CS 598: Communication Cost Analysis of Algorithms

Lecture 13: All-pairs shortest-paths

Edgar Solomonik

University of Illinois at Urbana-Champaign

October 5, 2016

BFS cost

Last lecture we gave the cost of BFS for a diameter d, $n = |V|$ vertex graph as (assuming $\nu \geq \gamma$)

$$T_{\text{BFS}} = O(d \log(P) \cdot \alpha + n/\sqrt{P} \cdot \beta + |E|/P \cdot \nu)$$

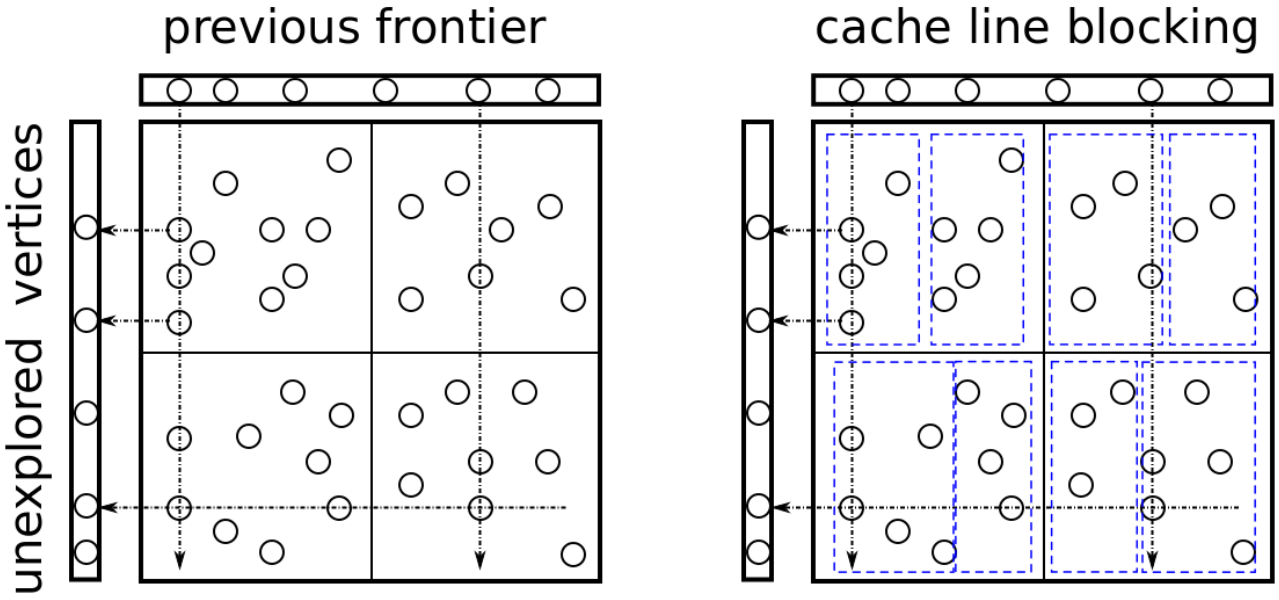
- based on 2D processor grid, where each processor is assigned all edges between two subsets of $n/\sqrt{P}$ vertices
- every BFS step is done by broadcasting the vertices in the frontier then reducing contributions
- each processor receives a total of $n/\sqrt{P}$ vertex labels and reduces as many
- cost assumes that the frontiers and edges are load balanced
- $\log(P)$ factor on latency cost assumes point-to-point $\alpha - \beta$ model, absent for BSP

BFS cost

Which costs will dominate?

$$T_{\text{BFS}} = O(d \log(P) \cdot \alpha + n/\sqrt{P} \cdot \beta + |E|/P \cdot \nu)$$

- Q: what average vertex degree would yield $|E|/P \geq n/\sqrt{P}$?
- A: $|E|/n \geq \sqrt{P}$ would mean the memory-bandwidth cost dominates the interprocessor communication if $\nu \geq \beta$
- if cache line size $L > 1$ and the frontiers are all relatively sparse, BFS may incur a higher memory bandwidth cost by factor of up to L



Bellman Ford

BFS does cannot find weighted shortest paths

- Dijkstra is sequential, Bellman Ford provides more parallelism, but does redundant work
- extra work in Bellman Ford depends simultaneously on weights and graph structure
  - implies proportionately higher memory bandwidth cost for $L = 1$
  - if weights are nearly the same or graph has a sensible (weighted) spatial embedding, Bellman Ford should not cost as much as BFS
  - worst case can cause much redundant work, e.g. one very low-weight length n path in a dense graph
- with respect to Dijkstra, Bellman Ford requires factor of $O(n/Y)$ fewer synchronizations, where Y is the maximum number of edges in any shortest path
- Δ-stepping can be more robust, but harder to implement and manage

All-pairs shortest paths

The output of APSP on a connected graph is a dense matrix of distances

- the usual algorithm is Floyd-Warshall
  - at iteration i, compute all shortest distances with at most $i+1$ edges going through intermediate nodes in the set $\{1, \ldots, i\}$
  - let D_i be the the shortest distances after iteration i, so $D_0 = A$ and $D_i = D_i \oplus D_{i-1}(:, i) \otimes D_{i-1}(i, :)$, that gives

$$D_i(j,k) = \min(D_{i-1}(j,k), D_{i-1}(j,i) + D_{i-1}(i,k))$$

  - requires $O(n^3)$ work (finds correct n^2 paths from exponential-sized set)!
  - each iteration is dependent on the previous

Parallel Floyd-Warshall

We can parallelize Floyd-Warshall by distributing A and D (each D_i) in blocks on a 2D grid

$D = \text{2D-FLOYD-WARSHALL}(A, \Pi[1:\sqrt{P}, 1:\sqrt{P}], n, P)$

$D = A$
for $i = 1$ **to** n
Broadcast $D[:, i]$ along processor rows
Broadcast $D[i, :]$ along processor columns
$D := D \oplus D[:, i] \otimes D[i, :]$

This algorithm has a BSP cost $T_{\text{FW}} = O(n \cdot \alpha + n^2/\sqrt{P} \cdot \beta + n^3/P \cdot \gamma)$.
Q: what memory bandwidth cost does it incur?
A: naively we need to read all of D from memory to cache n times, so $O(n^3/P \cdot \nu)$, but like in LU we can try to find a more efficient blocking...

APSP interpreted algebraically

Lets examine the relationship between APSP and LU more carefully

- denote $A^2 = A \otimes A$ and $A^k = A^{k-1} \otimes A$
- A^k contains all shortest distances with paths consisting of exactly k edges

$$A^k(i,j) = \min_l (A^{k-1}(i,l) + A(l,j))$$

$$= \min_{l_1,\ldots,l_{k-1}} \left(A(i,l_1) + \left(\sum_{m=1}^{k-2} A(l_m, l_{m+1}) \right) + A(l_{k-1}, j) \right)$$

- so we can write the distance matrix as $A^* = A \oplus A^2 \oplus \cdots \oplus A^n$
- in general, $A^* = A \oplus A^2 \oplus A^3 \oplus \ldots$ is called the **closure** of A over the given semiring
- Q: a closure is defined to sum all powers, why stop at A^n for APSP?
- A: no shortest path can contain more than n edges

Computing the closure

For numerical matrices, we can observe

$$(I + A^*)(I - A) = I + A + A^2 + A^3 + \ldots - (A + A^2 + A^3 + A^4 + \ldots) = I$$
$$A^* = (I - A)^{-1} - I$$

and try to compute it by inverting $I - A$. The inverse, like the closure, may not exist.

Gauss-Jordan elimination for the matrix inverse

Computing the matrix inverse is commonly referred to as Gauss-Jordan elimination, which looks very much like Gaussian elimination

$A^* = \textsc{Gauss-Jordan}(A, n)$

Partition $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, where all A_{ij} are $n/2$-by-$n/2$

$B_{11} = \textsc{Gauss-Jordan}(A_{11}, n/2)$

$B_{12} = A_{21} \oplus B_{11} \otimes A_{12}$

$B_{21} = A_{21} \oplus A_{21} \otimes B_{11}$

$B_{22} = A_{22} \oplus B_{21} \otimes B_{12}$

$A^*_{22} = \textsc{Gauss-Jordan}(B_{22}, n/2)$

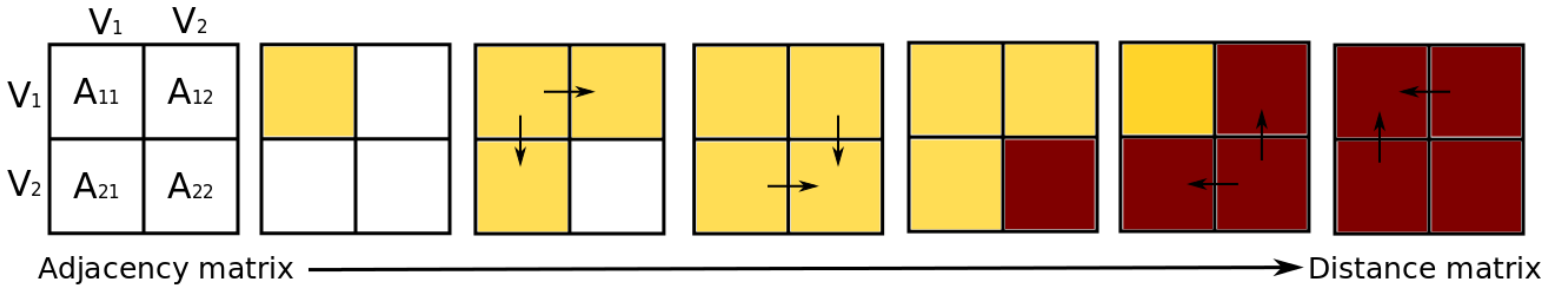
$A^*_{21} := B_{21} \oplus A^*_{22} \otimes B_{21}$

$A^*_{12} := B_{12} \oplus B_{12} \otimes A^*_{22}$

$A^*_{11} := B_{11} \oplus A^*_{12} \otimes A^*_{21}$

For APSP it is known as Kleene's algorithm [Aho, Hopcroft, Ullman 1974]

Divide-and-conquer all-pairs shortest-paths algorithm



Gauss-Jordan elimination requires two dependent recursive calls on $n/2 \times n/2$ matrices and $O(1)$ matrix multiplications, for a BSP cost of

$$T(n,P) = 2T(n/2,P) + O(\alpha + n^2/\sqrt{cP} \cdot \beta) = O(\sqrt{cP} \cdot \alpha + n^2/\sqrt{cP} \cdot \beta)$$

Short pause

Path doubling

We can achieve a logarithmic synchronization cost by a different technique

- note that any shortest path with up to $2k$ edges consists of two shortest paths of up to k edges
- we can write the above algebraically,

$$I \oplus A \oplus \cdots \oplus A^{2k} = (I \oplus A \oplus \cdots \oplus A^k)^2$$

- Q: what is the identity matrix I for the tropical semiring?
- A: the identity matrix is $I = \begin{bmatrix} 0 & \infty & \cdots & \infty \\ \infty & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \infty \\ \infty & \cdots & \infty & 0 \end{bmatrix}$
- i.e. self-loops are distance 0, all other distances are infinite

APSP by path doubling

We can compute APSP via the given recurrence

$$I \oplus A \oplus \cdots \oplus A^{2k} = (I \oplus A \oplus \cdots \oplus A^{k})^2$$

- starting with $I \oplus A$, we need to square the matrix $\log_2(n)$ to get A^*
- Q: does this require more or less operations (γ cost) than Floyd-Warshall?
- A: it requires $O(n^3 \log(n))$ operations, which is more than $O(n^3)$
- the total parallel cost would be

$$T_{\text{PD}} = O(\log(n) \cdot \alpha + n^2 \log(n)/\sqrt{cP} \cdot \beta + n^3 \log(n)/P \cdot \gamma)$$

- the memory bandwidth cost with cache size H is $O\left(\frac{n^3 \log(n)}{\sqrt{HP}} \cdot \nu\right)$

Tiskin's path doubling algorithm

[Tiskin 2001] gives a cheaper way to do path-doubling by realizing that each shortest path with up to $2k$ edges is composed of a shortest path with up to k edges and maybe also a shortest path of exactly k edges

Lets denote $A^{\otimes m} = I \oplus A \oplus \cdots \oplus A^m$, so $A^{\otimes n} = A^*$
and denote $A_s^{\otimes m}$ as all shortest paths with up to m edges that consist of exactly $s < m$ edges, so

$$A_s^{\otimes m}(i,j) = \begin{cases} A^s(i,j) : A^s(i,j) = A^{\otimes m}(i,j) \\ \infty : \text{otherwise} \end{cases}$$

Tiskin's observation amounts to the recurrence

$$A^{\otimes 2k} = A^{\otimes k} \oplus A^{\otimes k} \otimes A_k^{\otimes k}$$

note that $A_s^{\otimes m}$ becomes sparser as we increase m

Furthermore, if A is dense, in expectation $A_k^{\otimes k}$ has n^2/k edges, and if it does not, we can definitely find $s \in [2k/3, k]$ so that $A_s^{\otimes k}$ has at most $3n^2/k$ edges

Analysis of Tiskin's path doubling algorithm

At the kth recursive step a matrix multiplication is performed, with

- a dense matrix (in the worst case) of paths with up to k edges
- a sparse matrix with $O(n^2/2^k)$ non-trivial entries (edges) of paths with exactly $s \approx k$ edges
- the computation cost and bandwidth costs become geometric sums if the matrix multiplication is decomposed appropriately, leading to costs

$$T_{\text{TPD}} = O(\log(n) \cdot \alpha + n^2/\sqrt{cP} \cdot \beta + n^3/P \cdot (\nu/\sqrt{H} + \gamma))$$

- demonstrating that the sparse matrix multiplication cost decreases is nontrivial, blocking should intuitively be like rectangular matrix multiplication (instead of matrix size consider number of nonzeros)

Analysis of Tiskin's path doubling algorithm

We can further achieve $O(\log(P))$ synchronizations rather than $O(\log(n))$

- Q: given $A^{\otimes P}$ and $A_P^{\otimes P}$ with n^2/P edges, how can we compute efficiently A^* with $O(1)$ synchronizations *if all edge weights are nonnegative*?
- A: run Dijkstra's algorithm for n/P starting vertices with each processor on $A_P^{\otimes P}$ to get $A_P^{\otimes P*}$, then compute $A^* = A^{\otimes P} \oplus A_P^{\otimes P*} \otimes A^{\otimes P}$
- its more difficult to do this for arbitrary edge weights, see [Tiskin 2001]