Main Memory
Evaluation of Recursive Queries on Multicore Machines

Mohan Yang, Carlo Zaniolo
UCLA CS Department
Motivation

• Big data analytics on parallel systems like multicore machines and multi-node clusters.
• Many advanced applications require iteration and/or recursion.
  • PageRank, transitive closure, community discovery, network centrality, etc.
• But how to implement recursion on these new massively parallel systems is not well understood.
Motivation

• Recent studies [Afrati et al., 2011], [Afrati et al., 2012], [Shaw et al., 2012] on implementing transitive closure in multi-node clusters.

• Algorithms that deliver optimal performance on multi-node clusters are hardly optimal on multicore machines.

• Thus, we study the evaluation of transitive closure query on multicore machines.
Related Work

• Semi-naive evaluation, Smart, Floyd-Warshall
• I/O efficient algorithms
  • [Warren et al., 1975] [Agrawal et al., 1987] ...
• Parallel algorithms
  • [Valduriez et al., 1988] [Agrawal et al., 1988] [Wolfson et al., 1992] [Cacace et al., 1993]
Transitive Closure (TC)

Consider a relation \( \text{arc}(X, Y) \) that represents the edges of a directed graph, the \textit{transitive closure} of \text{arc} is a relation \( \text{tc}(X, Y) \) such that it contains all pairs \((X, Y)\) that \(X\) can reach \(Y\) via a path in the graph.

\[
\text{tc}(X, X) \leftarrow \text{arc}(X, \_).
\]
\[
\text{tc}(X, Y) \leftarrow \text{tc}(X, Z), \text{arc}(Z, Y).
\]

Similar problems: all-pairs shortest path, bill of materials, maximum capacity path, critical path, most reliable path, etc.
Evaluation Algorithms
Semi-Naive Evaluation

$tc$ after the $i$-th iteration

a path of length at most $i$
Semi-Naive Evaluation

tc after the $i$-th iteration

A path of length at most $i+1$
Semi-Naive Evaluation

\[ tc_0(X, X) = \text{arc}(X, \_), \quad tc = \text{arc}(X, \_). \]
\[ tc_{i+1} = \pi_{X,Z} tc_i(X, Y) \bowtie \text{arc}(Z, Y) - tc \]
\[ tc = tc \cup tc_{i+1} \]

- Require \( O(n) \) iterations for an \( n \)-vertex graph.
- Build an index (adjacency list) on \( \text{arc} \) since it is unchanged between iterations.
Smart Algorithm

tc after the $i$-th iteration

a path of length at most $2^i$

a path of length exactly $2^i$
Smart Algorithm

tc after the $i$-th iteration

a path of length at most $2^{i+1}$
Smart Algorithm

\[ \text{tc}_0(X, X, 0) = \text{arc}(X, \_ ) \]
\[ \text{tc}_0(X, Y, 1) = \text{arc}(X, Y) \]
\[ \text{tc}_{i+1} = \text{tc}_i \cup \pi_{X,Z,L+2^i} \text{tc}_i(X, Y, L) \bowtie \text{tc}_i(Y, Z, 2^i) \]

- No redundant derivations.
- Require \( O(\log n) \) iterations for an \( n \)-vertex graph.
Single-Source Closure Algorithms

For each vertex \( x \), use semi-naive to compute its single-source closure (SSC).

- **SSC1** – represent SSC as a set.
- **SSC2** – represent SSC as an array.

\[
\begin{align*}
\text{tc}(X, X) & \leftarrow \text{arc}(X, _). \\
\text{tc}(X, Y) & \leftarrow \text{tc}(X, Z), \text{arc}(Z, Y).
\end{align*}
\]

Instantiate variable \( X \) with a constant \( x \) where \( x \) a vertex in the graph.

\[
\begin{align*}
\text{tc}(x, x) & \leftarrow \text{arc}(x, _). \\
\text{tc}(x, Y) & \leftarrow \text{tc}(x, Z), \text{arc}(Z, Y).
\end{align*}
\]
Algorithm Implementations

• Parallel implementation in C++
  • Semi-naive
  • Smart
  • SSC: each thread works on one vertex at a time.

• Testing environment
  • A multicore machine with 4 AMD CPUs (64 cores in total) and 256G memory (8 NUMA regions).

• An optimization on NUMA hardware
  • Duplicate arc (adjacency list) on each NUMA region.
Performance Comparison
Synthetic Test Graphs

tree-11

grid-150
grid-250

sf-100K

gnp-0.001
gnp-0.01
gnp-0.1
gnp-0.5
Real-Life Test Graphs

patent

wiki
- A subgraph of Wikipedia knowledge graph.

road
- Eastern USA road network.

stanford
- Stanford web graph from 2002.
Optimal Execution Time

Neither Semi-naive nor Smart is the fastest algorithm on any test graph.
Execution Time Comparison

The array initialization cost dominates the total time.

Set representation used by SSC1 becomes less efficient.

![Bar chart for tree-17 showing execution times for SSC1 and SSC2.]

![Bar chart for grid-150 showing execution times for SSC1 and SSC2.]

- The array initialization cost dominates the total time.
- Set representation used by SSC1 becomes less efficient.
Hybrid Algorithm – SSC12

• Combine the advantage of SSC1 and SSC2.
• Start with SSC1, switch to SSC2 if the predicted number of set operations is above a certain threshold.
• Always select the better algorithm (between SSC1 and SSC2) for each vertex in the graph.
Optimal Execution Time

Time (s)

- tree-17
- grid-150
- patent
- wiki
- road
- stanford

Legend:
- SSC1
- SSC2
- SSC12
Conclusion & Future Work

• The simple SSC12 algorithm proposed in this paper is an ideal choice for main memory multicore evaluation of recursive query (expressing TC and similar computation).

• Future work
  • Investigate more algorithms, and derive simple criteria for deciding which system can be most cost-effective for the application at hand.
  • Integrate these results into DeALS towards an efficient and scalable system on multicore machines.
Questions