A short talk on parallelizing COSA

Jiading Gai

Center for Research Computing
University of Notre Dame
Notre Dame, IN 46556

jgai@nd.edu

Aug 23, 2010

Clustering on different subsets of the attributes (COSA):

N objects $\to$ L clusters:

$$c(i) = l \Rightarrow i \in G_l \ (1 \leq i \leq N, 1 \leq l \leq L)$$

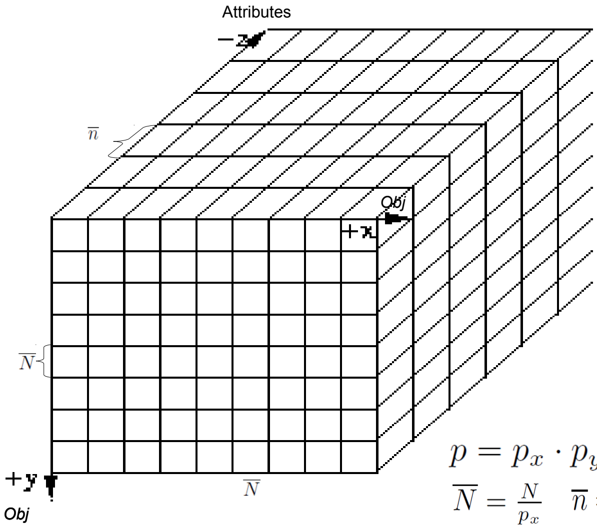
$$Q(c, \{\mathbf{w}_l\}_1^L) = \sum_{l=1}^{L} \frac{W_l}{N_l^2} \sum_{c(i)=l} \sum_{c(j)=l} D_{ij}[\mathbf{w}_l]$$

- Each object i has n attributes:$x_i = (x_{i1}, ..., x_{in})$
- $\mathbf{w}_l = \{w_{kl}\}_1^n$, $1 \leq l \leq L$.
- $D_{ij}[\mathbf{w}_l] = \sum_{k=1}^{n} w_{kl} d_{ijk}$, where $d_{ijk} = \frac{|x_{ik} - x_{jk}|}{s_k}$; $s_k = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} |x_{ik} - x_{jk}|$.
- Augumented to avoid clustering only on a single attribute: $D_{ij}[\mathbf{w}_l] \ \to \ D_{ij}^{(\lambda)}[\mathbf{w}_l]$.

Minimization Scheme

- Start with an initial guess for $w_{ki} = \frac{1}{n}$, $\eta = \lambda$
- **outer WHILE**
- **inner WHILE**
- Fix w_{ki}, minimize the criterion w.r.t the encoder $c(\cdot)$
  - $D_{ij}^{(\eta)}[\mathbf{w}] = -\eta \cdot \log \sum_{k=1}^{n} w_{ki} \cdot e^{-\frac{d_{ijk}}{\eta}}$.
    $d_{ijk} = \frac{|x_{ik} - x_{jk}|}{s_k}$, $s_k = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} |x_{ik} - x_{jk}|$.
  - $D_{ij}^{1}[\mathbf{W}] = \max(D_{ij}^{(\eta)}[\mathbf{w}_{c(i|\mathbf{W})}], D_{ij}^{(\eta)}[\mathbf{w}_{c(j|\mathbf{W})}])$ (30)
    $D_{ij}^{(2)}[\mathbf{W}] = \sum_{k=1}^{n} \max(w_{k,c(i|\mathbf{W})}, w_{k,c(j|\mathbf{W})}) d_{ijk}$ (33)
  - $\mathrm{KNN}(i) = \{j | D_{ij} \le d_{i(K)}\}$.
- Fix $c(\cdot)$, minimize the criterion w.r.t the weights w_{ki}
  - $w_{ki} = \frac{e^{-\frac{S_{ki}}{\lambda}}}{\sum_{k'=1}^{n} e^{-\frac{S_{ki}}{\lambda}}}$, $S_{ki} = \frac{1}{K} \sum_{j \in KNN(i)} d_{ijk}$.
- **END inner WHILE**
- Increment $\eta = \eta + \alpha \cdot \lambda$
- **END outer WHILE**
- This iterative procedure is continued until convergence.

Parallel COSA algorithm:



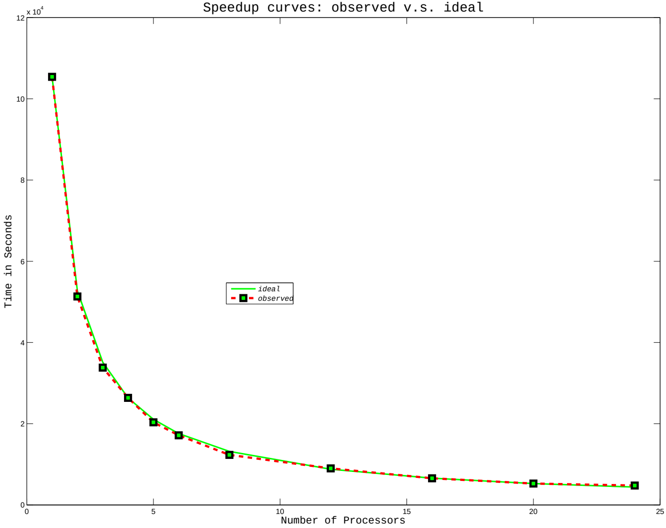
$$p = p_x \cdot p_y \cdot p_z$$

$$\overline{N} = \frac{N}{p_x} \quad \overline{n} = \frac{n}{p_z}$$

Observed speedup on a real-world data

c22_ceu.txt: (N=1000, n=13,000)
DELL R810 with 24 Xeon processors and a total 132GB memory

p	p_x	p_y	p_z	time
1	1	1	1	105417.38615 sec
2	1	1	2	51324.306341 sec
3	1	1	3	33812.867425 sec
4	1	1	4	26374.128593 sec
5	1	1	5	20340.902349 sec
6	1	1	6	17151.958595 sec
8	2	2	2	12327.523793 sec
12	2	2	3	9007.370069 sec
16	2	2	4	6559.531571 sec
20	2	2	5	5273.171368 sec
24	2	2	6	4778.485166 sec


Speedup curves: observed v.s. ideal
x 10^4
12
10
8
6
4
2
0
Time in Seconds
0
5
10
15
20
25
Number of Processors
ideal
observed

Conclusion

- Matching performance between MPI-COSA and Fortran-COSA executable (released by the original authors)
- MPI-COSA Source code was released publicly on launchpad.net

Changes | Files | Help

~jgai/+junk/COSA : /parallel_cosa2.c (revision 49)

browse files | view revision | view changes to this file | download file

Line	Revision	Contents
1	13	`#include <stdio.h>`
2		`#include <stdlib.h>`
3	46	`#include <assert.h>`
4	13	`#include <math.h>`
5		`#include <string.h>`
6		`#include <float.h>`
7		`#include <mpi.h>`
8		`#include <unistd.h>`
9	36	`#include "matrix_util.h"`
10	19	`#define GAI_DEBUG 0`
11		
12	13	`typedef struct{`
13	36	`        int objectID;`
14	13	`        float distance;`
15		`} NN_T; // nearest neighbor struct`
16	36	
17	14	`typedef struct`
18		`{`
19		`        int p;`
20		`        MPI_Comm comm;//entire grid`
21	20	`        MPI_Comm slice_comm_along_z;// free_coords=[1,1,0]`
22		`        MPI_Comm slice_comm_along_x;// free_coords=[0,1,1]`
23	16	`        MPI_Comm depth_comm;// free_coords=[0,0,1]`
24	18	`        MPI_Comm row_comm;// free_coords=[1,0,0]`
25	25	`        MPI_Comm col_comm;// free_coords=[0,1,0]`

Performance Analysis

Sequential COSA:

$$T_1 = [4N^2 + (10n + 2K)N^2 + (5K + 5)nN + 3n + N] \cdot T^{flop} \cdot \mathbb{T}$$

Parallel COSA :

$(p = p_x \cdot p_y \cdot p_z,\ \overline{N} = \frac{N}{p_x},\ \overline{n} = \frac{n}{p_z})$

$$\begin{aligned}
T_q^{comp} &= [5\overline{N}^2 + (10\overline{n} + 2K)\overline{N}^2 + (5K + 5)\overline{n}\overline{N} + \\
&\quad (2K + 1)\overline{N} \cdot p_x \cdot K + 2\overline{N} + 2\overline{n}] \cdot T^{flop} \cdot \mathbb{T} \\
T_q^{comm} &= [\overline{n} \cdot p_x p_y \cdot \log(p_x p_y) + \overline{N}^2 \cdot p_z \log(p_z) + 2\overline{N}^2 + \\
&\quad 3\overline{N}K \cdot p_x \log(p_x) + \overline{n}\overline{N} \cdot p_x \log(p_x) + \overline{N} \cdot p_z \log(p_z) + \\
&\quad p \log(p)] \cdot T^{Transmit} \cdot \mathbb{T}
\end{aligned}$$

$$\begin{aligned}
T_q &= T_q^{comp} + T_q^{comm} \\
&= [\frac{6N^2 n}{q} + \frac{N^2}{q} + \frac{N^2}{q}\log(\frac{N}{p}) + pK + \frac{nNK}{p} + \frac{n^2 N}{p} + \frac{2nN}{p}] \cdot \\
&\quad T^{flop} \cdot \mathbb{T} + [np\log(p) + 2pK] \cdot T^{Transmit} \cdot \mathbb{T}
\end{aligned}$$

Performance Analysis:

$$
\begin{aligned}
T_q &= T_q^{comp} + T_q^{comm} \\
&= [\frac{6N^2 n}{q} + \frac{N^2}{q} + \frac{N^2}{q}\log(\frac{N}{p}) + pK + \frac{nNK}{p} + \frac{n^2 N}{p} + \frac{2nN}{p}] \cdot \\
&\quad T^{flop} \cdot \mathbb{T} + [np\log(p) + 2pK] \cdot T^{Transmit} \cdot \mathbb{T}
\end{aligned}
$$

- $T^{Transmit}$ is the time of sending a single message containing a floating point number.
- T_q^{comm} is insignificant compared to T_q^{comp}, as N increases. For example
  - $N = 11,000$, $n = 2,000$, $K = \sqrt{N} = 105$ and $q = 16 (p = 4)$
  - $T^{Transmit} = 10 \times 10^{-6}$ *seconds*, $T^{flop} = 7 \times 10^{-11}$ *seconds*
    $T_q^{comp} = 5.9875 \cdot \mathbb{T} \gg T_q^{comm} = 0.1684 \cdot \mathbb{T}$

Predicted Speedup:

$$
S = \frac{T_1}{T_q} = \frac{[6N^2 n + N^2 + N\log(N) + KnN + n^2 N + 2nN] \cdot T^{flop} \cdot \mathbb{T}}{T_q^{comp} + T_q^{comm}}
$$