PARALLEL IMPLEMENTATION OF DIJKSTRA'S ALGORITHM USING MPI LIBRARY ON A CLUSTER.

INSTRUCTOR: DR RUSS MILLER

ADITYA PORE
THE PROBLEM AT HAND

- Given: A directed graph $G = (V, E)$. Cardinalities $|V| = n$, $|E| = m$.
- $S$(Source $)$: distinguished vertex of the graph.
- $w$: weight of each edge, typically, the distance between the two vertexes.
- **Single source shortest path**: The single source shortest path (SSSP) problem is that of computing, for a given source vertex $s$ and a destination vertex $t$, the weight of a path that obtains the minimum weight among all the possible paths.
DIJKSTRA’s ALGORITHM AT A GLANCE

- Dijkstra’s algorithm is a graph search algorithm that solves single-source shortest path for a graph with nonnegative weights.
- Widely used in network routing protocol, e.g., Open Shortest Path First (OSPF) protocol

How to reach Downtown from Maple Road??

24 Node US-Mesh Network
LETS GET TO KNOW THE ALGORITHM WITH AN EXAMPLE

Dijkstra’s Algorithm

Fig. 2 8-node simple network

Table 1. The routing table for node A
Dijkstra’s algorithm 1st round

Fig. 2  8-node simple network

Table 1. The routing table for node A
Dijkstra’s algorithm-2\textsuperscript{nd} round

![8-node simple network](image)

Fig. 2 8-node simple network

<table>
<thead>
<tr>
<th>cluster</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1, A</td>
<td>4, A</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>AB</td>
<td></td>
<td></td>
<td>3, B</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>5, B</td>
</tr>
</tbody>
</table>

Table 1. The routing table for node A
Dijkstra’s algorithm-3rd round

Fig. 2  8-node simple network

Table 1.  The routing table for node A
Dijkstra’s algorithm-4th round

Fig. 2  8-node simple network

Table 1. The routing table for node A
Dijkstra’s algorithm-5th round

Fig. 2  8-node simple network

Table 1. The routing table for node A
Dijkstra’s algorithm - 6th round

Fig. 2 8-node simple network

Table 1. The routing table for node A
Dijkstra’s algorithm-6th round

Fig. 2 8-node simple network

Table 1. The routing table for node A
Create a cluster $cl[V]$  
Given a source vertex $s$  
While (there exist a vertex that is not in the cluster $cl[V]$)  
  
  FOR (all the vertices outside the cluster)  
  Calculate the distance from non-member vertex to $s$ through the cluster  
  END  
  ** $O(V)$ **  
  Select the vertex with the shortest path and add it to the cluster  
  ** $O(V)$ **  

**ANALOGY**

```plaintext
Dijkstra(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```
DIJKSTRA’S ALGORITHM

- **Running time** $O(V^2)$
  - In order to obtain the routing table, we need $O(V)$ rounds iterations (until all the vertices are included in the cluster).
  - In each round, we will update the value for $O(V)$ vertices and select the closest vertex, so the running time in each round is $O(V)$.
  - So, the total running time is $O(V^2)$

- **Disadvantages:**
  - If the scale of the network is too large, then it will cost a long time to obtain the result.
  - For some time-sensitive app or real-time services, we need to reduce the running time.
PARALLEL DIJKSTRA’S ALGORITHM

- Each core identifies its closest vertex to the source vertex;
- Perform a parallel prefix to select the globally closest vertex;
- Broadcast the result to all the cores;
- Each core updates its cluster list.
Parallel Dijkstra’s algorithm

- **Step 1:** find the closest node in my subgroup.
- **Step 2:** use parallel prefix to find the global closest.

<table>
<thead>
<tr>
<th>cluster</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>A AB</td>
<td>1, A</td>
<td>4, A</td>
<td>5, A</td>
<td>5, A</td>
<td>6, A</td>
<td>7, A</td>
<td>8, A</td>
</tr>
<tr>
<td>ABC</td>
<td>3, B</td>
<td>4, B</td>
<td>5, B</td>
<td>6, B</td>
<td>7, B</td>
<td>8, B</td>
<td>9, B</td>
</tr>
<tr>
<td>ABCH</td>
<td>4, C</td>
<td>5, C</td>
<td>6, C</td>
<td>7, C</td>
<td>8, C</td>
<td>9, C</td>
<td>10, C</td>
</tr>
<tr>
<td>ABCHD</td>
<td>4, C</td>
<td>5, C</td>
<td>6, C</td>
<td>7, C</td>
<td>8, C</td>
<td>9, C</td>
<td>10, C</td>
</tr>
<tr>
<td>ABCHDE</td>
<td>5, D</td>
<td>6, D</td>
<td>7, D</td>
<td>8, D</td>
<td>9, D</td>
<td>10, D</td>
<td>11, D</td>
</tr>
<tr>
<td>ABCHDEG</td>
<td>6, E</td>
<td>7, E</td>
<td>8, E</td>
<td>9, E</td>
<td>10, E</td>
<td>11, E</td>
<td>12, E</td>
</tr>
<tr>
<td>ABCHDEGF</td>
<td>6, E</td>
<td>7, E</td>
<td>8, E</td>
<td>9, E</td>
<td>10, E</td>
<td>11, E</td>
<td>12, E</td>
</tr>
</tbody>
</table>

THE ACTUAL ALGORITHM AT WORK
PARALLEL DIJKSTRA’S ALGORITHM

Create a cluster $cl[V]$

Given a source vertex $s$

Each core handles a subgroup of $V/P$ vertices

While (there exist a vertex that is not in the cluster $cl[V]$)
{
    FOR (vertices in my subgroup but outside the cluster)
        Calculate the distance from non-member vertex to $s$
        through the cluster;
        Select the vertex with the shortest path as the local
        closest vertex;
    END

    ** Each processor work in parallel $O(V/P)$ **

Use the parallel prefix to find the global closest vertex
among all the local closest vertices from each core.

** Parallel prefix $\log(P)$ **
}

MPI _MINLOC operation??
PARALLEL DIJKSTRA’S ALGORITHM

**RUNNING TIME**: $O(V^2/P + V \log(P))$

- $P$ is the number of cores used.

- In order to obtain the routing table, we need $O(V)$ rounds iteration (until all the vertices are included in the cluster).

- In each round, we will update the value for $O(V)$ vertices using $P$ cores running independently, and use the parallel prefix to select the global closest vertex, so the running time in each round is $O(V/P) + O(\log(P))$.

- So, **the total running time is $O(V/P + V^2 \log(P))$**
RESULTS AND ANALYSIS

- Implemented using MPI: Stats Averaged over 10 rounds of Computation.
- Establish trade-off between running times as a function of number of cores deployed.
- Evaluate speed up and efficiency!!!!

EXPERIMENT A: (More Graphs and Analysis)
- Compute for fixed size input: 10000
- Run Routines for: 1 32-core node, 3 12-core node, 16 dual-core

EXPERIMENT B: (Achieved Desired Results)
- Compute for different input size: Typically 625, 2500, 10000
- Run Routine on 1 32-core Node.
EXPERIMENT A: RUN TIME

Tabulation of Results:
Relationship Observed: Number of Cores Versus The Running Time (seconds)

Conclusions:
(a) Run Time is Inversely proportional to number of cores: Cores belong to the same node in cluster
(b) Significant Increase observed for two configurations out of three, namely 16*2 Core and 3*12 Core.

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Configurations</th>
<th>RUNTIMES</th>
<th>I N</th>
<th>SECONDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16*2 Core</td>
<td>4.37263</td>
<td>2.36273</td>
<td>1.98442</td>
</tr>
<tr>
<td>2</td>
<td>3*12 Core</td>
<td>4.67321</td>
<td>2.42865</td>
<td>1.34567</td>
</tr>
<tr>
<td>3</td>
<td>1*32 Core</td>
<td>5.45321</td>
<td>2.68753</td>
<td>1.56782</td>
</tr>
</tbody>
</table>
EXPERIMENT A: RUN TIME

GRAPHICAL DESCRIPTION OF RUN TIME ANALYSIS

Run-Time (Seconds) vs Number of Cores

- 16*2 Core
- 3*12 Core
- 1*32 Core
EXPERIMENT A : SPEED UP

Tabulation of Results :
Relationship Observed : Number of Cores Versus The Speed-Up

Conclusions:
(a) Speed-Up is Directly proportional to number of cores :Cores belong to the same node in cluster
(b) Significant Decrease observed for two configurations out of three, namely 16*2 Core and 3*12 Core.

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurations</td>
<td>SPEED UP: GIVES A MEASURE OF SCALABILITY OF THE SYSTEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16*2 Core</td>
<td>1</td>
<td>1.85324</td>
<td>2.10978</td>
<td>0.85432</td>
<td>0.54332</td>
<td>0.32456</td>
</tr>
<tr>
<td>3*12 Core</td>
<td>1</td>
<td>1.94433</td>
<td>3.75567</td>
<td>6.74352</td>
<td>1.86432</td>
<td>0.86032</td>
</tr>
<tr>
<td>1*32 Core</td>
<td>1</td>
<td>1.98765</td>
<td>3.66541</td>
<td>6.40321</td>
<td>6.78432</td>
<td>4.89543</td>
</tr>
</tbody>
</table>
### EXPERIMENT A: SPEED-UP

**Graphical Description of Speed-Up Analysis**

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>16*2 Core</th>
<th>3*12 Core</th>
<th>1*32 Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.85432</td>
<td>0.32456</td>
<td>0.54332</td>
</tr>
<tr>
<td>2</td>
<td>1.85324</td>
<td>1.94433</td>
<td>1.86432</td>
</tr>
<tr>
<td>4</td>
<td>2.10978</td>
<td>3.75567</td>
<td>3.66541</td>
</tr>
<tr>
<td>8</td>
<td>6.74352</td>
<td>6.40321</td>
<td>6.78432</td>
</tr>
<tr>
<td>16</td>
<td>6.74352</td>
<td>6.40321</td>
<td>6.78432</td>
</tr>
<tr>
<td>32</td>
<td>4.89543</td>
<td>0.86032</td>
<td>0.32456</td>
</tr>
</tbody>
</table>

**Graph: Number of Cores vs Speed-Up**
- **Legend:**
  - 16*2 Core
  - 3*12 Core
  - 1*32 Core

The graph shows the speed-up for different numbers of cores, comparing 16*2 Core, 3*12 Core, and 1*32 Core configurations.
EXPERIMENT A : EVALUATING EFFICIENCY VIA SPEED-UP

Tabulation of Results :

Relationship Observed : Number of Cores Versus The Efficiency

Conclusions:

(a) Efficiency varies inversely with number of cores.

(b) Significant Decrease observed for two configurations out of three, namely 16*2 Core and 3*12 Core

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16*2 Core</td>
<td>0.92662</td>
<td>0.52745</td>
<td>0.10679</td>
<td>0.05395</td>
<td>0.03014</td>
<td></td>
</tr>
<tr>
<td>3*12 Core</td>
<td>0.97216</td>
<td>0.93891</td>
<td>0.84294</td>
<td>0.11652</td>
<td>0.04688</td>
<td></td>
</tr>
<tr>
<td>1*32 Core</td>
<td>0.99383</td>
<td>0.91630</td>
<td>0.80040</td>
<td>0.42402</td>
<td>0.15298</td>
<td></td>
</tr>
</tbody>
</table>

EFFICIENCY: Gives a measure of fraction of time utilized by processors (Cores) for particular Computation.
EXPERIMENT A: EFFICIENCY

GRAPHICAL DESCRIPTION OF ANALYSIS

NUMBER OF CORES VS EFFICIENCY (%)

16*2 Core  3*12 Core  1*32 Core
EXPERIMENT A : COST

- Tabulation of Results :
- Relationship Observed : Number of Cores Versus Cost of Computation
- Conclusions:
  (a) Run Time is Inversely proportional to number of cores
  (b) Significant Increase observed for 16*2 Core configuration.
  (c) Parallel computing is cost effective for modest speedups.

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurations</td>
<td>Cost: Product of number of cores/resources used times execution time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16*2 Core</td>
<td>4.37263</td>
<td>4.72546</td>
<td>15.93768</td>
<td>43.90672</td>
<td>126.29936</td>
<td>404.9094</td>
</tr>
<tr>
<td>3*12 Core</td>
<td>4.67321</td>
<td>4.85730</td>
<td>5.38268</td>
<td>5.78728</td>
<td>46.20224</td>
<td>206.5027</td>
</tr>
</tbody>
</table>
EXPERIMENT A : COST
GRAPHICAL DESCRIPTION

Number of Cores VS Cost of Computation
EXPERIMENT B : RUN TIME

- Tabulation of Results:
- Relationship Observed: Input-Size VS Running-Time
- Conclusions:
  (a) Run Time varies Inversely with the number of Cores.
  (b) Algorithm found to be most-effective performance-wise for 16 Core configuration.
  (c) 32-Cores: Run time increases Slightly as communication overhead defeats the purpose of using more number of cores for computation.

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input-Size</td>
<td>RUNTIME</td>
<td>IN</td>
<td>SECONDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>625</td>
<td>0.76589</td>
<td>0.70187</td>
<td>0.58532</td>
<td>0.42618</td>
<td>0.25125</td>
<td>0.30325</td>
</tr>
<tr>
<td>2500</td>
<td>1.08971</td>
<td>0.79816</td>
<td>0.57821</td>
<td>0.41344</td>
<td>0.38815</td>
<td>0.44516</td>
</tr>
<tr>
<td>10000</td>
<td>3.25618</td>
<td>1.89876</td>
<td>1.10542</td>
<td>0.78516</td>
<td>0.54812</td>
<td>0.80124</td>
</tr>
</tbody>
</table>
EXPERIMENT B: RUN TIME
GRAPHICAL DESCRIPTION OF RUN TIME ANALYSIS

Run Times Vs Input-Size (Measured as function of increasing no of Cores)

- 625
- 2500
- 10000

Run Times:

- 1: 0.76589, 1.08971, 3.25618
- 2: 0.70187, 0.79816, 1.89876
- 4: 0.58532, 0.78216, 1.10542
- 8: 0.42618, 0.78516
- 16: 0.25125, 0.54812
- 32: 0.30325, 0.44516

Input Sizes:

- 0
- 0.5
- 1
- 1.5
- 2
- 2.5
- 3
- 3.5

Cores:

- 625
- 2500
- 10000
EXPERIMENT B : SPEED UP

Tabulation of Results:

Relationship Observed: Input-Size (with increasing number of nodes) Versus The Speed-Up

Conclusions:
(a) Speed-Up is Directly proportional to number of cores.
(b) Significant Decrease observed, after a certain point for all three input sizes owing to communication latency.
(c) As the input size increases, the number of cores used to achieve maximum speed up increases.

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input-Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>625</td>
<td>1</td>
<td>1.09121</td>
<td>1.30849</td>
<td>1.79710</td>
<td>3.05906</td>
<td>2.52560</td>
</tr>
<tr>
<td>2500</td>
<td>1</td>
<td>1.36527</td>
<td>1.88462</td>
<td>2.63571</td>
<td>2.80744</td>
<td>2.44790</td>
</tr>
<tr>
<td>10000</td>
<td>1</td>
<td>1.71489</td>
<td>2.94564</td>
<td>4.14715</td>
<td>5.94801</td>
<td>4.06392</td>
</tr>
</tbody>
</table>

SPEED UP: most obvious benefit of using a parallel computer is the reduction in the running time of the code.
EXPERIMENT B : EVALUATING EFFICIENCY VIA SPEED-UP

Tabulation of Results :
Relationship Observed : Input-Size(Increasing number of cores )Versus The Efficiency

Conclusions:
(a) Efficiency varies inversely with number of cores .
(b) Significant Decrease observed as number of cores increases
(c) Gives an indication that benefit of reduced running time cannot outperform cost of operation.

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input-Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>625</td>
<td>1</td>
<td>0.54560</td>
<td>0.32712</td>
<td>0.22463</td>
<td>0.19119</td>
<td>0.07893</td>
</tr>
<tr>
<td>2500</td>
<td>1</td>
<td>0.68263</td>
<td>0.47115</td>
<td>0.32946</td>
<td>0.17546</td>
<td>0.07649</td>
</tr>
<tr>
<td>10000</td>
<td>1</td>
<td>0.85744</td>
<td>0.73641</td>
<td>0.51834</td>
<td>0.37175</td>
<td>0.12699</td>
</tr>
</tbody>
</table>

Efficiency: For example, if E = 50%, the processors are being used half of the time to perform the actual computation.
EXPERIMENT B: EFFICIENCY

GRAPHICAL DESCRIPTION OF ANALYSIS

Input-Size (increasing number of cores) VS Efficiency (%)

- 625
- 2500
- 10000
A QUICK LOOK UP AT THE AMDAHL’S LAW

The maximum speed up that can be achieved by using N resources is:

$$\frac{1}{F+(1-F)/N}.$$  

As an example, if $F$ is only 10%, the problem can be sped up by only a maximum of a factor of 10, no matter how large the value of $N$ used.

A great part of the craft of parallel programming consists of attempting to reduce $F$ to the smallest possible value.
SUMMARY OF ACCOMPLISHMENTS

- Parallel Implementation using MPI library routines and CCR.
- Intel implementation of the Message Passing Interface
- Multi-network support: TCP/IP, Infiniband, Myrinet - by default the best network is tried first.
- GNU Compiler Wrapper
- Used simplified startup `mpirun`
- Launch combines mpd daemons and mpiexec.
- Detailed Understanding of MPI APIs()
  - MPI_Init() and MPI_Finalize()
  - MPI_Comm_size() and MPI_Comm_rank()
  - MPI_Reduce() MPI_Bcast()
  - MPI_Gather()
REFERENCES

THANK-YOU

ANY QUESTIONS??