INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING
ICPP/2021/CHICAGO/USA
acm In-Cooperation
sighpc
AUGUST 9–12, 2021

Communication Avoiding All-Pairs Shortest Paths Algorithm for Sparse Graphs

Lin Zhu, Qiang-sheng Hua*, and Hai Jin

Huazhong University of Science and Technology, China

Outline

- Background
- Several Algorithmic Techniques
- Our Method
- Proof of Lower Bound
- Summary

The all-pairs shortest-paths problem

An undirected weighted graph $G = \{V, E, W\}$

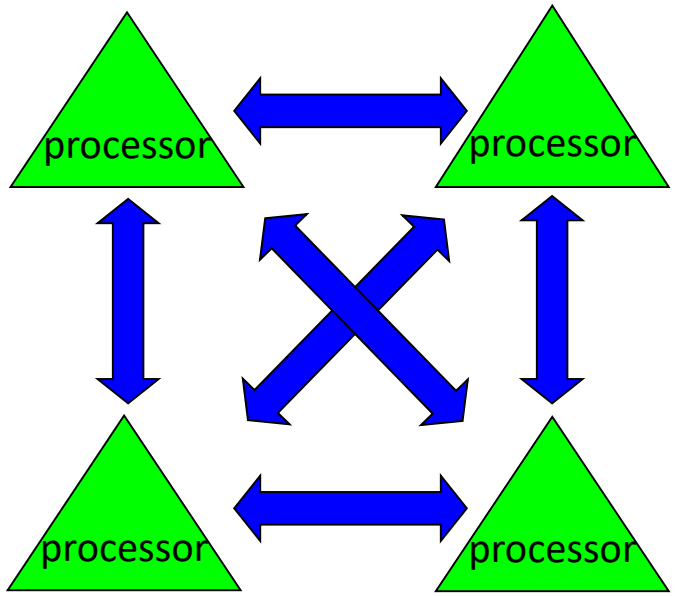
- vertex set V containing $n = |V|$ vertices
- edge set E with $m = |E|$ edges
- weights W

The all-pairs shortest path problem computes the length of the shortest paths between every pair of vertices in the graph G.

The distributed-memory cost model

We quantify interprocessor bandwidth (the number of words) and latency (the number of messages) costs of a parallelization via a network model

- the architecture is homogeneous
- a processor can only send/receive a message to/from one other processor at a time
- there is a link between each processor pair (all-to-all network)



Previous work

Algorithm	*Research problem*	*Applicable graph*	*Model*
2D-DC APSP Edgar Solomonik et al. *IPDPS 2013*	*APSP*	*dense graphs*	*distributed-memory model*
SuperFW Piyush Sao et al. *PPoPP 2020*	*APSP*	*dense/sparse graphs*	*share-memory model*
dSparseLU3D Piyush Sao et al. *IPDPS 2018*	*LU factorization*	*sparse matrices*	*distributed-memory model*

However, there are few studies focusing on efficient APSP algorithms for sparse graphs in distributed memory system

Goals

- To design an APSP algorithm with minimum communication cost for sparse graphs

- To give the lower bound of bandwidth cost and latency cost

Goals

- To design an APSP algorithm with minimum communication cost for sparse graphs
  - Bandwidth cost: $O(\frac{n^2 log^2 P}{P} + |S|^2 log^2 P)$
  - Latency cost: $O(log^2 P)$
- To give the lower bound of bandwidth cost and latency cost
  - Bandwidth lower bound: $\Omega(\frac{n^2}{P} + |S|^2)$
  - Latency lower bound: $\Omega(log^2 P)$

Floyd-Warshall algorithm (FW)

Floyd-Warshall algorithm

At each iteration k, the distance matrix A is updated

$$A(i,j) = A(i,j) \oplus \mathrm{A(i,k)} \otimes \mathrm{A(k,j)}$$

$$x \oplus \mathrm{y} = \min\{\mathrm{x},\mathrm{y}\},\ x \otimes \mathrm{y} = \mathrm{x} + \mathrm{y}$$

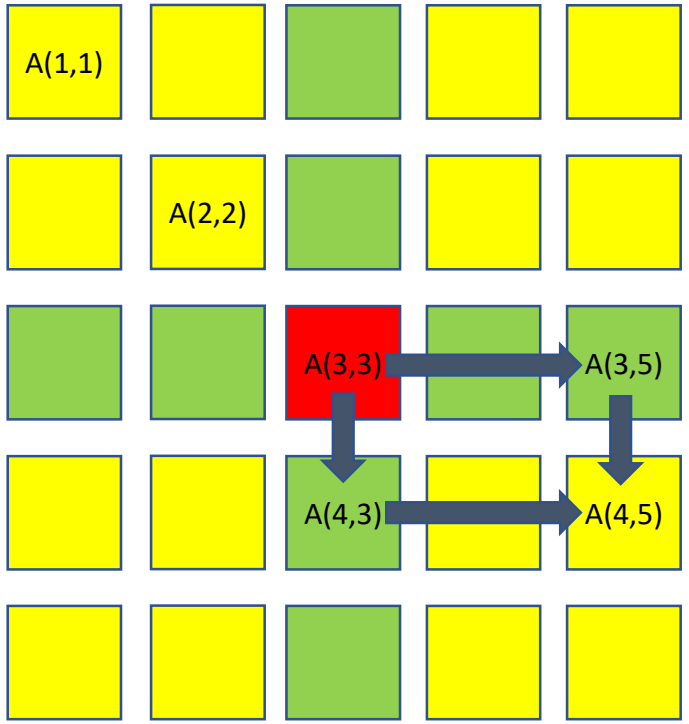
Blocked Floyd-Warshall algorithm

Divide A into $\frac{n}{b} \times \frac{n}{b}$ blocks, each of size $b \times b$

At each iteration k

- $A(k,k) = FW(A(k,k))$
- $A(:,k) = A(:,k) \oplus \mathrm{A(:,k)} \otimes \mathrm{A(k,k)}$
- $A(k,:) = A(k,:) \oplus \mathrm{A(k,k)} \otimes \mathrm{A(k,:)}$
- $A(i,j) = A(i,j) \oplus \mathrm{A(i,k)} \otimes \mathrm{A(k,j)}$

For example: k=3



All blocks of A need to be updated

How to avoid the update of certain blocks for sparse graphs ?

Outline

- Background
- Several Algorithmic Techniques
- Our Method
- Proof of Lower Bound
- Summary

Several Algorithmic Techniques

For $k = 3$, if $Dist(4,3) = \infty$, then $Dist(4,:) = \min\{Dist(4,:), Dist(4,3) + Dist(3,:)\} = Dist(4,:)$

The update of $Dist(4,:)$ can be avoided

Similar, for $k = 3$, if all entries in block $A(4,3)$ is ∞, then $A(4,:) = A(4,:) \oplus A(4,3) \otimes A(3,:)$

The update of $A(4,:)$ can be avoided

However, A is irregular and there may not be all infinite values in a block

Nested-Dissection Ordering (ND process)

ND process: reorder the adjacency matrix

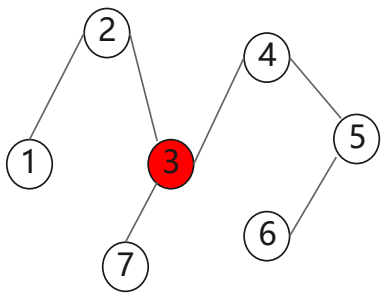
Find the vertex separator S, S partitions V into three disjoints sets, $V = V_1 \cup S \cup V_2$, and

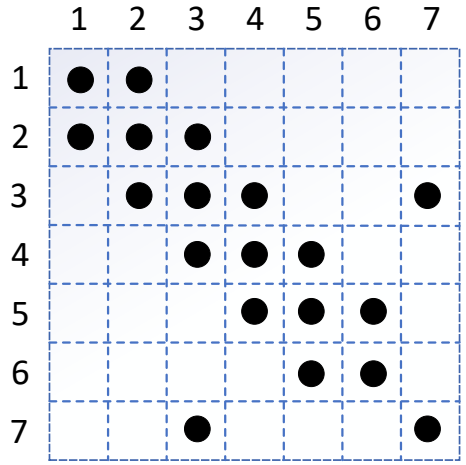
- No edges between V_1 and V_2
- $|V_1| \approx |V_2|$
- S is as small as possible

V_1 , V_2 and S are called supernodes

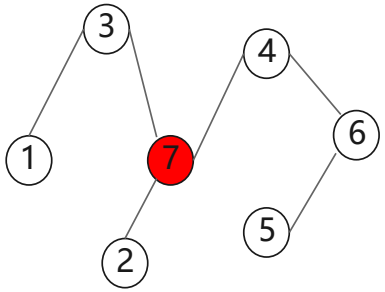
The vertices within V_1 and V_2 have consecutive indices; vertices in S have a higher index

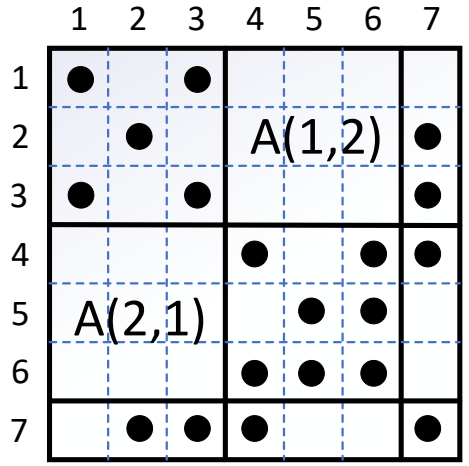
Initial





Reorder

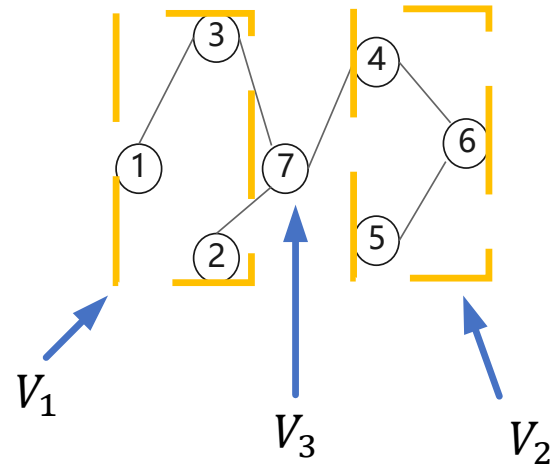




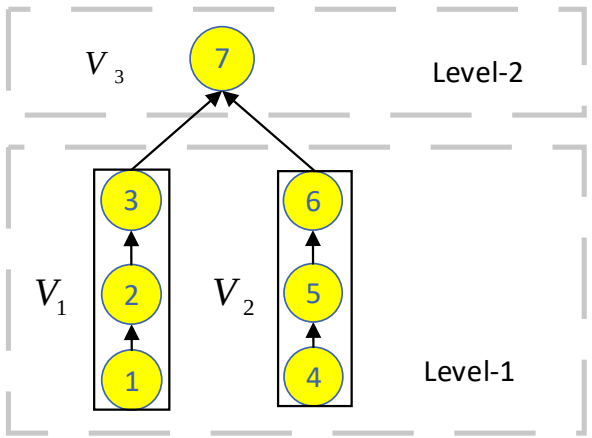
All entries in block A(1,2) and A(2,1) are ∞

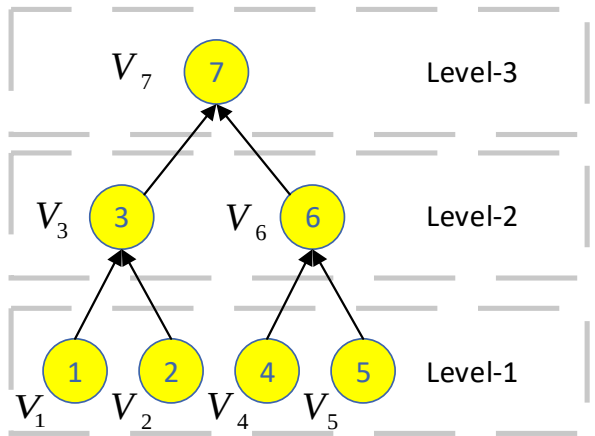
Elimination tree (eTree)

By computing the separator of the graph G, we can get a two-level elimination tree (eTree)



By recursively computing the separators of V_1 and V_2 , we can obtain a multi-level eTree





The eTree can guide parallelism.

- The elimination of supernodes in the same level is independent.

Challenge

The computational cost of the FW algorithm is $O(n^3)$. Using ND process and eTree techniques, the computational cost can be reduced to $O(n^2 S)$

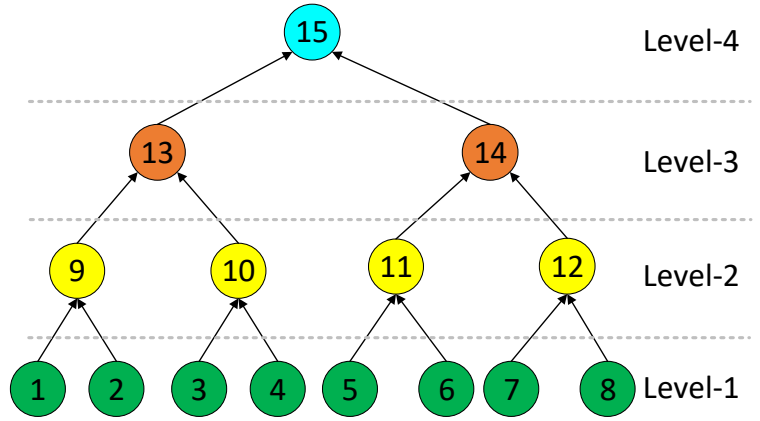
How to reduce communication cost in the distributed memory model?

Outline

- Background
- Several Algorithmic Techniques
- Our Method
- Proof of Lower Bound
- Summary

Symbol description

We map the supernodal block sparse matrix A to a $\sqrt{P} \times \sqrt{P}$ grid in a block layout



Symbol description:

$A(k)$: the set of all ancestors of supernode k

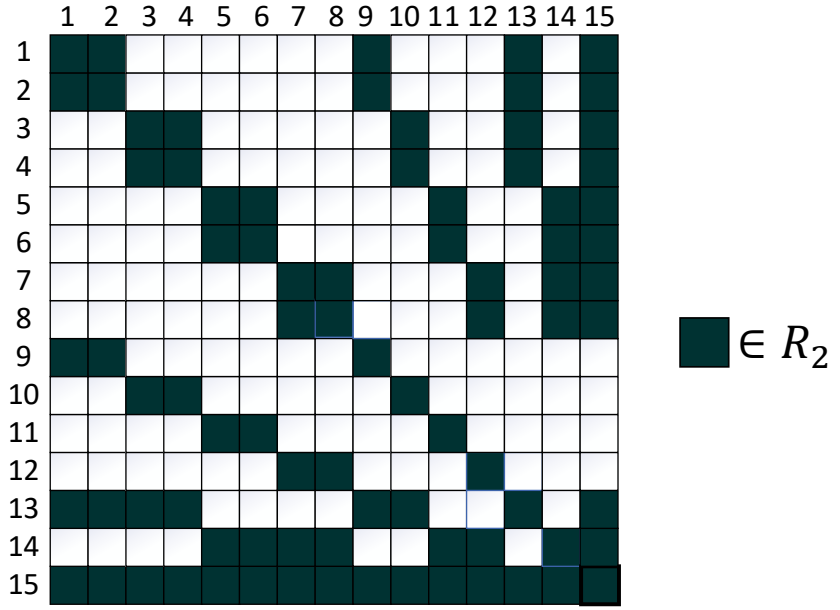
$D(k)$: the set of all descendants of supernode k

$C(k)$: the set of all cousins of supernode k

Q_l: the collection of the l-th level supernodes

R_l: the updated region of A during the elimination of the l-th level supernodes

$$R_l = \cup_{k \in Q_l} (k \cup A(k) \cup D(k), k \cup A(k) \cup D(k))$$



Our Method

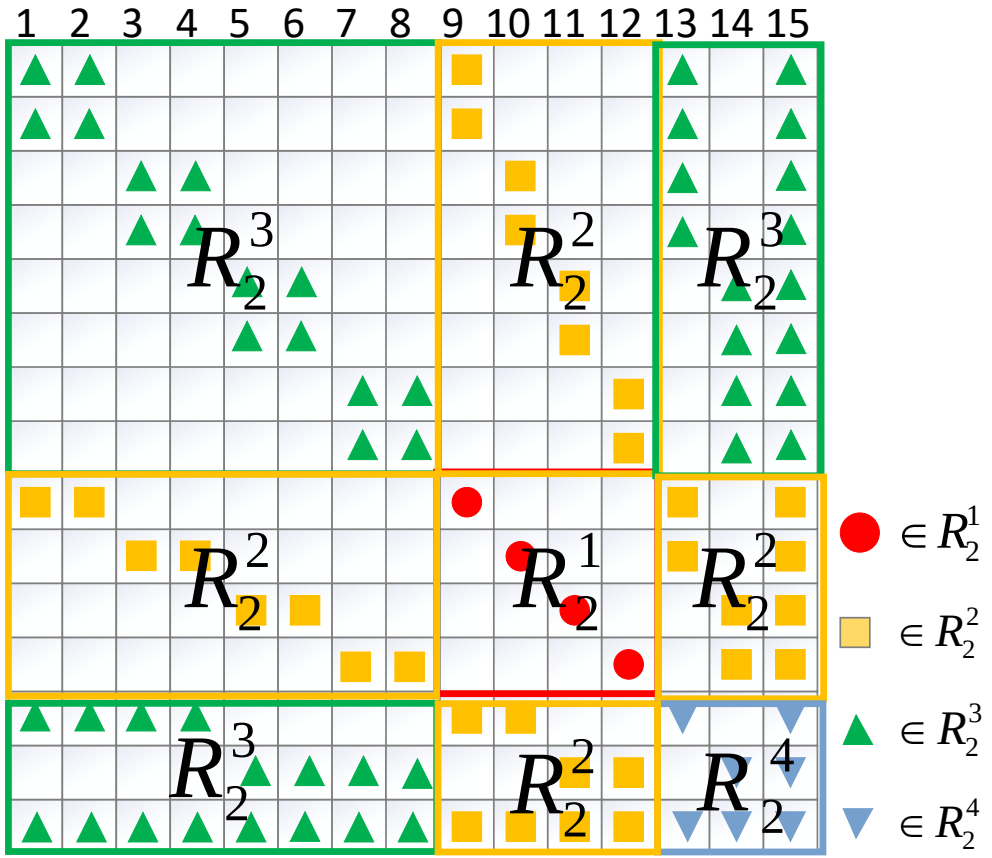
Divide R_l into four subsets:

$R_l^1 = \bigcup_{k \in Q_l} (k, k)$
$R_l^2 = \bigcup_{k \in Q_l} (A(k) \cup D(k), k) \cup (k, A(k) \cup D(k))$
$R_l^3 = \bigcup_{k \in Q_l} (A(k) \cup D(k), D(k)) \cup (D(k), A(k))$
$R_l^4 = \bigcup_{k \in Q_l} (A(k), A(k))$

For each $(i, j) \in R_l$, the update of $A(i, j)$ is

$$A(i,j) = A(i,j) \oplus \sum_{k}^{\oplus} A(i,k) \otimes A(k,j)$$

where $k \in (A(i) \cup D(i)) \cap (A(j) \cup D(j)) \cap Q_l$



The update of R_l^1, R_l^2 and R_l^3

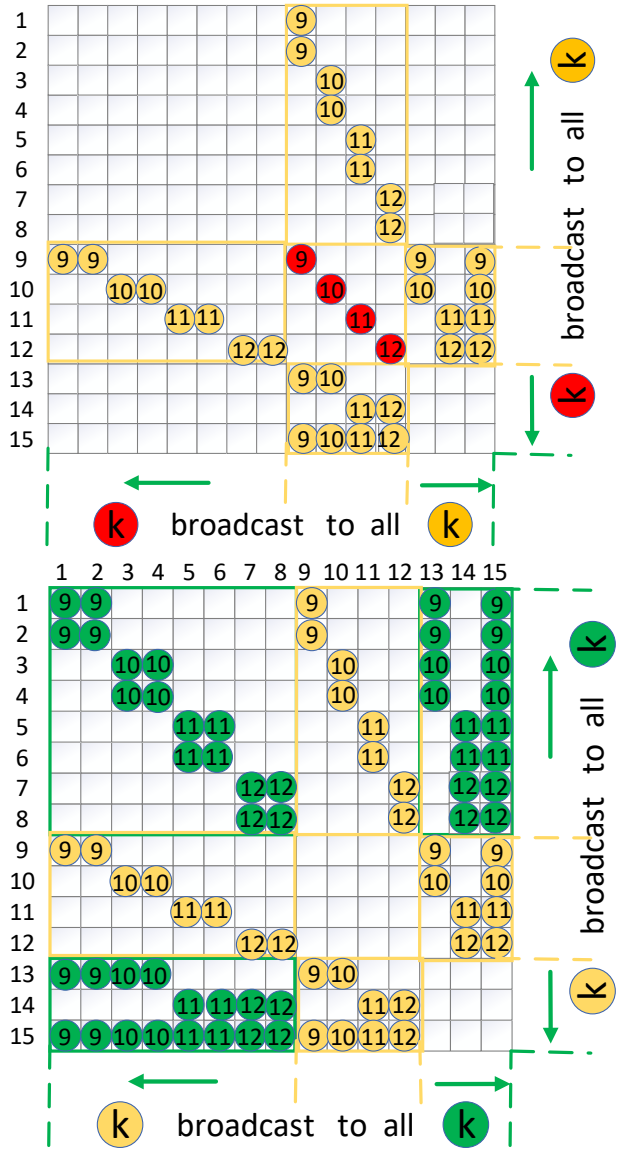
The update of R_l^1: P_{kk} performs local updates

The update of R_l^2:

- P_{kk} broadcast to all $P_{\mathrm{i}k}$, where $\mathrm{i} \in A(k) \cup D(k)$
- P_{kk} broadcast to all $P_{k\mathrm{j}}$, where $\mathrm{j} \in A(k) \cup D(k)$

The update of R_l^3:

- For each $(i,k) \in R_l^2$, $P_{\mathrm{i}k}$ broadcast to all $P_{\mathrm{i}j}$, $\mathrm{j} \in A(k) \cup D(k)$
- For each $(k,j) \in R_l^2$, P_{kj} broadcast to all $P_{\mathrm{i}j}$, $\mathrm{i} \in A(k) \cup D(k)$



The update of R_l^4

The update of R_l^4:

If $\left|\left(A(i) \cup D(i)\right) \cap \left(A(j) \cup D(j)\right) \cap Q_l\right| = q$, then $A(i,j)$ needs to be updated q times, i.e.,

$$A(i,j) = A(i,j) \oplus A(i,1) \otimes \mathrm{A}(1,\mathrm{j}) \oplus A(i,2) \otimes \mathrm{A}(2,\mathrm{j}) \ldots \oplus A(i,q) \otimes \mathrm{A}(\mathrm{q},\mathrm{j})$$

A trivial strategy: $P_{i1}, P_{i2} \ldots, P_{iq}$ send local data to P_{ij} in sequential

- latency cost: $\Omega(q)$

Optimal strategy: $P_{i1}, P_{i2} \ldots, P_{iq}$ send local data to q different processors, each processor performs a computing unit and then reduce to $P(i,j)$.

- latency cost: $O(\log q)$

There are more than one block $A(i,j)$ in R_l^4 needs to be updated.

The update of R_l^4

In order to update all the blocks in R_l^4 with a maximum degree of parallelization, the optimal strategy is to allocate each computing unit that updates R_l^4 to a separate processor one-to-one

the number of computing units required to update R_l^4 is $O(P)$.

- by summing the number of units of each Aij

We get such an one-to-one mapping from the computing units for updating R_l^4 to the processors.

- for each $\mathrm{A(i,j)\ in\ R_l^4}$, each computing unit is all $\mathrm{A(i,k) \otimes A(k,j\ )}$, where $\mathrm{k \in Q_l \cap D(i) \cap D(j)}$.
- each computing unit can be assigned to a separate processor $\mathrm{P_{fg}}$, where $\mathrm{f = \sum_{b=h+a-c}^{h-1} 2^b + (a-l)}$, $\mathrm{g = k - \sum_{b=h-l+1}^{h-1} 2^b}$, $\mathrm{a \in \{l+1, l+2 \dots, h\}}$ and $\mathrm{c \in \{a, a+1 \dots, h\}}$.

Outline

- Background
- Several Algorithmic Techniques
- Our Method
- Proof of Lower Bound
- Summary

3NL computation model

The computation of matrix multiplication and APSP can be expressed in a three nested-loop (3NL) way. multiplying two $n \times n$ matrices: $C_{ij} = C_{ij} + A_{ik} \cdot B_{kj}$

Informally, the 3NL computation model is defined as follows:

- There are two non-trivial parameter-dependent functions f_{ij}, g_{ijk} such that $C_{ij} = f_{ij}(g_{ijk}(A_{ik}, B_{kj}))$
- The elements in A, B, and C are mapped to memory locations one by one

3NL computation model lower bound:

- bandwidth lower bounds $\Omega(\frac{\mathrm{F}}{\mathrm{P}\sqrt{\mathrm{M}}})$
- latency lower bounds $\Omega(\frac{\mathrm{F}}{\mathrm{PM}^{2/3}})$

F: the number of computation operations
M: per-process memory size.

Proof of Lower Bound

Computing the APSP of a sparse graph is a 3NL computation.

The total number of operations to compute the APSP is $\Omega(n^2|S|)$.

- By calculating the number of computation operations required in the elimination of the top-level supernodes, which is a part of the total operations

The bandwidth and latency lower bounds for solving the APSP of sparse graphs are $\Omega(n^2P + |S|^2)$ and $\Omega(\log^2 p)$, respectively.

- By applying the lower bound of operations and M to the 3NL computation model lower bound
- By summing the lower bound of the latency cost during the elimination of each level of supernodes

Outline

- Background
- Several Algorithmic Techniques
- Our Method
- Proof of Lower Bound
- Summary

Summary

Parameter	2D-DC-APSP	SPARSE-APSP	Lower bound	
			Dense graph	Sparse graph
Per-process memory (M)	$O(\frac{n^2}{P})$	$O(\frac{n^2}{P} + \|S\|^2)$	$\Omega(\frac{n^2}{P})$	$\Omega(\frac{n^2}{P})$
Bandwidth cost (B)	$O(\frac{n^2}{\sqrt{P}})$	$O(\frac{n^2 log^2 P}{P} + \|S\|^2 log^2 P)$	$\Omega(\frac{n^2}{\sqrt{P}})$	$\Omega(\frac{n^2}{P} + \|S\|^2)$
Latency cost (L)	$O(\sqrt{P} log^2 P)$	$O(log^2 P)$	$\Omega(\sqrt{P})$	$\Omega(log^2 P)$

Thank you!