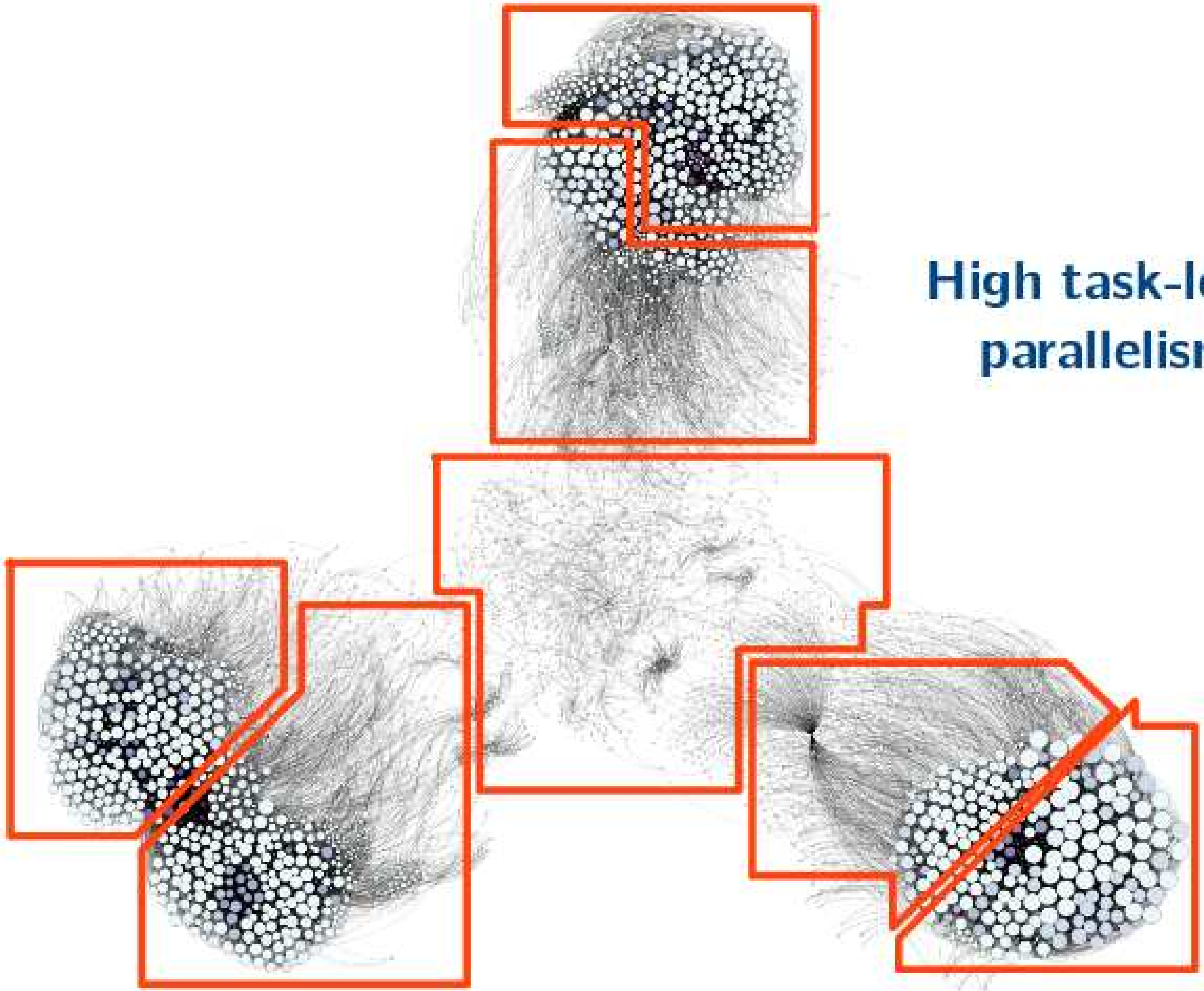
Our approach is to partition the graph into sub-graphs and process each one inside a compute unit

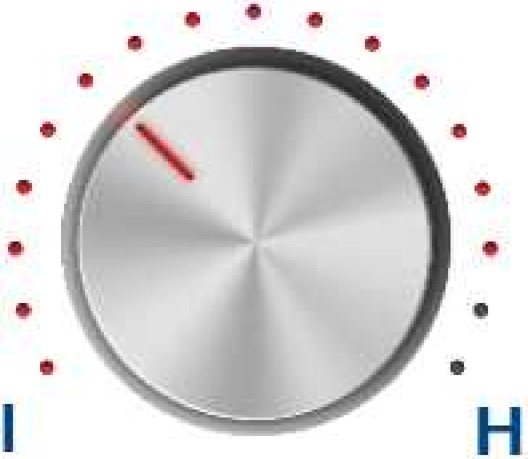






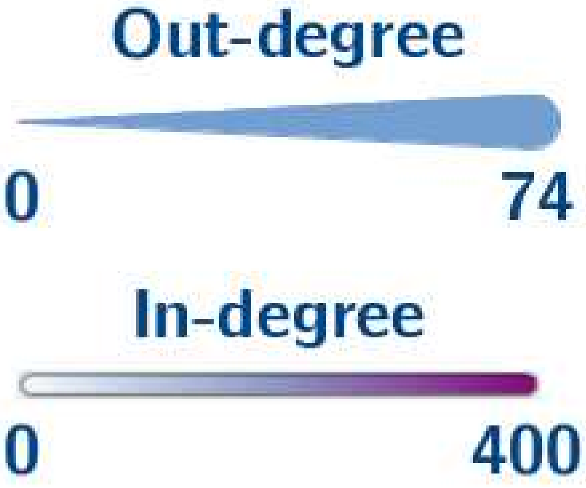
Our approach is to partition the graph into sub-graphs and process each one inside a compute unit



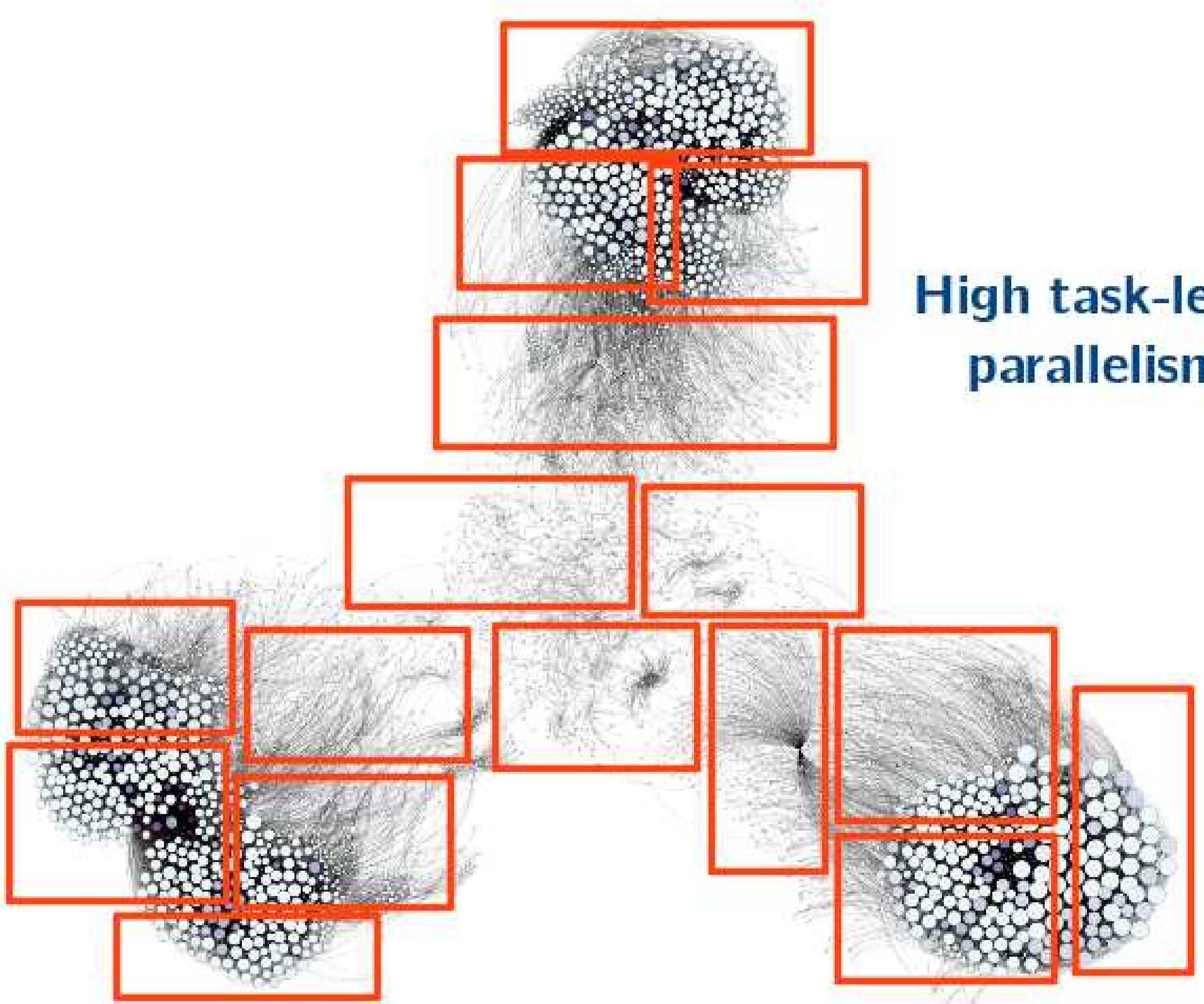


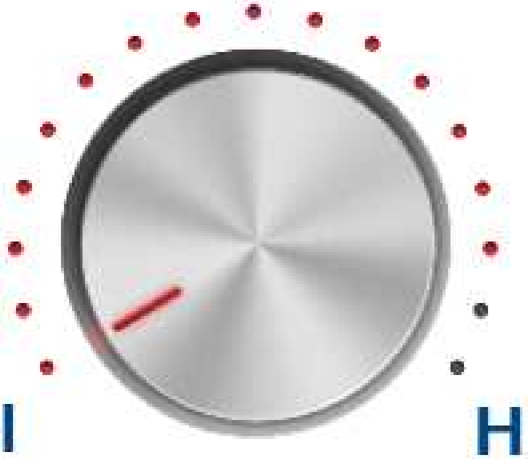
High task-level parallelism

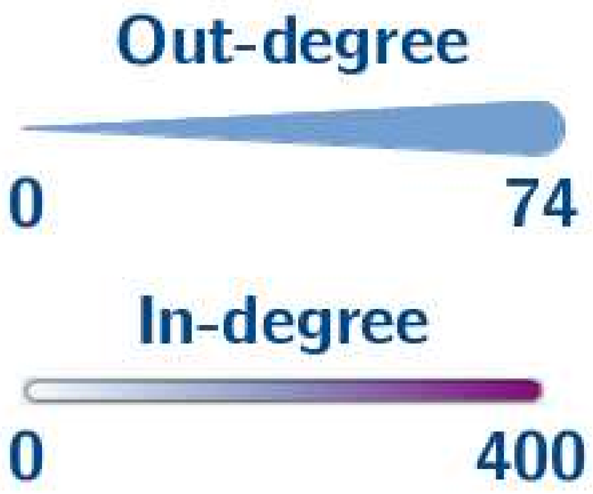
High data-level parallelism



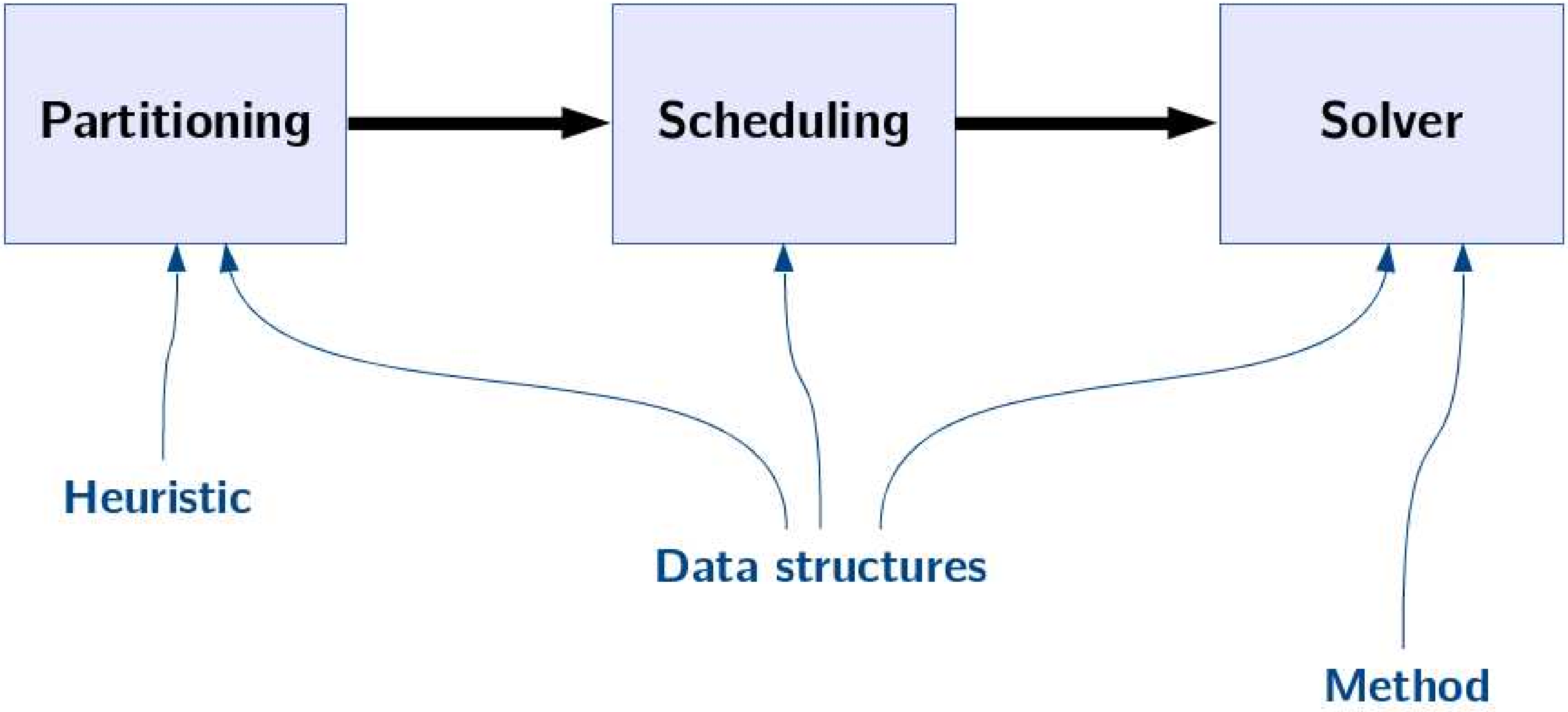
Our approach is to partition the graph into sub-graphs and process each one inside a compute unit





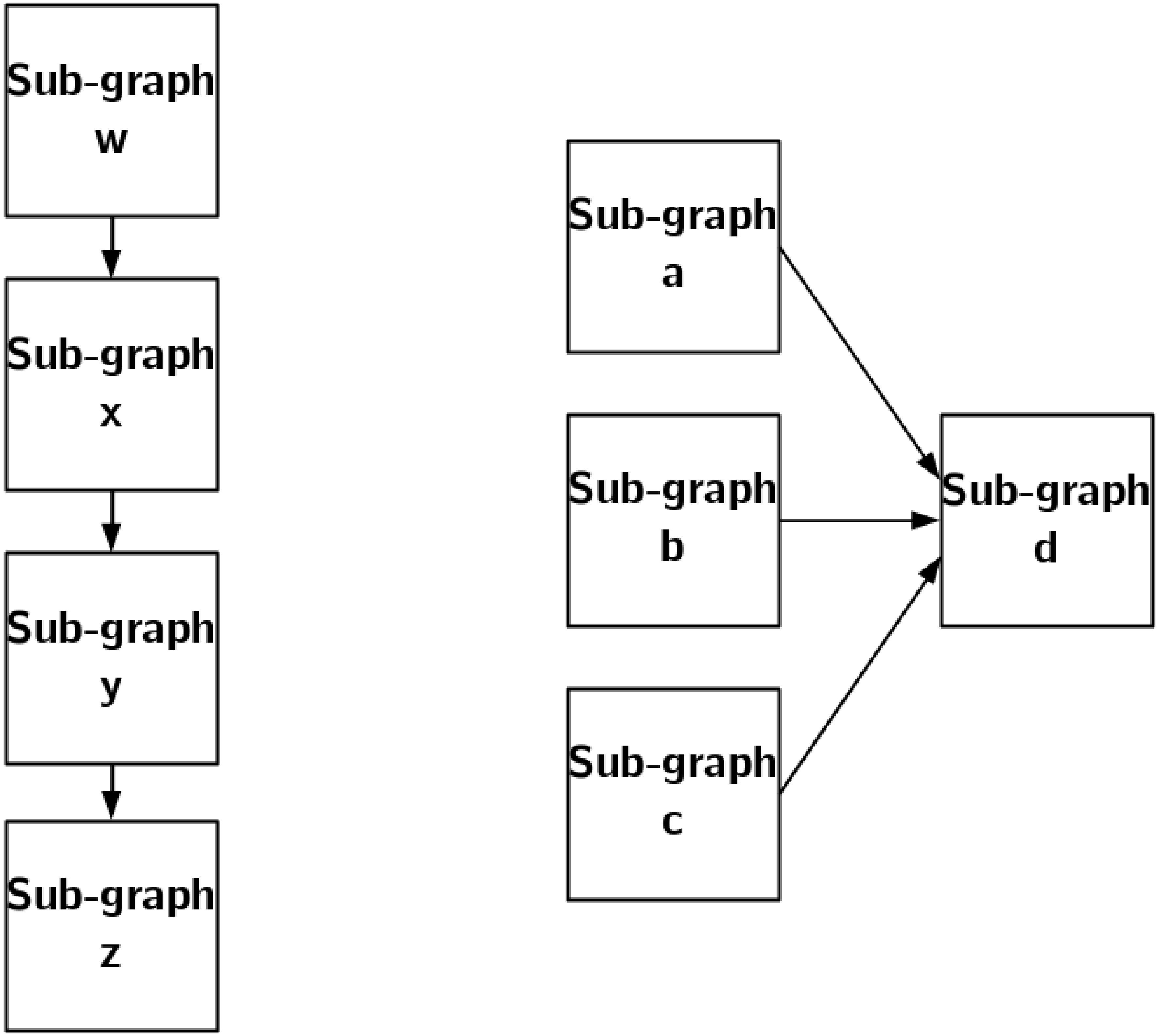


Our technique is divided in three phases



Imperial College London

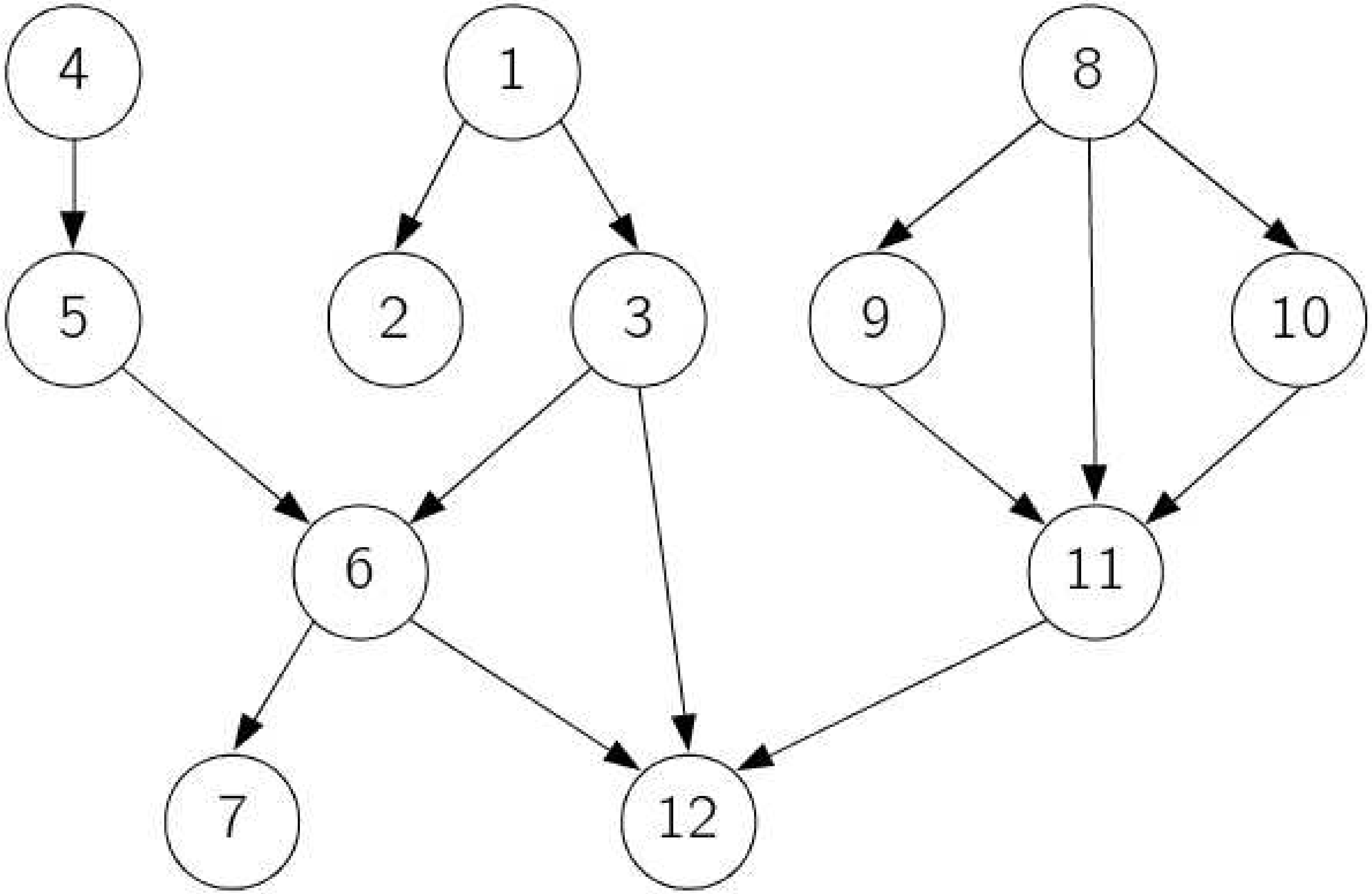
Our partitioning heuristic aims to achieve a specific dependency pattern for sub-graphs



At each step of the algorithm, we consider the vertices in order of their out-degree

Local memory size = 3

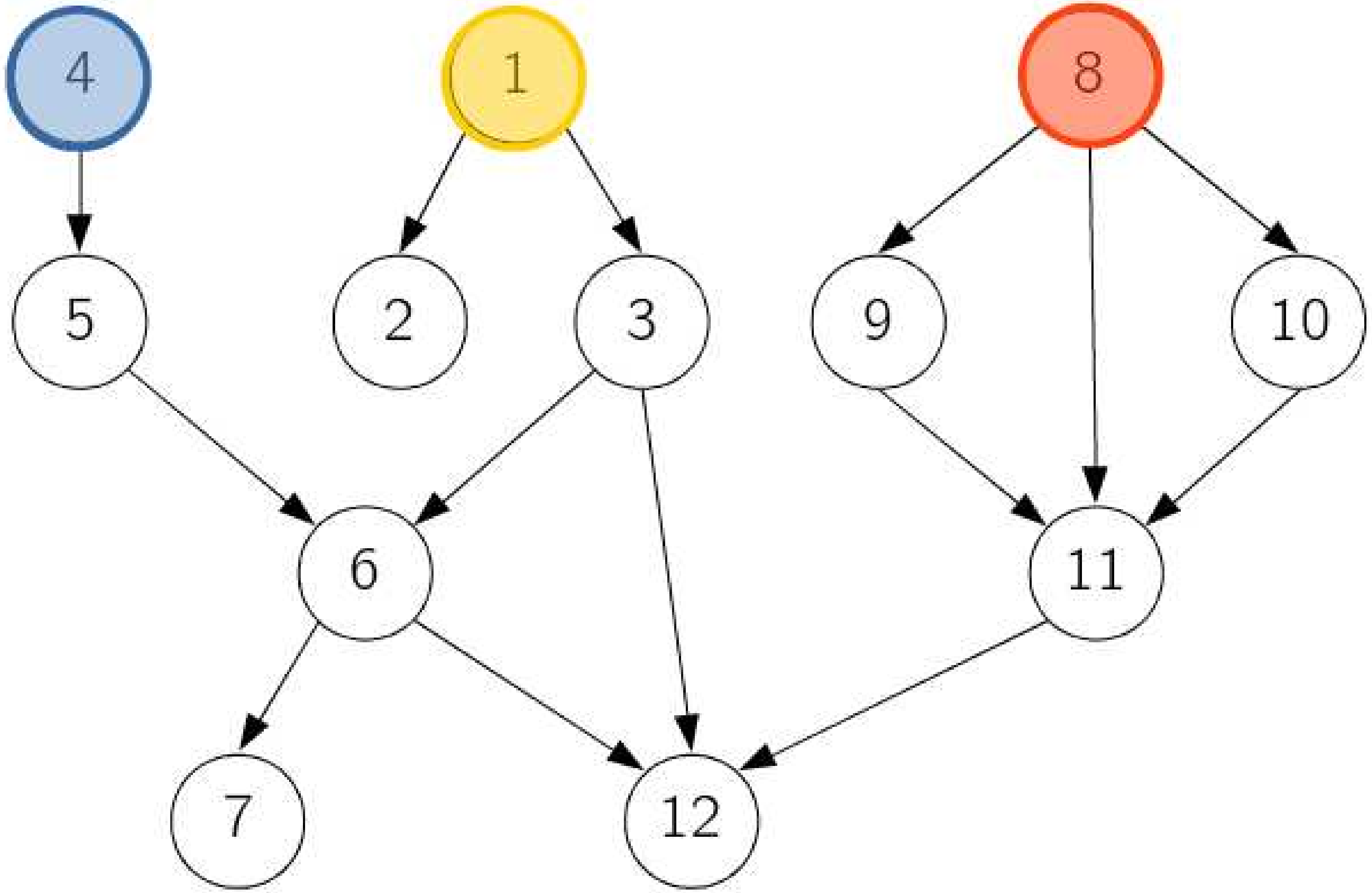
Priority rule: 🟥 1 🟨 2 🟦 3 🟩 4



At each step of the algorithm, we consider the vertices in order of their out-degree

Local memory size = 3

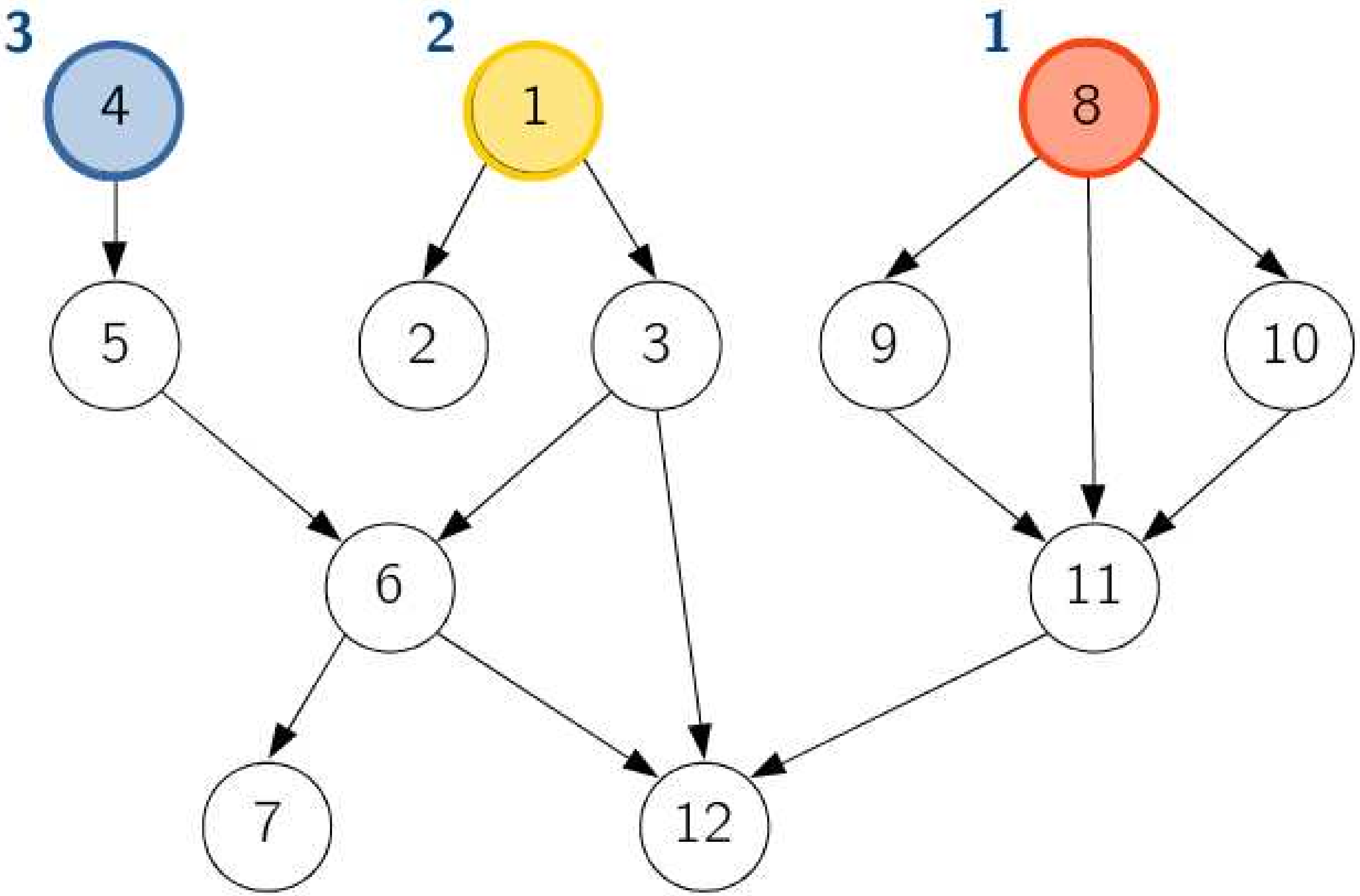
Priority rule: 🟥 1 🟨 2 🟦 3 🟩 4



At each step of the algorithm, we consider the vertices in order of their out-degree

Local memory size = 3

Priority rule: 🟥 1 🟨 2 🟦 3 🟩 4

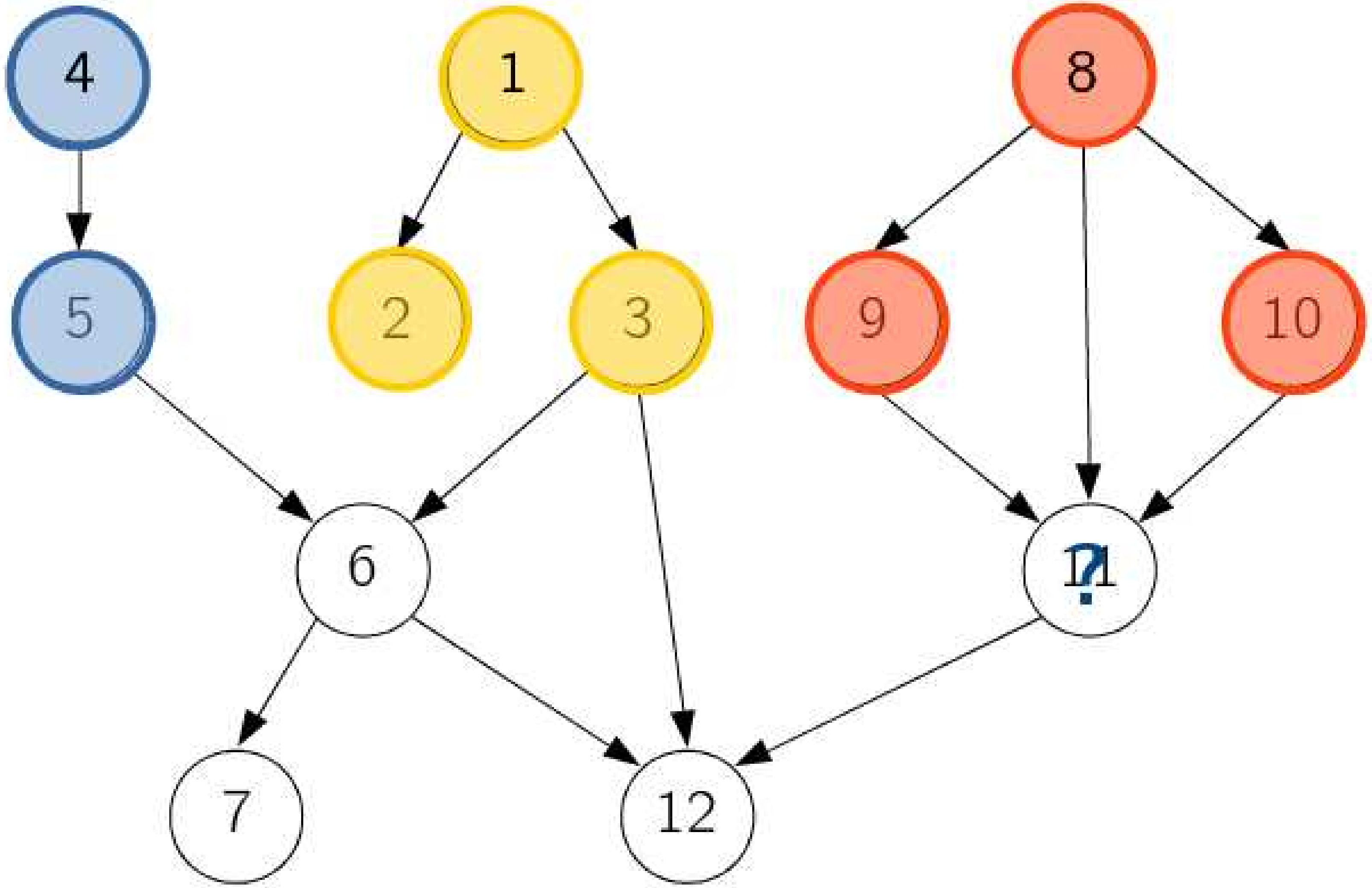


Imperial College London

At each step of the algorithm, we consider the vertices in order of their out-degree

Local memory size = 3

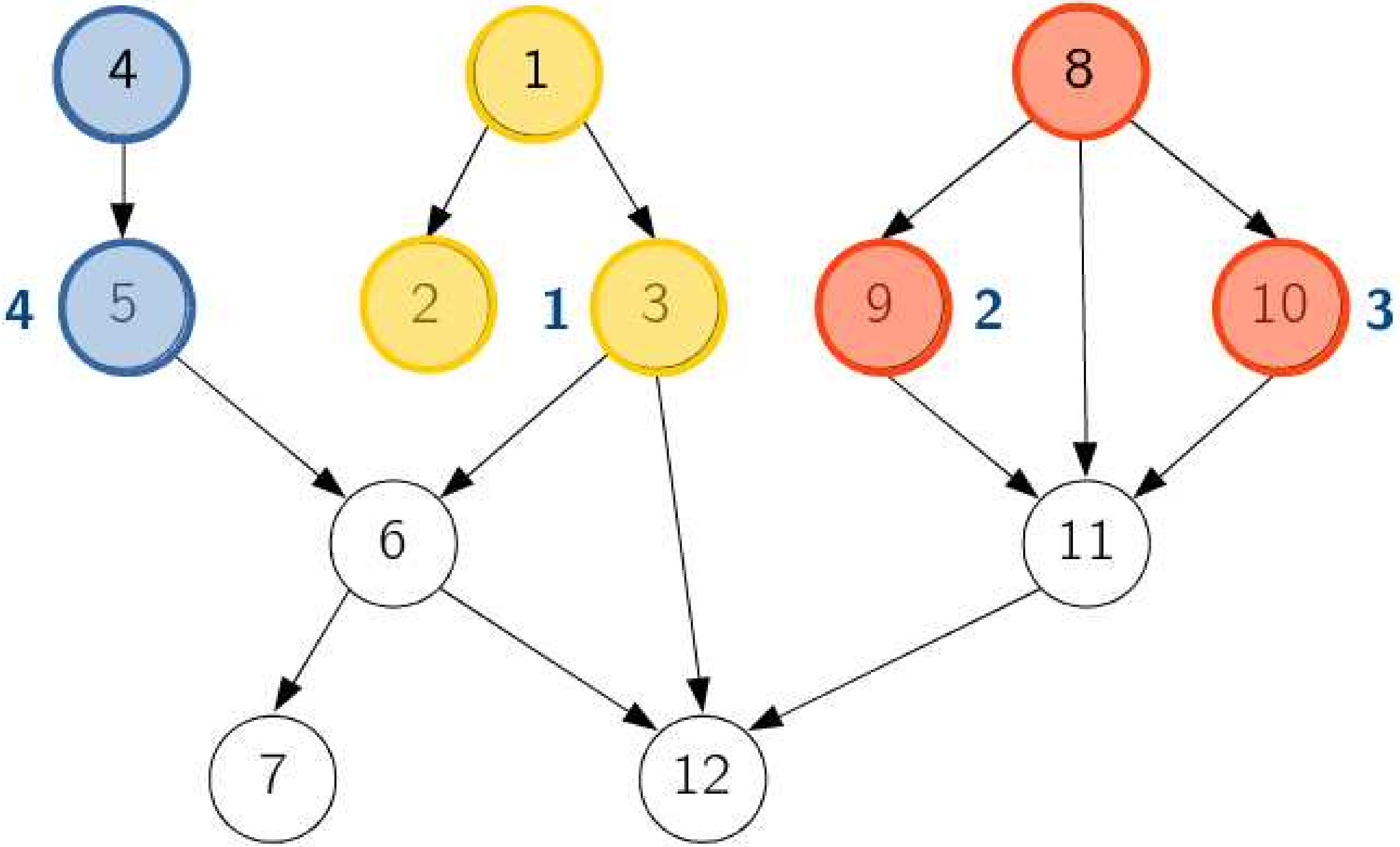
Priority rule: 1 2 3 4



At each step of the algorithm, we consider the vertices in order of their out-degree

Local memory size = 3

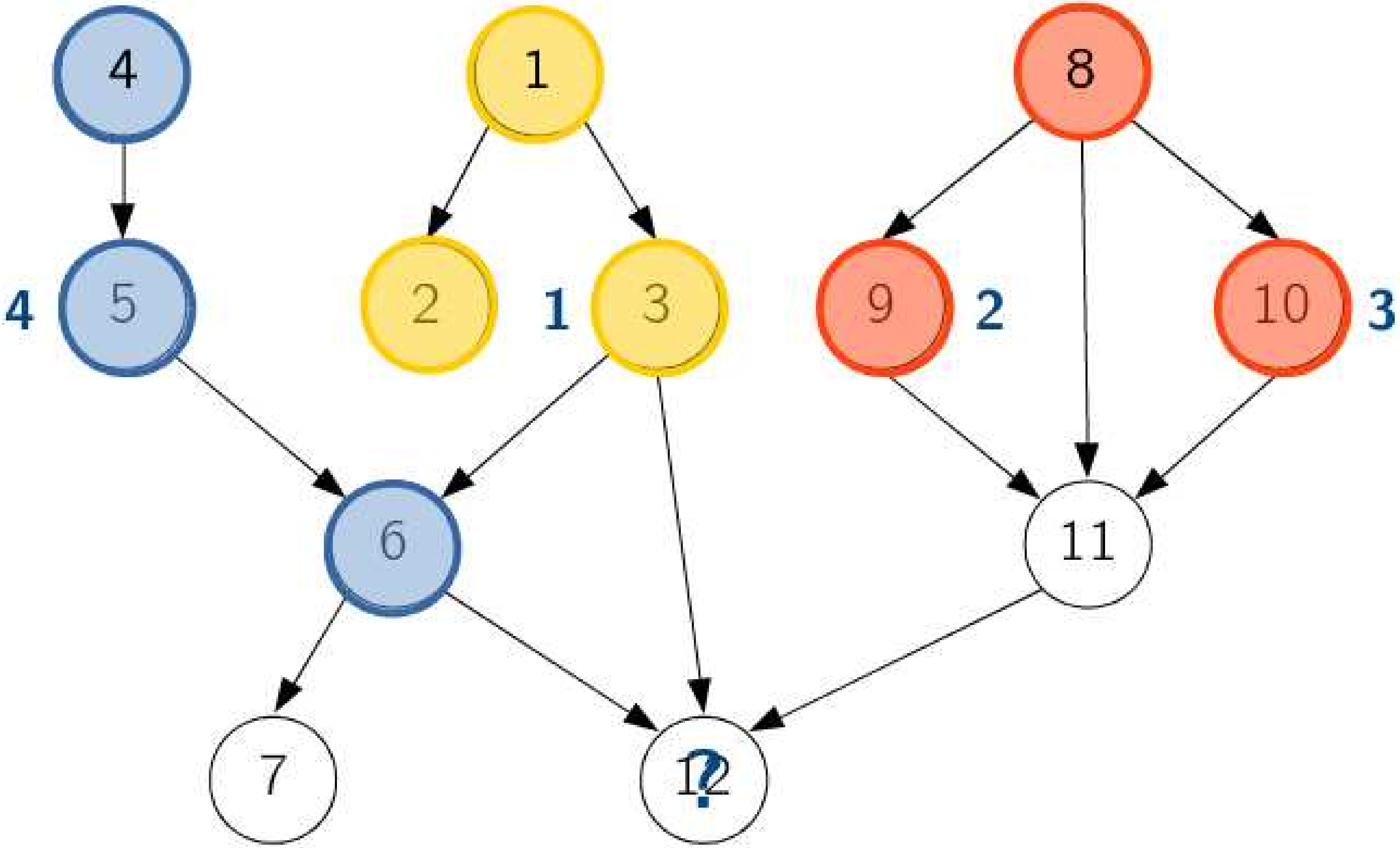
Priority rule: 🟥 1 🟨 2 🟦 3 🟩 4



At each step of the algorithm, we consider the vertices in order of their out-degree

Local memory size = 3

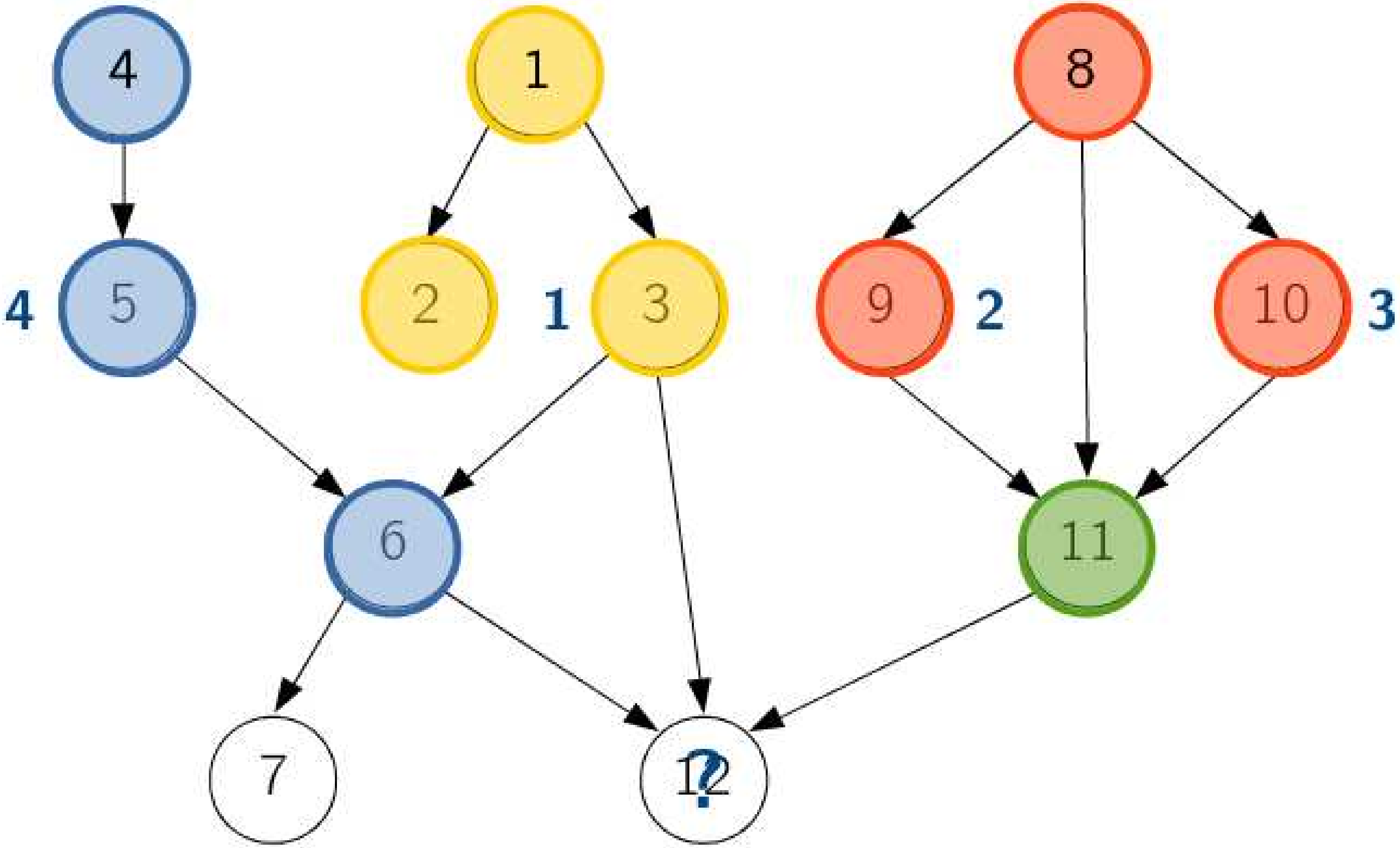
Priority rule: 🟥 1 🟨 2 🟦 3 🟩 4



At each step of the algorithm, we consider the vertices in order of their out-degree

Local memory size = 3

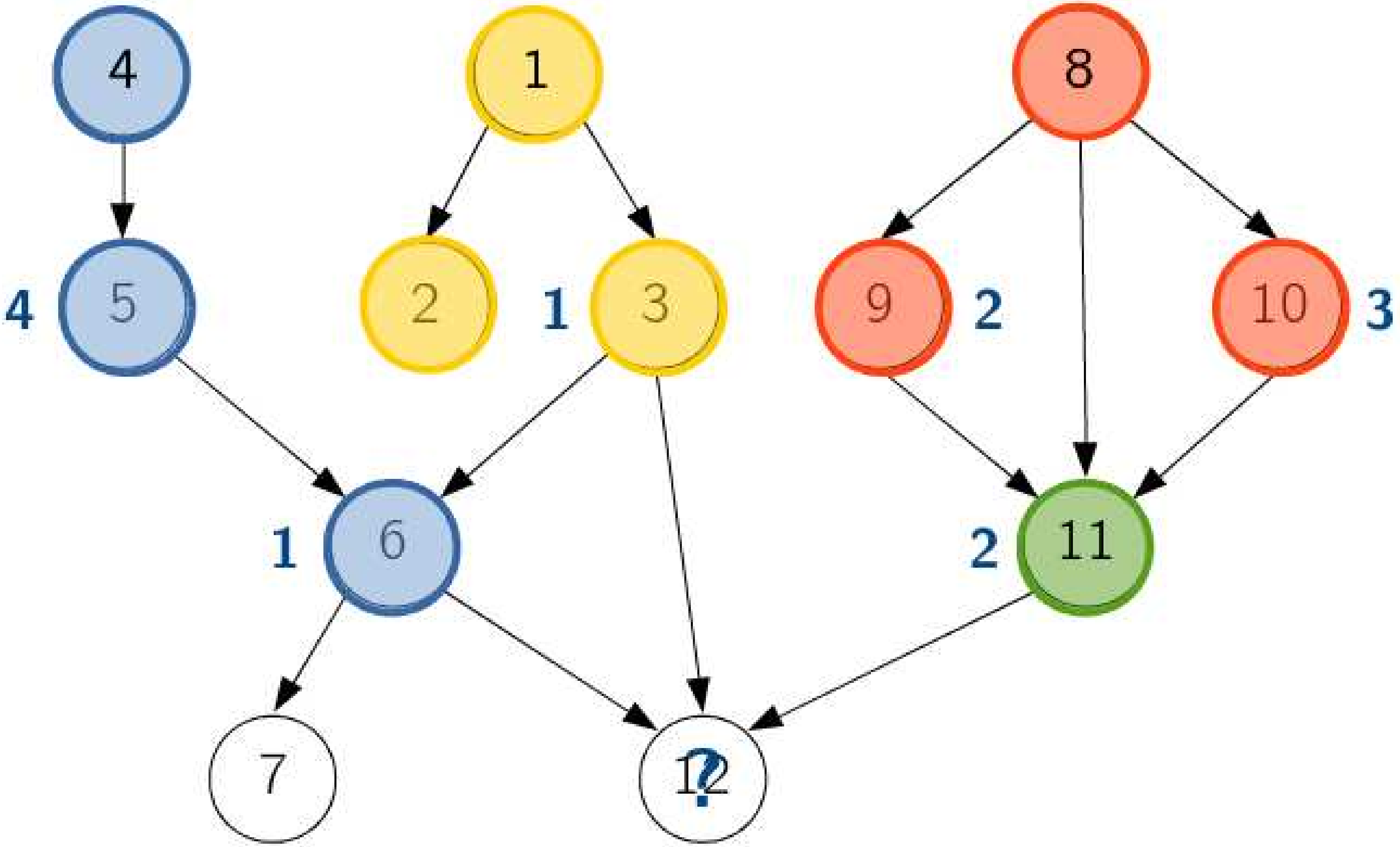
Priority rule: 🟥 1 🟨 2 🟦 3 🟩 4



At each step of the algorithm, we consider the vertices in order of their out-degree

Local memory size = 3

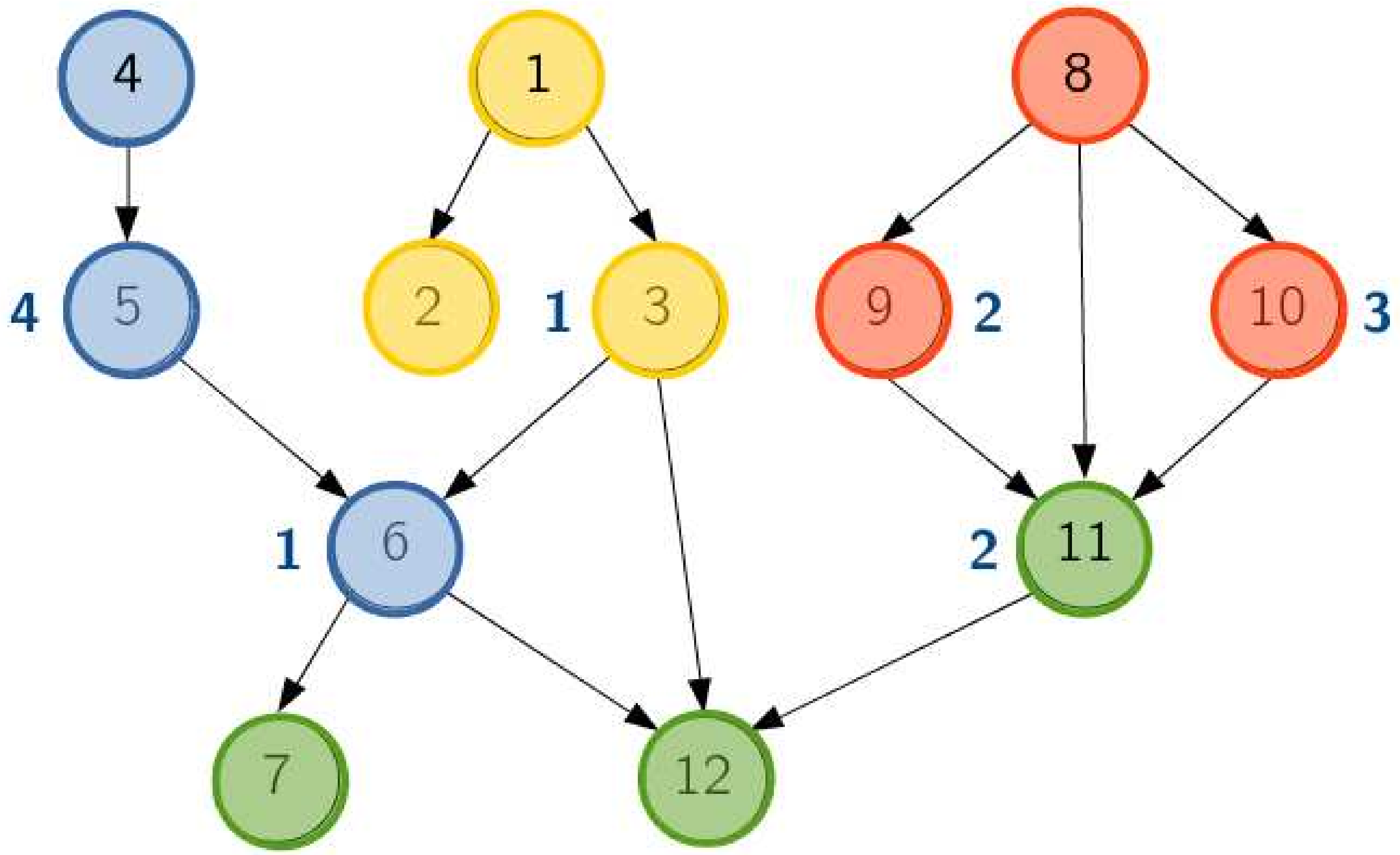
Priority rule: 🟥 1 🟨 2 🟦 3 🟩 4



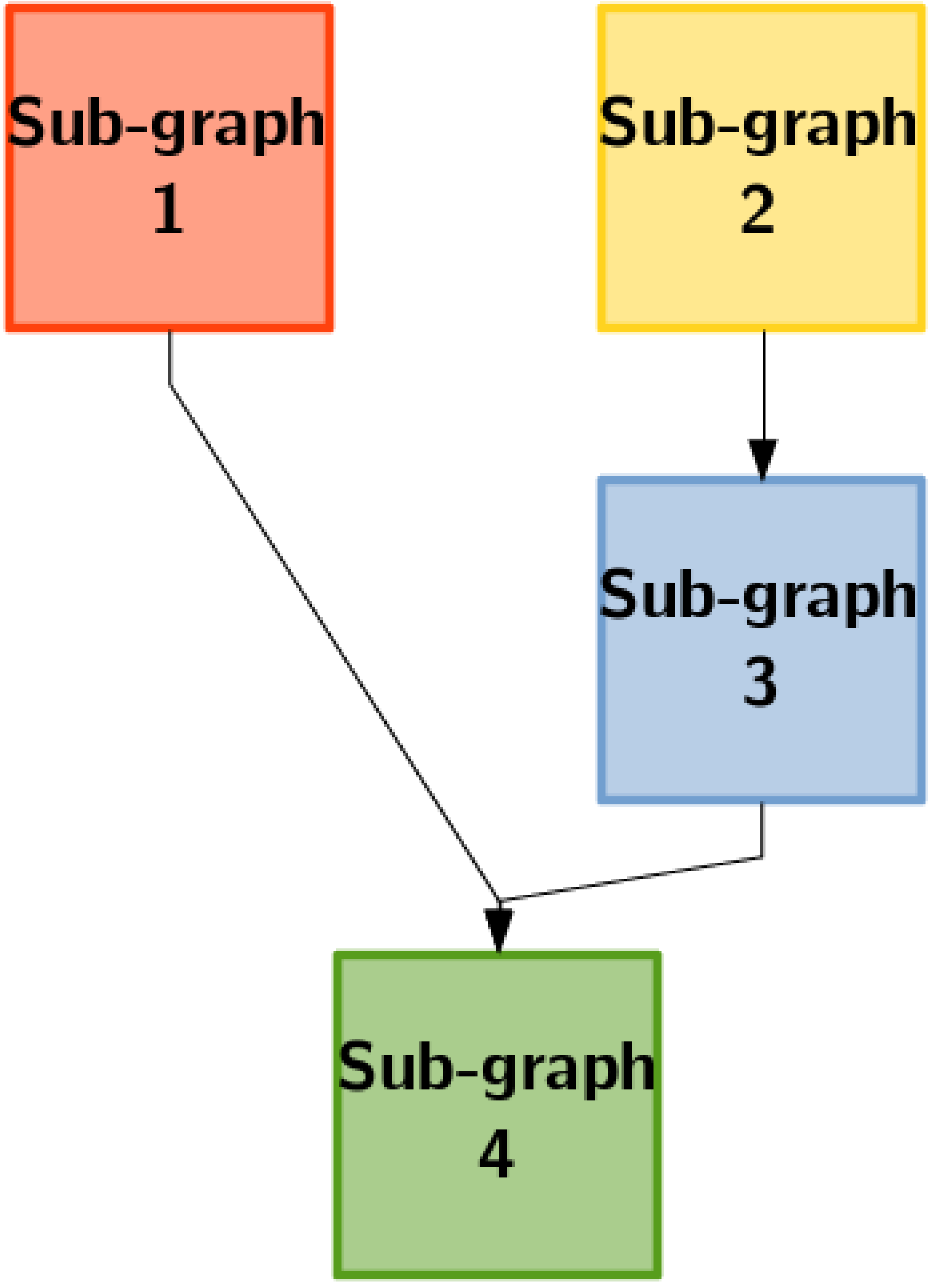
At each step of the algorithm, we consider the vertices in order of their out-degree

Local memory size = 3

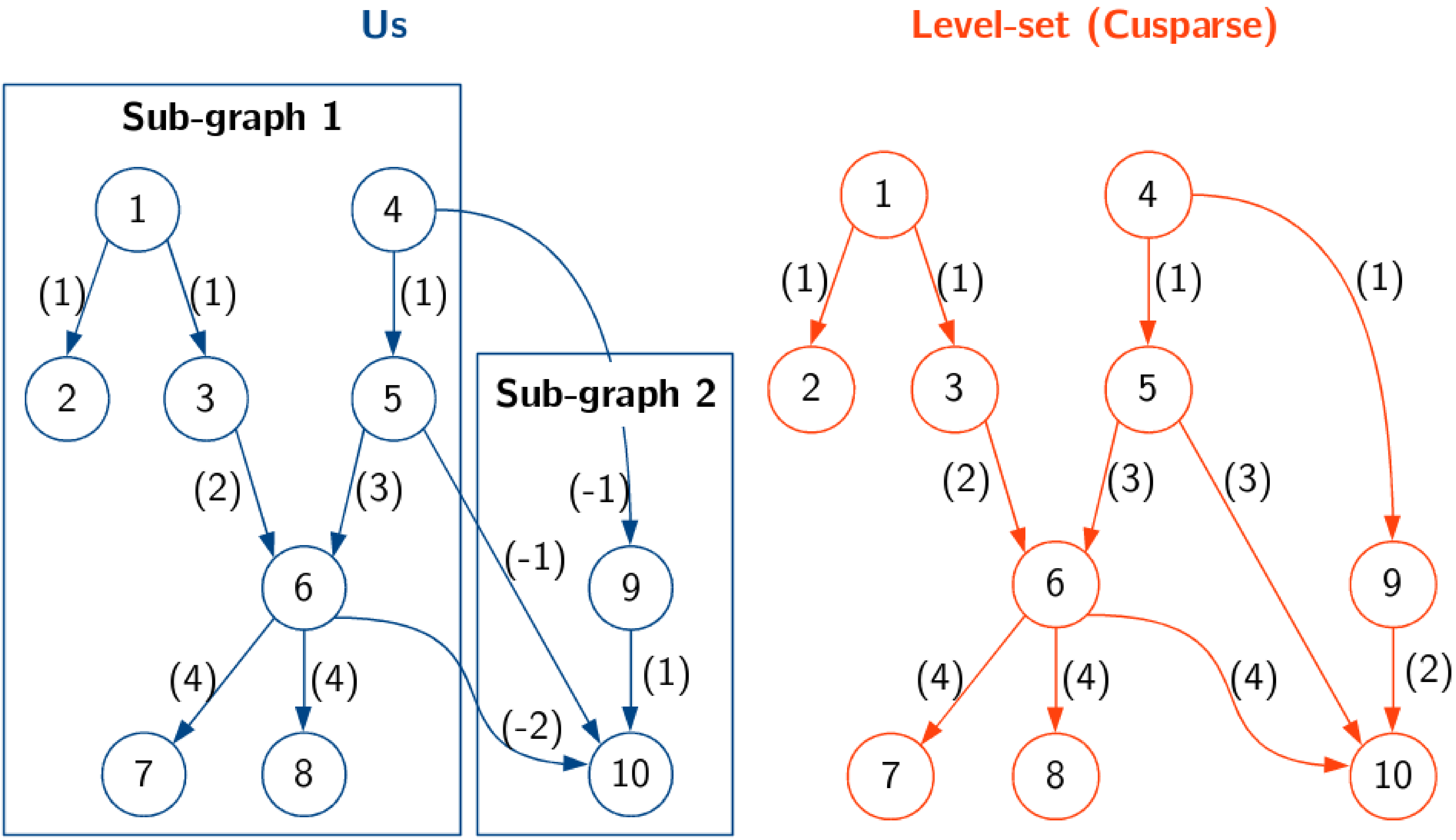
Priority rule: 1 2 3 4



At each step of the algorithm, we consider the vertices in order of their out-degree

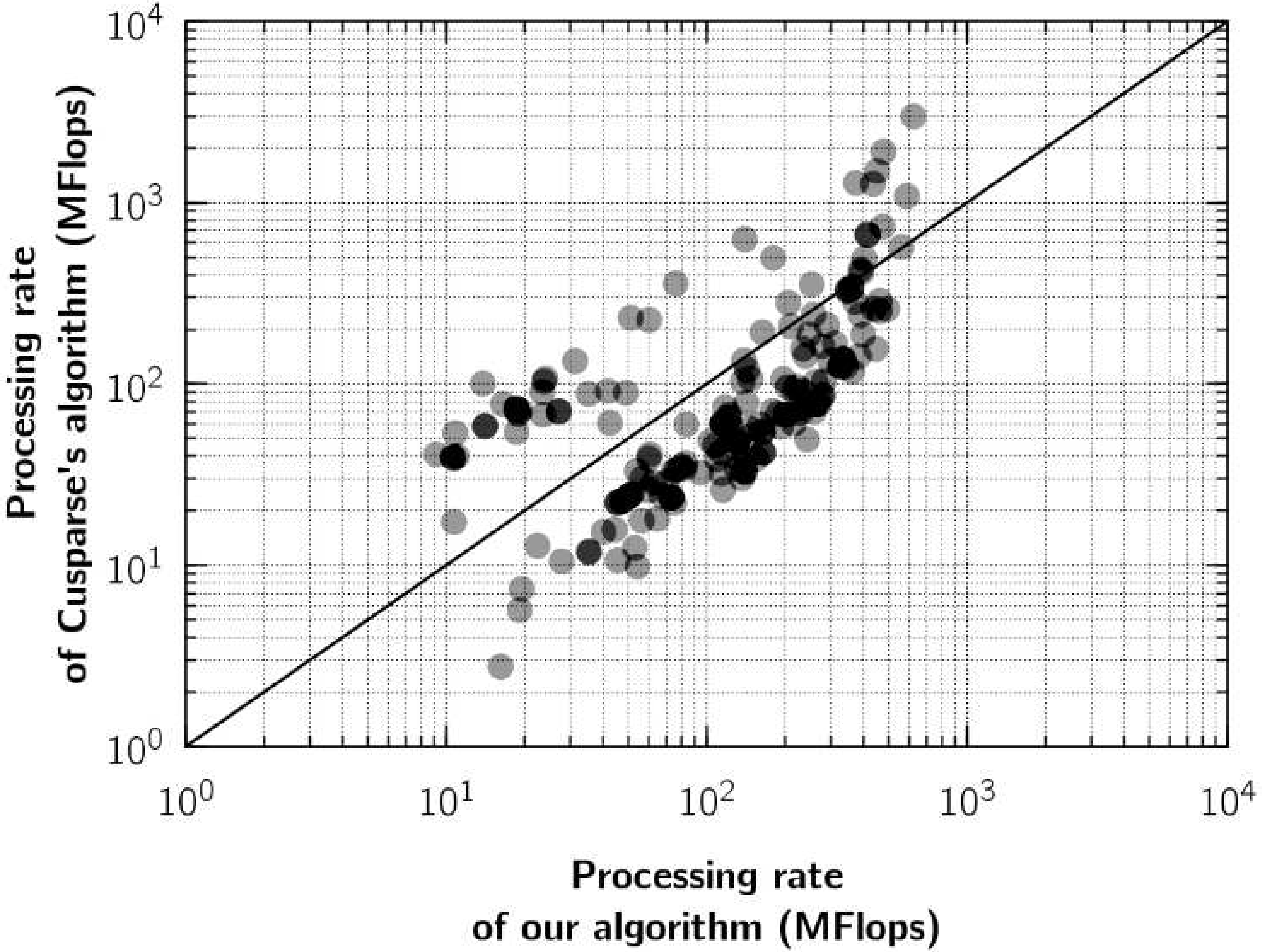


The most used algorithm does not perform any partitioning because it aims to use the whole GPU

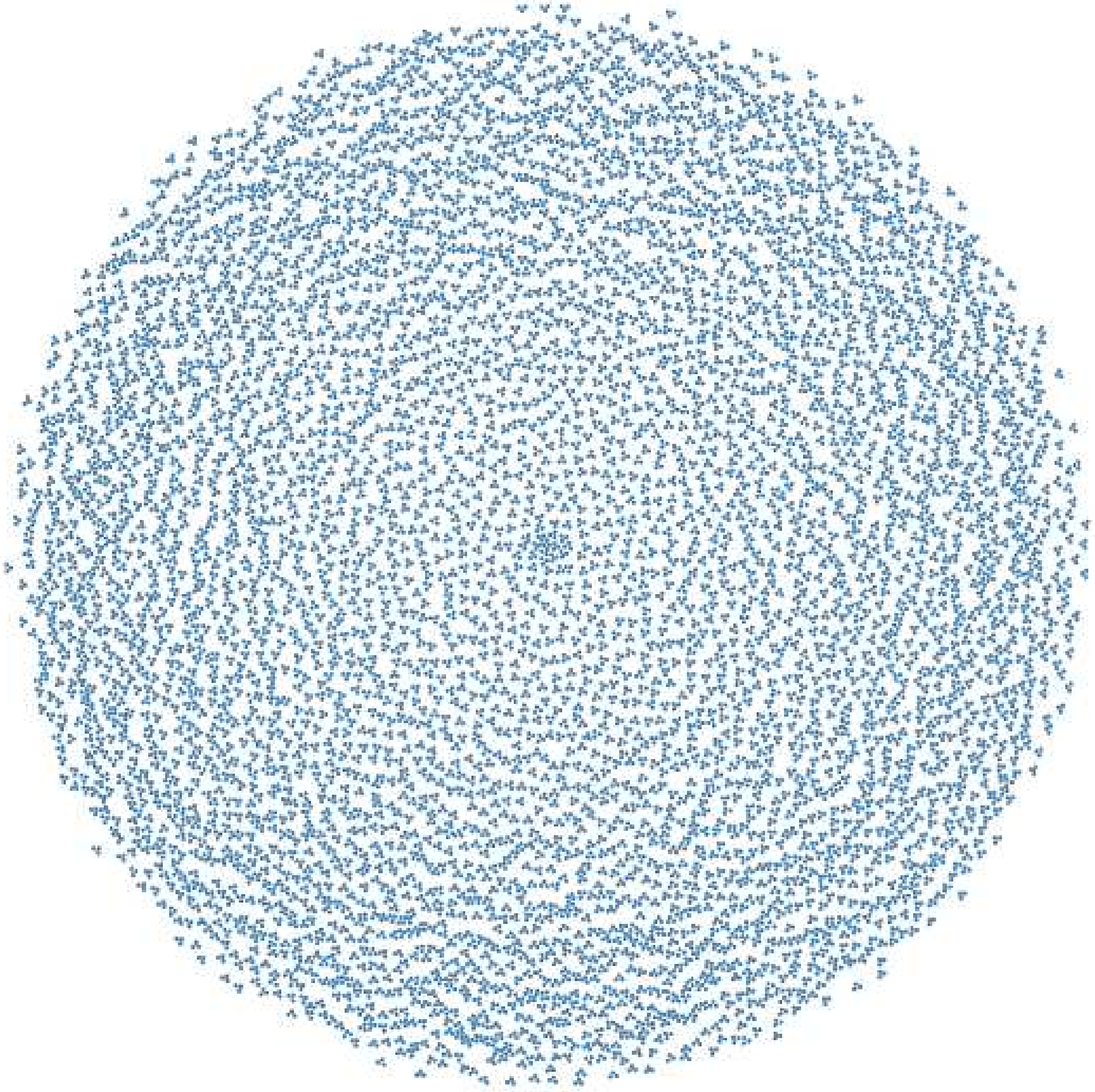


Imperial College London

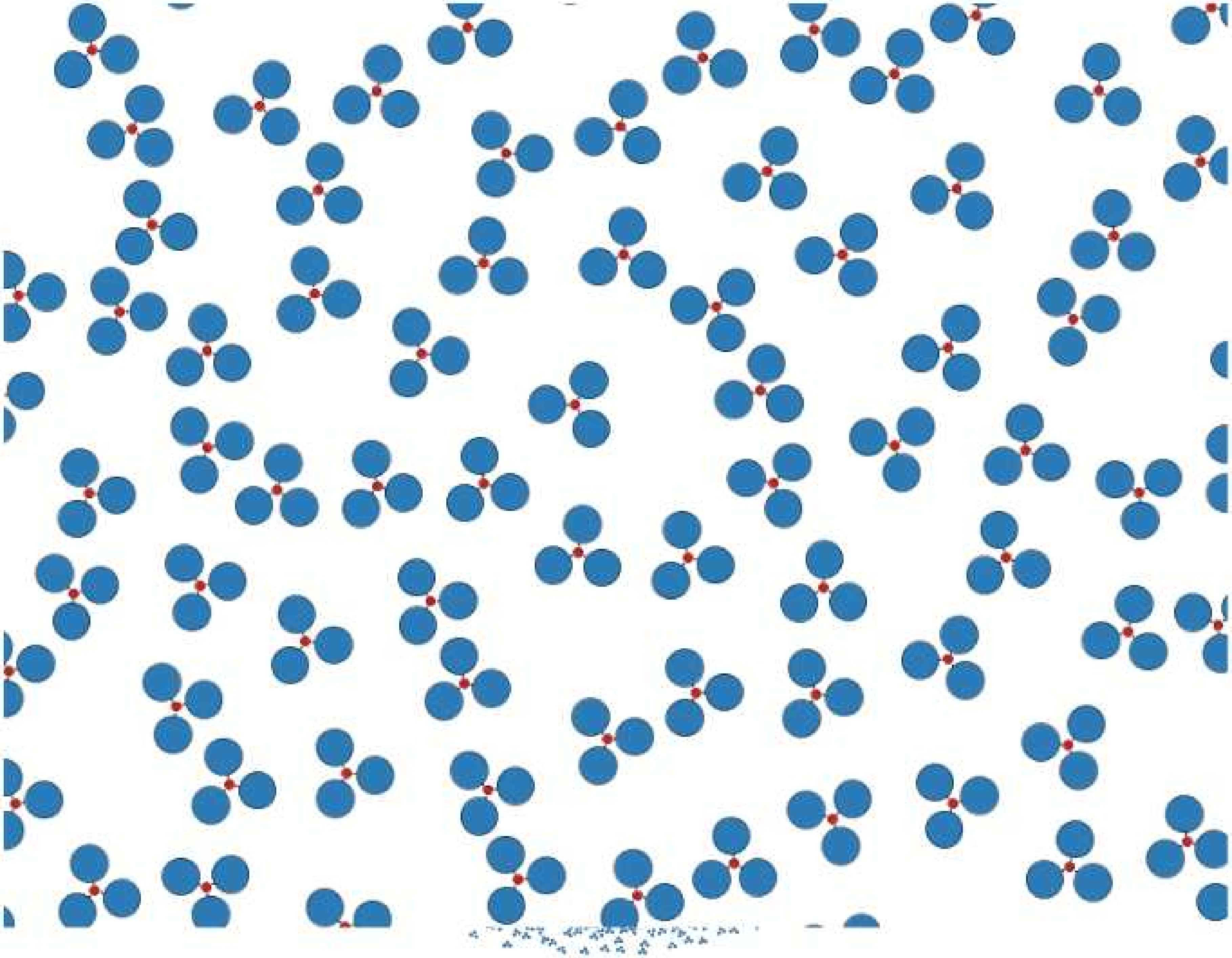
We achieved a maximum speedup of 6x against Cuda's library and a minimum speedup of 0.1x



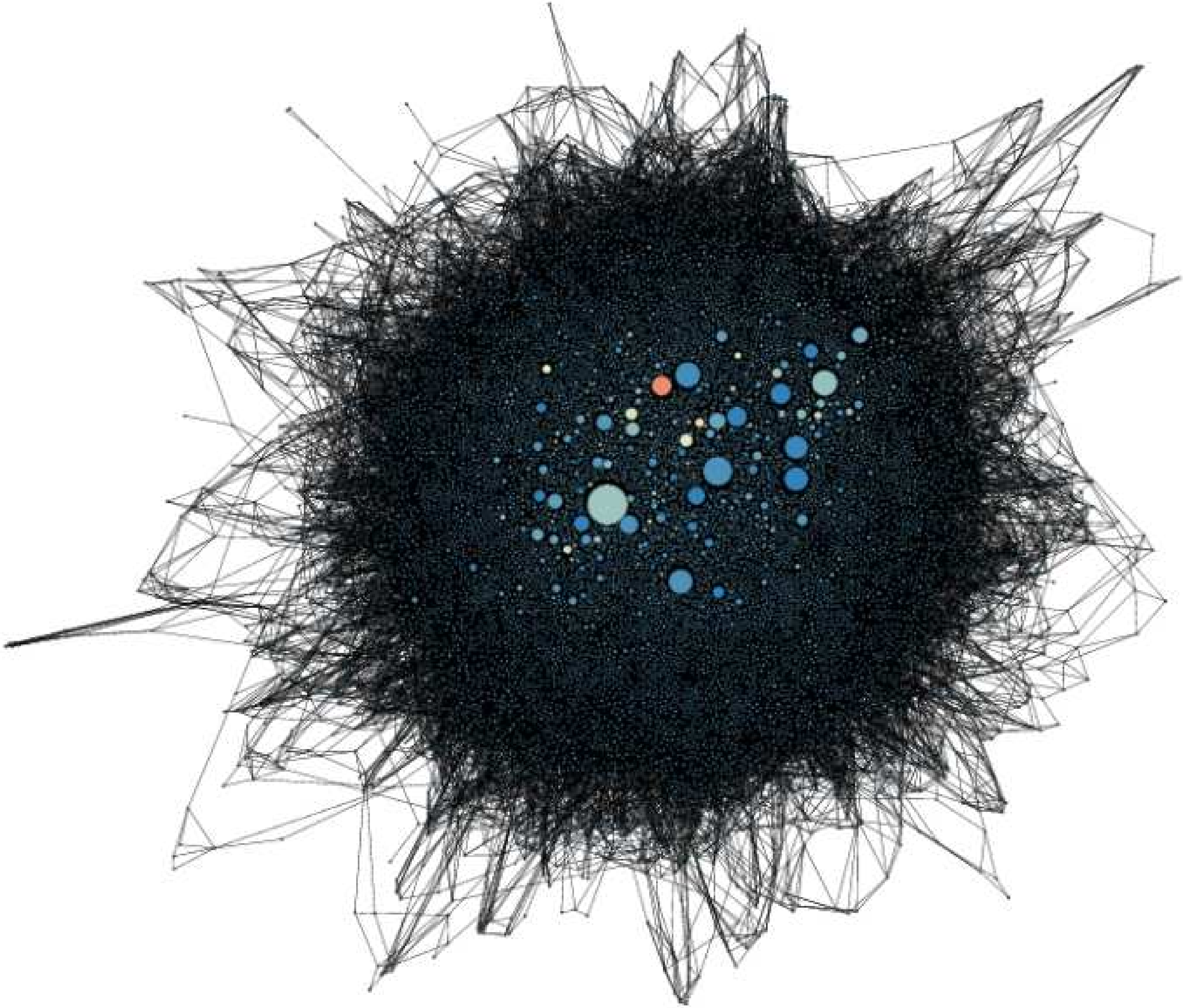
In the worst case, there was too little data-level parallelism that we could extract



In the worst case, there was too little data-level parallelism that we could extract

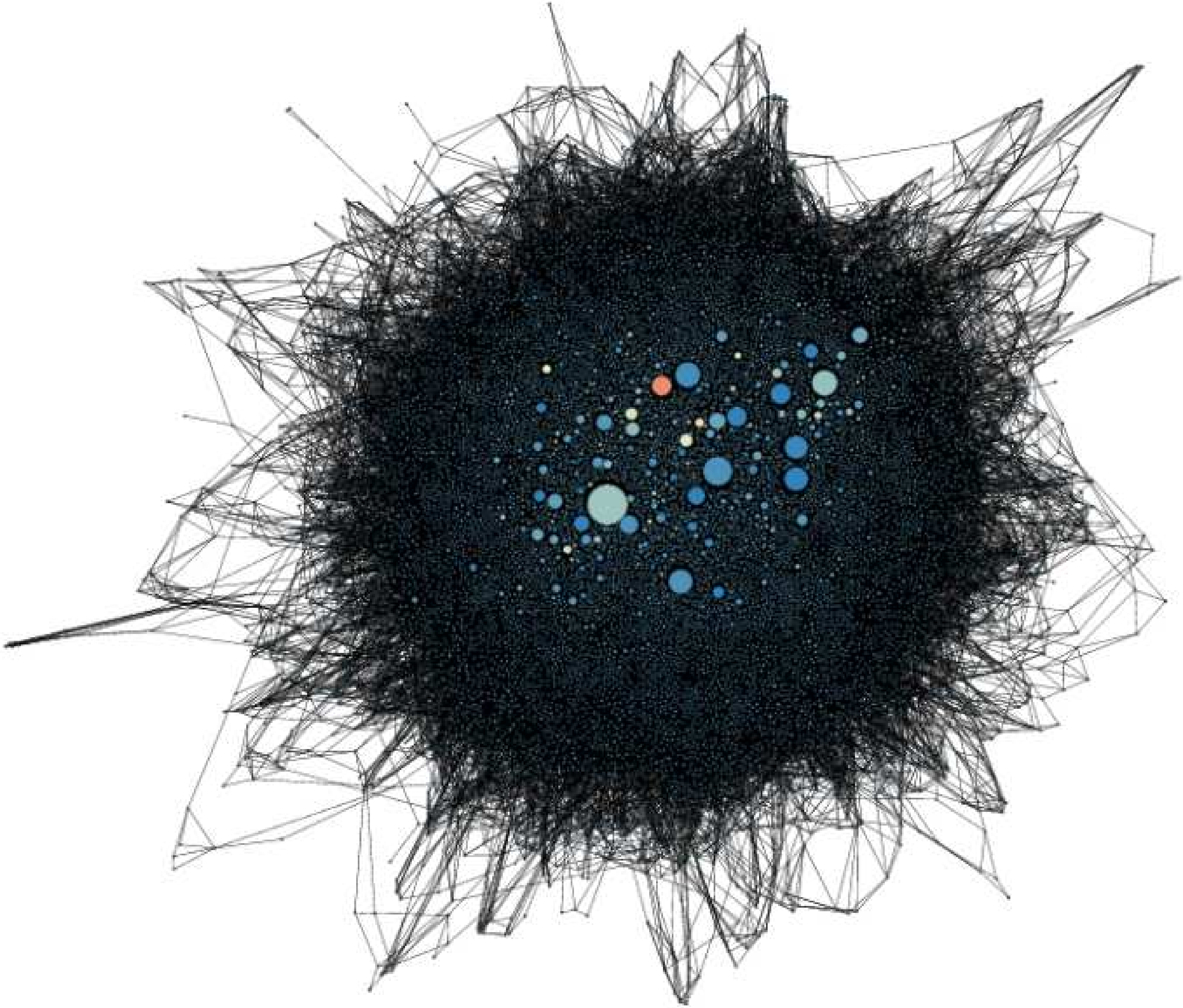


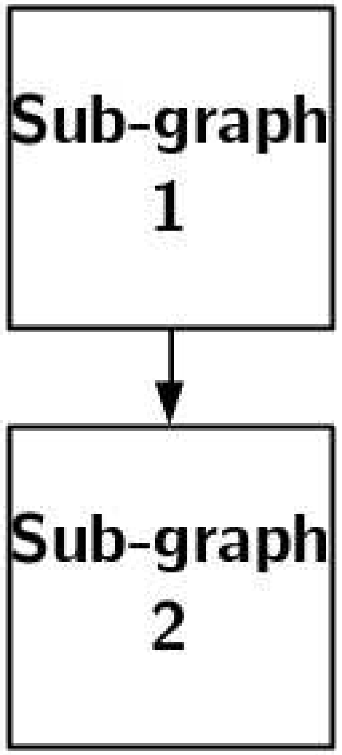
In the best case, we used only one compute unit instead of four, but we were 6 times faster than Cusparse



Imperial College London

In the best case, we used only one compute unit instead of four, but we were 6 times faster than Cusparse

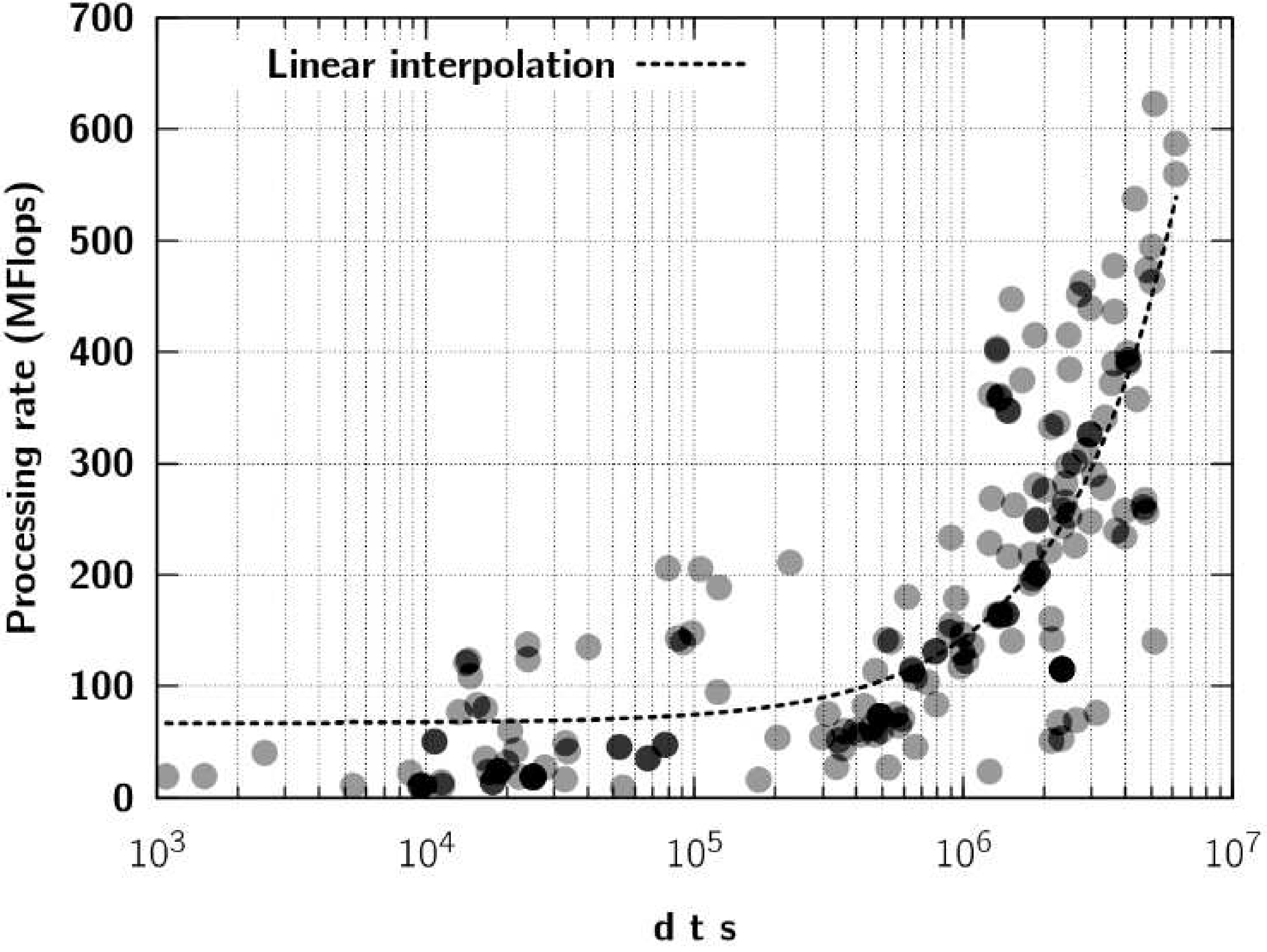




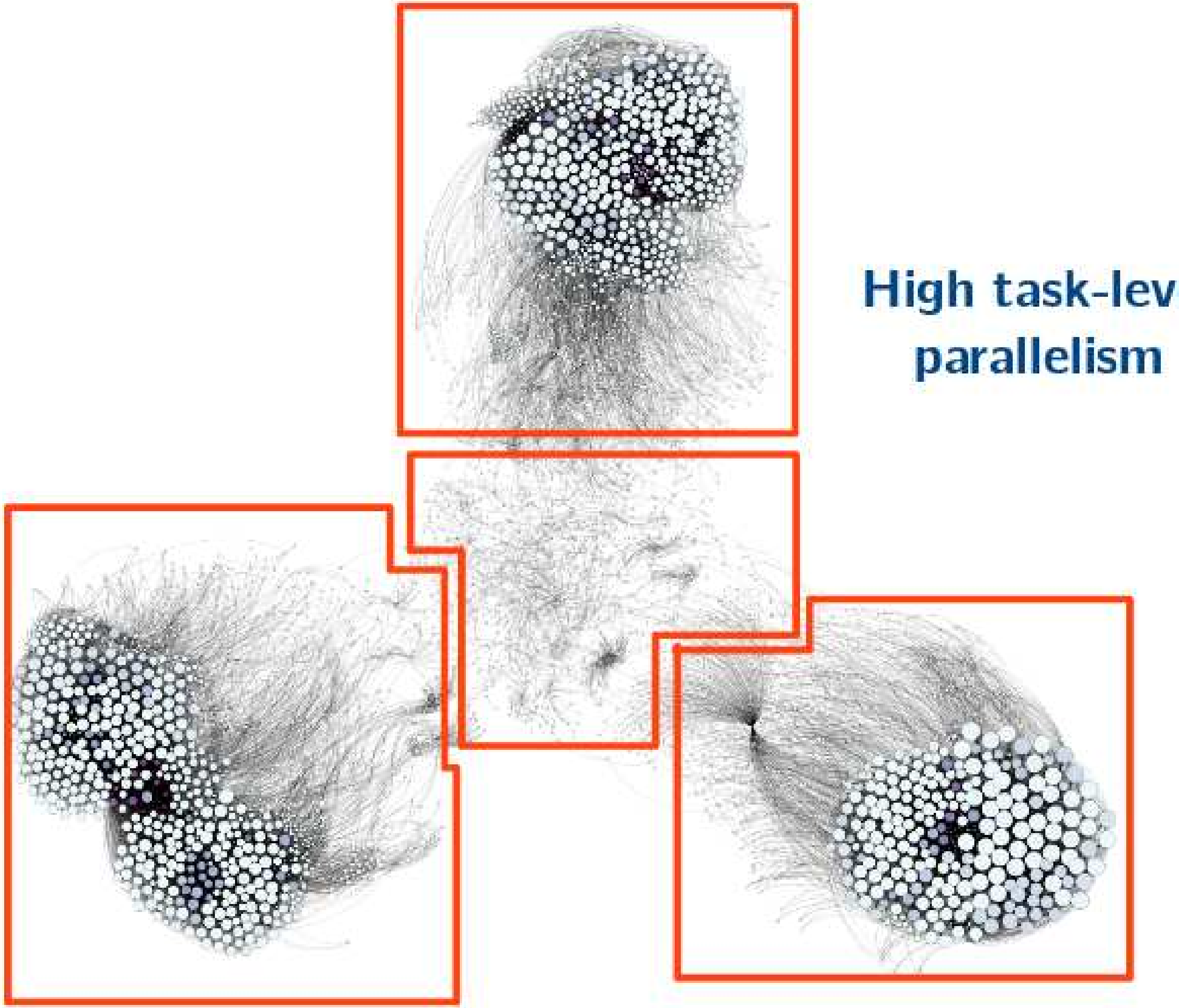
Sub-graph 1

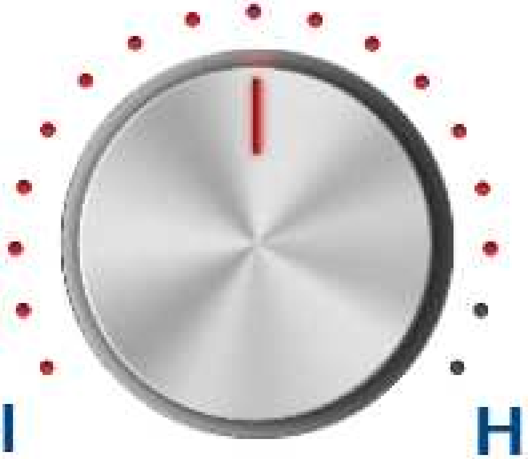
Sub-graph 2

The performance of our algorithm is mainly given by data-level parallelism, task-level parallelism and graph size



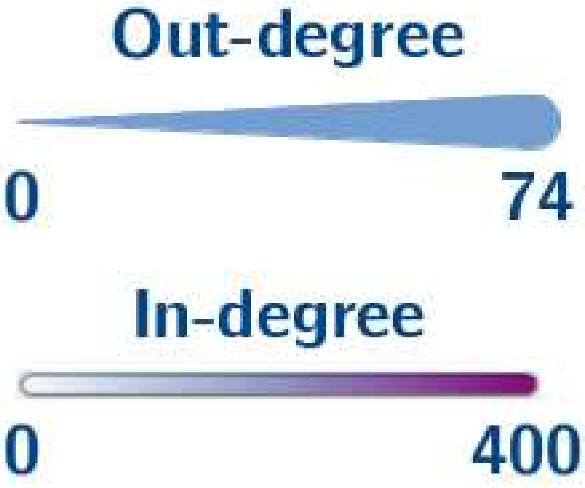
By trading task-level parallelism and data-level parallelism, our algorithm can improve data locality and performance





High task-level parallelism

High data-level parallelism


Out-degree
0
74
In-degree
0
400