Applied Mathematics & Information Sciences
An International Journal

© 2012 NSP
Natural Sciences Publishing Cor.

Graceful Degradation for Top-Down Join Enumeration via similar sub-queries measure on Chip Multi-Processor

YongHeng Chen[1],[2] *and ChunYan Yin*[1]

[1] Zhangzhou Normal University, Zhangzhou 363000, China
[2] Department of Computer Science and Engineering, Zhangzhou Normal University, Zhangzhou 363000, China

Received: Feb. 8, 2012; Revised Apr. 24, 2012; Accepted Jun. 6, 2012

Abstract: Most contemporary database systems query optimizers exploit System-R's dynamic programming method (DP) to find the optimal query execution plan (QEP) without evaluating redundant sub-plans. However, in the relational database setting today, large queries containing many joins are becoming increasingly common. Based on this trend, it has become temping to improve the DP performance. Chip Multi-Processor (CMP) present new opportunities for improving database performance on large queries. Based on CMP, this paper realizes the partial execution plans among the identified similar sub-queries and global execution plan among the constructed connected join pairs according to the generated partial solutions by uniform parallelizing top-down dynamic programming query optimization. Our theoretical results and empirical evaluation show that our algorithm could gracefully degrade the complexity degree for top-down join enumeration with large number of tables and impressive gains in the performance in terms of both output quality and running time.

Keywords: chip multi-processor, parallel query processing, DP query optimization.

1. Introduction

On the hardware front, the development trend of processor is transforming from high-speed single-core to Chip Multi-Processor, and from instructions level parallel to thread level parallel. Tomorrow's computer will have more cores rather than exponentially faster clock speeds, and software designs must be restructured to fully exploit the new architectures. The question for database researchers is this: how best can we use this increasing multithreading capability to improve database performance in a manner that scales well with machine size [4, 11, 12]?

Based on this trend, it has become temping to revisit the concepts of database parallelism in the light of those emerging hardware architectures. Recently, by exploiting the new wave of multi-core processor architecture, Han et al. first propose a novel algorithm PDPsva to parallelize query optimization process to exploit multi-core processor architectures whose main memory is shared among all cores [1]. PDPsva generated QEPs for all smaller quantifiers' sets (i.e. size-driven). On contrary, DP optimizers such as DPcpp [2] and DPhyp [3], which directly traverse a query graph to generate join pairs, i.e., only considers pairs of connected sub-queries. Thus, plan generation mainly use of join pair without cross products, reduce execution time. Further, DPhyp is capable to handle complex join predicates efficiently.

But these algorithms discussed above which all constructed based on bottom-up join enumeration method. By contrast with the research about bottom-up method, the research about top-down join enumeration is relative less recently. Leonard D. Shapiro et al estimates the lower and upper bounds of top-down query optimization [**?**,13]. Top-down join enumeration dynamic programming method can derive upper bounds for the cost of the plans it generated which is not available to typical bottom-up DP method.

Dynamic programming methods, regardless of Top - down or not, face a difficult for complex queries because of its inherent exponential nature owing to the explosive search space. Many works have been done to find a good plan for a complex query by greedy and randomized methods of query optimization, e.g. iterative improvement (II) [6], iterative Dynamic Programming (IDP) [5], simulated annealing (SA), genetic algorithm (GA) and so on. These

* Corresponding author: e-mail:cyh771@163.com

© 2012 NSP
Natural Sciences Publishing Cor.

algorithms are mostly resolving problem by the randomization method.

In order to improve quality of the output plan and consider the characteristics of the query, the method based on identifying similar sub-queries in the complex query is proposed. John W. Raymond and Peter Willett give a thorough survey of the various approaches towards the detection of subgraph isomorphism [14]. Qiang Zhu et al. aim at finding the largest common induced subgraph of two graphs [9]. Meduri Venkata Vamsikrishna constructed plan by re-using the query plans among the identified similar sub-queries and avoided multiple plan construction for each join candidate in order to make memory efficient [10, 8]. However these algorithms are proposed for single - core CPUs.

In this paper, we firstly combine the measurement of similar sub-queries with the constructing connected join pairs. In order to take advantage of multi-cores architecture, a comprehensive and practical framework for parallelizing top-down dynamic programming query optimization is further been proposed to achieve partial solutions among the identified similar sub-queries and global execution plan among the constructed connected join pairs according to the generated partial similar sub-query plans.

The rest of this paper is organized as follows. We firstly construct similar sub-queries SSQ and connected join pairs set CSLQS. Then we generate query plan based on constructed SSQ and CSLQS. Finally we present the results of performance evaluation and conclude this paper.

2. Generation of SSQ and CSLQS

2.1 Preliminary Concepts

A query structure graph G is denoted by G(V, E, T, P, α, β), where V is the finite set of its vertices, $E \in V * V$ the set of edges, α a function assigning labels to the vertices and β a function assigning labels to the edges. $T = \{R_1, R_2 \dots R_n\}$ is the set of tables referred in G and $P = \{p_1, p_2 \dots p_m\}$ the set of all predicates referred in G. $\alpha : x \to R$ is a one-to-one function, where $x \in V$ and $R \in T$. sizeof (x) denotes the size of table α(x). In G, each vertice $v \in V$ is labeled with sizeof(α(x)). $\beta : e \to c$ is a function, where $e \in E$ and $c \in 2^P$. selof (e) denotes the selectivity of β(e). NumRel (T) denotes the number of relations in T.

The edge $e = (u, v) \in E$ is said to be incident with vertices u and v, where u and v are the end of e. These two vertices are called adjacent. The set of vertices adjacent to v is presented as ad(v). vertices(e) denotes the set of (one or two) vertices connected by edge e in a query graph.

For a subquery $S(V', E', T', P', \alpha', \beta')$ of G(V, E, T, P, α, β), the neighborhood of S is denoted as $ad(S) = \{v \in (V - V')|(u, v) \in E \; and \; u \in V'\}$. In order to get the starting node form S, the operation of $min(S) = min\{i|V_i \in V'\}$ is introduced. If S is empty, then min(S) is also empty. If S is singleton set, then min(S) equals the only element contained in S. verticesofall(S) denotes the set of all vertices connected by each edge in S.

Definition 1: Let $S_1(V_1', E_1', T_1', P_1', \alpha_1', \beta_1')$, $S_2(V_2', E_2', T_2', P_2', \alpha_2', \beta_2')$ are two connected sub-query of G(V, E, T, P, α, β), if $V_2' \subseteq (V - V_1')$ and existing a edge (u, v)$\in$ E such that u$\subseteq V_1'$ and v$\subseteq V_2'$, we call (S_1, S_2) a join pair.

In order to prevent duplicate join pairs from happening, we consider only join pair, (S_1, S_2), where S_2 only contain V_j With j large than any i with $V_i \in V_1'$. In order to complete achieve this, the operation of $W_i = \{V_j | j \leq i, V_j \in H\}$ is introduced. H is a connected and non-empty sub-query of V.

Every join pair is contained by CSLQS list. CSLQS are grouped by the size of the larger quantifier set in the join pair. We have $CSLQS_{[*,i]}$ = $\{CSLQS_{[j,i]} | j$ <i and $(i + j) \leq NumRel$ (T)$\}$. CSLQS $_{[0,1]}$ is used to represent a set of single quantifiers. The number of CSLQS is 2*NumRel (T) when NumRel (T) is even number or $2 * NumRel(T) + \lfloor NumRel(T)/2 \rfloor$ when NumRel (T) is odd number for a full connectivity query graph G.

There is dependence among CSLQS. The operation D (CSLQS$_{[x,y]}$) is introduced to solve the set of CSLQS depended by CSLQS$_{[x,y]}$. we can include $D(CSLQS_{[x,y]})$ = $\{CSLQS_{[a,b]} | y \geq a + b\}$. The number of D (CSLQS$_{[*,y]}$) denoted by $num\{D(CSLQS_{[*,y]})\}$ that equals number D(CSLQS$_{[1,y]}$) is $(y^2/4) + 1$ when y is even number or (y+1)(y-1)/4 when y is odd number.

Definition 2: Let $S_1(V_1', E_1', T_1', P_1', \alpha_1', \beta_1')$, S_2 (V_2', E_2', T_2', P_2', α_2', β_2') be two connected subquery of G (V, E, T, P, α, β). Ev and Ee are two given error bounds for relation table sizes and condition selectivities respectively. The pair of S_1, S_2 is similar sub-queries, if it meets these conditions as follows:

1. If there exists a one-to-one mapping f between S_1 and S_2, for any $x \in V_1'$ and $f(x) \in V_2'$, we have ComV (x,f(x)) = $|$sizeof(α_1'(x))-sizeof(α_2'(f(x)))$|$ /min(sizeof(α_1' (x)), sizeof(α_2'(f(x)))) < Ev.

2. If there exists a one-to-one mapping g between E_1' and E_2', for any $e \in E_1'$, $g(e) \in E_2'$, if vertices(e)={x,y}, then vertices(g(e))={f(x),f(y)}, we have ComE(e,g(e))= $|$selof(e)-selof(g(e))$|$/min(selof(e),selof(g(e)))<Ee.

2.2 Graph-driven Traversal

The top-down algorithm begins with a group consisting entirely of node, then considers generate all candidate logically equivalent multi-expression. This processing is called as logical-to-logical transformations. The traditional strategy relies on transformation rules, which do not consider the query graph to generate logical join pairs. Since the enumeration of this method is very fast, this is a very efficient strategy if the search space is dense, e.g. for clique queries. However, if the search space is spare, e.g. for chain queries, this method will product many logical join pairs which are not connected or which contain unconnected sub-queries, therefore, are not relevant for the solution.

The following statement gives a hint on how to construct the join pairs and similar sub-queries. Let S be a

© 2012 NSP
Natural Sciences Publishing Cor.

```
Partition (G, Ev, Ee)
Output: similar connected sub-queries and join pairs.
1:  initialize CSLQS= ∅ SSQ= ∅ iden=0
2:  N=NumRel (T)
//determine the similar vertices
3:  for i = 1 To N -1 do
4:    for id =0 to i -1 do
5:      if ComV(V_id, V_i)<Ev
6:          if not existing V_i in S_L of SSQ_1
7:           append V_i, V_id to S_L and S_R
8:          else
9:            append V_id to S_R corresponding V_i
// determine the join pairs and similar sub-queries
10: for all i in [N-1…0] descending
11:   append V_i to CSLQS_[0, 1]
12:   PairQueue + = {V_i}
13:  PairQueue + = MinOptimistic(G, V_i, W_i, SSQ, iden)
14: iden=1
15: for each S′_1 ∈ PairQueue
16:   S_2←S_2+CmpSub (G, S′_1, SSQ, iden)
17:   for each S′_2 ∈ S_2
18:     qs_2=max (S′_1, S′_2)
19:     qs_1=min (S′_1, S′_2)
20:     append (qs_1, qs_2) to CSLQS_[NumRel(qs1), NumRel(qs2)]
MinOptimistic (G, S, T, SSQ, iden)
Output: minimum cuts extended from S
1:  N= {ad(S) -T}
2:  R=NumRel (s) +1
3:  for all v ∈ N, v≠∅
//identifying if or not existing similar sub-query for S
4:   if iden=0
5:   for every similarity S′ of S in SSQ_R
6;     if exists node u in ad (S′) similarity to v and edge (v, x) x ∈ S edge
(u, y) y=f(x) ∈ S′
7:       ComE (edge (v, x), edge (u, y)) <Ee
8:       S_1←verticesofall (S) ∪ {v} and S_1′←verticesofall (S′) ∪ {u}
9:       if not existing S_1′ in S_L and S_1 in corresponding S_R of SSQ_R+1
10:          if not existing S_1 in S_L of SSQ_R+1
11:            append (S_1, S_1′) to SSQ_R+1
12:         else
13:            append S_1′ to S_R corresponding S_1
14:  do S←verticesofall (S) ∪ {v}
15:  return (S)
16:  MinOptimistic (G, S, N ∪ T, SSQ, AdjList, iden)
CmpSub (G, S, SSQ, iden)
Output: all connected subset S′ supplementing S
1:  T= {W_min(S) ∪ verticesofall(S)}
2:  N= {ad(S)-T}
3:  for all v_i ∈ N descending by i
4:   return (v_i)
5:   MinOptimistic(G, {v_i}, N ∪ T, SSQ, iden)
```

connected sub-query of a graph G and S′ be any sub-query of ad(S). Then S⋃S′ is connected. As a consequence, a connected sub-query can be enlarged by adding any subset of its neighborhood. Multiple similar sub-queries can also be enlarged by estimating the similarity of their adjoining vertices and edges.

Partition algorithm provides a skeleton framework how to generate the non-empty connected join pairs based on graph-traversal driven that accompany the measurement of similar sub-queries. We will use the non-empty connected join pairs to replace the logical-logical transformation through traditional transformational rule. The top-down enumeration with the optimized logical-logical transformation is called as TD_JP. The similar sub-queries are employed to avoid multiple plan construction, and therefore degrade the complexity degree for top-down algorithm.

For all elements of V, Partition firstly determines the similar vertices from line 3 to 9. Then it expands every element $\{v\}$ of V by calling a routine MinOptimistic that extends a given connected sub-query to bigger connected sub-query at line 13. For every constructed connected sub-query, CmpSub generates all connected sub-queries adjoining it at line 16. Line 17-20 adds every join pair constructed to CSLQS. MinOptimistic is an iteration function and mainly expands the node S by calculating the neighborhood ad(S). Line 5-13 in MinOptimistic expands the current pair of similar sub-queries by adding the similar adjoining edge.

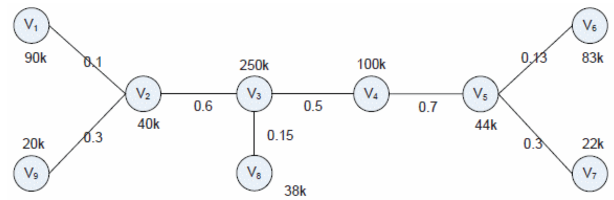

Figure 1 Query Graph G

Let us consider an example. Fig.1 shows a query graph using error bound parameters (Ev=0.25, Ee=0.3) with the table size and selectivity.

The similar sub-queries (SSQ) and connected join pairs grouped by the size of the larger quantifier set in the join pair (CSLQS) is given in Fig.2. SSQ_1 contains the similar vertices of G. Note that $\{(V_2), (V_5)\}$ are similar vertices in SSQ_1 and there is existing similar vertices V_1, V_2 in ad (V_2) and ad(V_5), respectively. The reason $\{(V_2, V_1), (V_5, V_6)\}$ is not contained by SSQ_2 is because edge (V_2, V_1) and edge (V_5, V_6) are not similar.

Through the graph in Fig.2 we also can see the structure of CSLQS constructed. The set of CSLQS connected by oblique line denotes logically equivalent multi- expression and will been used to logical-to-logical transformations. For example, we use $CSLQS_{[1,8]}$, $CSLQS_{[3,6]}$ and $CSLQS_{[4,5]}$ to logical express G with nine vertices. So the solution of G can be split three parts. Parallel Execution can be done on these devised parts. Note that $CSLQS_{[2,7]}$ is not included in $CSLQS_{[*,7]}$, because there is not existing connected join pair (S_1, S_2) where S_1 and S_2 have two and seven vertices respectively by the concrete implementation of Partition algorithm. So the CSLQS constructed by Partition algorithm makes the top-down dynamic programming not relying on transformation rule of traditional. It is optimal with respect to the join graph and avoids the Cartesian products which can extremely decreasing the search space. The set of CSLQS connected by horizontal line have same dependence set of CSLQS. For example, the CSLQS set comprised by triangle are depended by $CSLQS_{[*,4]}$.

3. Construction of Query Plan

3.1 Parallel Top-down Enumeration on CSLQS

In order to solve the solution of a sub-query of G using parallelize the top-down enumeration, we need allocate the set of $CSLQS_{[x,y]}$ that the sum of x and y equals NumRel(T) of the query G to different threads. We use the number of cores, num(cores), to denote the number of

© 2012 NSP
Natural Sciences Publishing Cor.

	S_L	S_R
SSQ_1	V_1	V_6, V_4
	V_2	V_5, V_8
	V_4	V_6
	V_5	V_8
	V_9	V_7
SSQ_2	V_1,V_2	V_6,V_5
	V_2,V_9	V_5,V_7
SSQ_3	V_1,V_2,V_9	V_6,V_5,V_7

Figure 2 SSQ and CSLQS of Query Graph G

TDP_CJP (G, Ev, Ee)
Output: the optimal query plan
1: N=num (cores)
2: CSLQS←Partition (G, Ev, Ee)
3: SSQMemo=∅
4: for each $CSLQS_{[x,\ y]}$ with x+y= NumRel (T)
5: AllocateT($CSLQS_{[x,y]}$, N)
6: for i←1 to N // N thread parallel implement
7: pool.SubmitJob
8: TD_CJP (CSLQS, group[i], $ThreadMemo_i$, SSQMemo)
9: pool.Sync ()
10:MergeAndPrunePlans(MEMO, $\{ThreadMemo_1 \ldots ThreadMemo_N\}$)
11:return MEMO

threads. However num(D($CSLQS_{[*,y]}$)) that equals number num (D ($CSLQS_{[1,y]}$)) is (y2/4) +1 when y is even number or $(y+1)(y-1)/4$ when y is odd number reduces with the decrease of y. we use D ($CSLQS_{[*,y]}$) to denote the workload of $CSLQS_{[*,y]}$. Because it will cause the imbalance workload among threads based on CSLQS allocation, we need refined allocation granularity. The balance workload among threads can be completed by using join pairs in CSLQS as allocation granularity. The concrete way as follow:

Each $CSLQS_{[xy]}$ that the sum of x and y equals NumRel(T) of the query G firstly is partitioned into num(cores) groups. Then the different $group_{[i]}$ in each $CSLQS_{[xy]}$ are allocated to $thread_{[i]}$. The high-level description of parallelize top-down enumeration with the optimized logical-logical transformation with CSLQS is given by the following TDP_CJP algorithm.

The function of AllocateT achieves the distribution with balance workload among threads at line 5. The TD_CJP algorithm has three parameters, CSLQS, group[i], ThreadMemo $_i$, and SSQMemo. CSLQS mainly be used to realize the optimization of logical-logical transformation. Group[i] is the allocated execution set of join pairs of $thread_i$. ThreadMemo $_i$ is applied to store the optimal query plan of group [i]. SSQMemo contain the partial solution by SSQ. MergeAndPrunePlans function (line 9) selects the optimal query plan among partial solutions.

3.2 Optimal TDP Algorithm Based on SSQ

In this subsection, we optimize TDP_CJP through considering SSQ. Our approach involves two steps:

a) Generating the sub-query plan of S_L in SSQ by TD _ CJP algorithm

b) Re-using the sub-query plan of S_L to the similar sub - query S_R in SSQ.

The structure of SSQ and CSLQS are different. In order to construct the sub-query plan in SSQ by TD_CJP algorithm, we re-consider expressing the sub-query of S_L using connected join pairs. This process can achieve by adding a JPS_L column. JPS_L column contains all the logical express of S_L. When the sub-query of S_L only contains a node, the JPS_L contains a node. Otherwise, the sub-query q of S_L in SSQ_i can be logical expressed via traveling every sub-query q_x and q_y of S_L in SSQ_x and SSQ_y respectively that the sum of x and y equals i and verticesofall (q) equals the sum of verticesofall(q_x) and verticesofall (q_y). For example, the sub-query (V_1, V_2, V_9) of S_L in SSQ_3 can be expressed (V_1, V_2V_9) and (V_9, V_1V_2). Through this method, the constructed SSQ in Fig.2 can be reconstructed as Fig.3.

	S_L	JPS_L	S_R
SSQ_1	V_1	(V_1)	V_6, V_4
	V_2	(V_2)	V_5, V_8
	V_4	(V_4)	V_6
	V_5	(V_5)	V_8
	V_9	(V_9)	V_7
SSQ_2	V_1,V_2	(V_1,V_2)	V_6,V_5
	V_2,V_9	(V_2,V_9)	V_5,V_7
SSQ_3	V_1,V_2,V_9	(V_1, V_2V_9)	V_6,V_5,V_7
		(V_9, V_1V_2)	

Figure 3 Reconfigurable SSQ

The high-level description of the optimized TDP_CJP with SSQ is given by the following OTDP_CJP algorithm. Note that the major objective to first TD_CJP (line 13) is to obtain the query plan of JPS_L but the optimal plan for S_L. So the multiple logical express of S_L can not be executed in parallel. On the contrary, the second (line 22) can be executed in parallel.

The function of Reconstruct mainly achieves all S_L in SSQ logical express by adding a JPS_L column (line 3). Line 4-7 uses CSSQ to preserve all join pairs in SSQ according to the method of constructing CSLQS. The purpose of this is mainly to fulfill a consistent approach of two TD_CJP. MaxIndex (SSQ) solves the max index of

© 2012 NSP
Natural Sciences Publishing Cor.

```
OTDP_CJP (G, B, Ev, Ee)
Output: the optimal query plan with cost not exceeding B
1: N=num (cores)
2: SSQ, CSLQS←Partition (G, Ev, Ee)
3: SSQ←Reconstruct(SSQ)
4: for each join pair (S′1, S′2) in JPS_L
5:   qs2=max (S′1, S′2)
6:   qs1=min (S′1, S′2)
7:   append (qs1, qs2) to CSSQ[NumRel(qs1), NumRel(qs2)]
8: m=MaxIndex (SSQ)
9: SSQMemo=∅
//Constructing the query plan for all join pairs in JPS_L of SSQ
10: SSQ_m existing logical express JP without query plan in
SSQMemo
11:   for each logical express JP
12:     JPMemo=∅
13:     TD_CJP (CSSQ, JP, JPMemo, SSQMemo)
14:    SSQMemo ←SSQMemo∪JPMemo
15:  m=m-1
16: SSQMemo←SSQMemo∪Reuse_Plan(SSQ, SSQMemo)
//Constructing global solution
17: for each CSLQS[x, y] with x+y= NumRel (T)
18:    AllocateT(CSLQS[x, y], N)
19: for i←1 to N // N thread parallel implement
20: pool.SubmitJob
21:   ThreadMemo_i=∅
22:   TD_CJP (CSLQS, group[i], ThreadMemo_i, SSQMemo)
23: pool.Sync ()
24: MergeAndPrunePlans (MEMO, {ThreadMemo_1…
ThreadMemo_N})
25: return MEMO
```

SSQ at line 8. Line 11-15 constructs the query plans for all join pairs in JPS$_L$ of SSQ and uses SSQMemo to contain the constructed query plans. SSQMemo will be reviewed whether or not contain the plan before solving the plan of anyone join pair. The function of Reuse_Plan completes the reusing of query plans contained in SSQMemo through the similar sub-query S$_L$ and S$_R$ at line 16.

4. Performance Analysis

All the experiments were performed on a Windows Vista PC with two Intel Xeon Quad Core E540 1.6GHz CPUs (=8 cores) and 8GB of physical memory. Each CPU has two 4Mbyte L2 caches, each of which is shared by two cores. The experimental parameters and their values are illustrated by Table 1.

In the first experiment, we compares the running time of TTD, TD_CJP and IDP algorithms by changing the number of quantifiers for varying query graphs in Figure 4. The running time consists of two parts, optimization time used to construct query plan and execution time for query plan. Execution time reflects the quality of constructed query plan. None of these are parallel algorithms. We wanted to answer that besides clique queries the algorithm TD_CJP

Type	Enumeration Style	
Bottom-up DP	Parallel Size-Driven	PDPsva
Iterative DP	Randomized query	IDP
Top-down DP	traditional logical-to-logical transformation	TTP
	logical-to-logical transformation based CSLQS	TD_CJP
	paralleled TD_CJP	TDP_CJP
	optimized TDP_CJP based SSQ	OTDP_CJP

Table 1 Experimental Parameters

based on the optimization of CSLQS significantly outperforms the conventional TTD and IDP algorithms.

Figure 4 (a) compares the running time for clique queries. As illustrated in Figure 4 (a), the total running time increases as the number of relations is increased. TTD and TD_CJP have the same execution time because they are exhaustive search DP algorithms and can construct the best query plan. Because the optimization of CSLQS for clique queries is unnecessary, the optimization time for TD_CJP is longer than TTD. IDP algorithm has the shortest optimization time due to combining randomized and DP algorithm. However, it cannot guarantee an optimal query plan. So the execution time is the longest. Figure 4 (b) compares the running time for star queries. Because the optimization of CSLQS for star queries can avoid constructing logical join pairs which are not connected, the optimization time for TD_CJP is shorter than TTD. Figure 4 (c) shows similar experiments with Figure 4 (b). Figure 4 shows that TD_CJP algorithm is better than TTD and IDP, apart from clique queries. This shows the optimization of CSLQS is effective for star and cycle queries.

In the second experiment, we compared OTDP_CJP, TDP_CJP and PDPsva algorithms in Figure 5. They are parallel algorithms. From the running time of Figure 5, it should be noted the parallel algorithms are superior to TTD, IDP and TD_CJP algorithm. By Figure 5 we wanted to answer that besides clique queries the algorithm OTDP_CJP based on the optimization of CSLQS and SSQ significantly outperforms the TDP_CJP and PDPsva algorithms.

Figure 5(a) compares the running time of clique queries. As illustrated in Figure 5 (a), the optimization time for PDPsva is the shortest because the optimization of CSLQS for clique queries is unnecessary. However, OTDP_CJP cuts the optimization time by using SSQ. The optimization of OTDP_CJP is shorter than TDP_CJP. For execution time, PDPsva equals TDP_CJP. It should be noted that the execution time of OTDP_CJP is longest. The query plan constructed by OTDP_CJP is the ideal plan, not the best plan. Figure 5 (b) compares the running time for star queries. As it shown the running time of OTDP_CJP is the shortest. Figure 5 (c) shows similar experiments with Figure 5 (b). Figure 5 shows OTDP_CJP algorithm is optimal for star and cycle queries.

© 2012 NSP
Natural Sciences Publishing Cor.

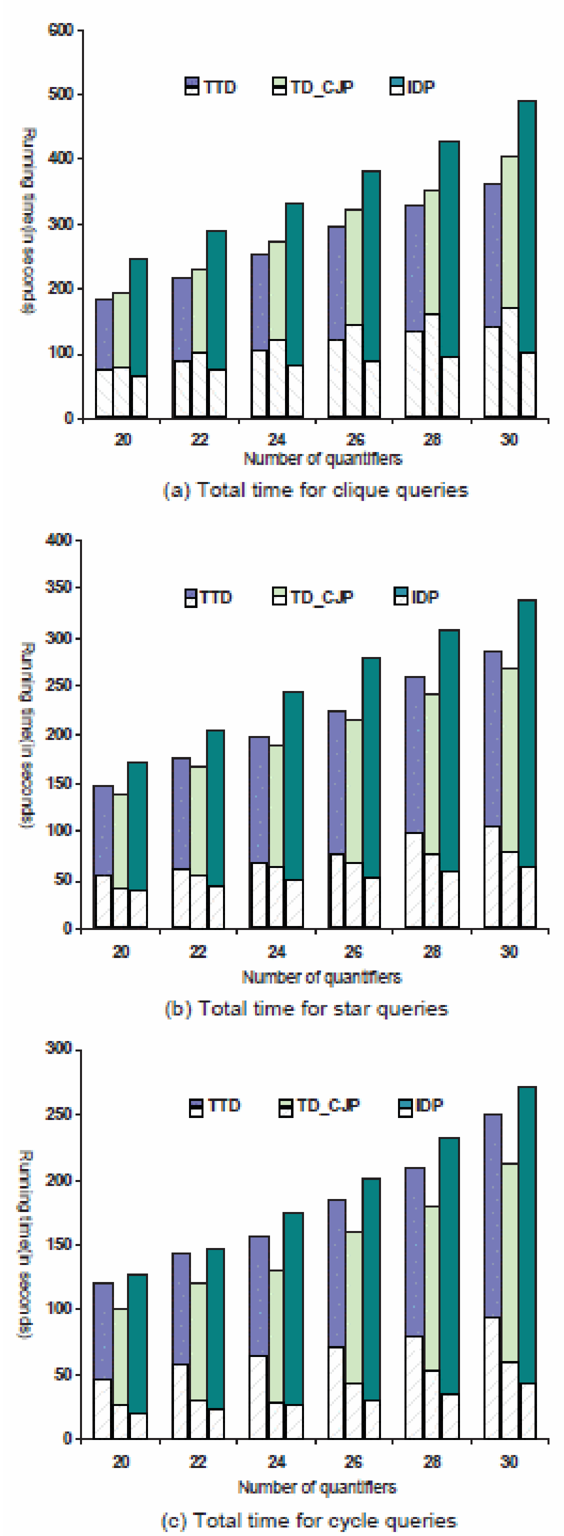

Figure 4 Total time for single thread algorithms by varying number of quantifiers

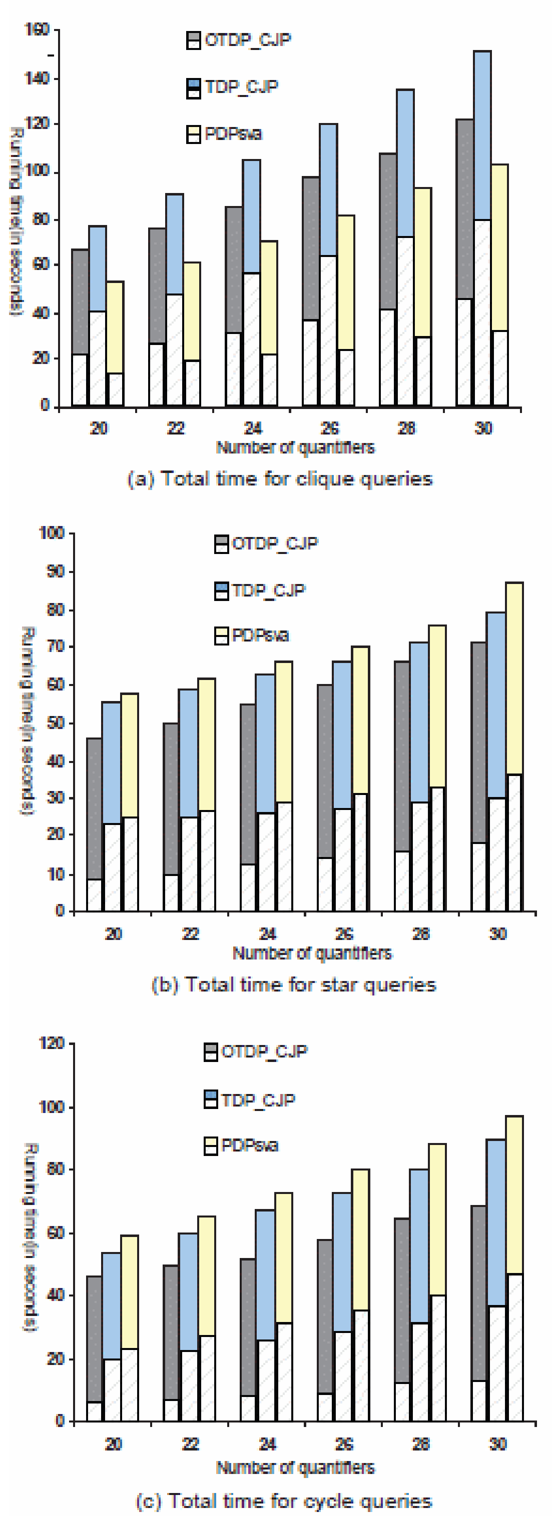

Figure 5 Total time for single thread algorithms by varying number of quantifiers

© 2012 NSP
Natural Sciences Publishing Cor.

5. Conclusion and Future Work

In this paper, parallelizing top-down dynamic programming query based on CMP is completed by three phases. In the first phase, through graph-driven traversal we constructed connected join pairs used to optimal the traditional transformation rule of logical-to-logical and simultaneously similar sub-queries employed to reduce multiple plan construction for connected join pairs. In the second phase, based on the reconfigurable SSQ we use TD_CJP algorithm to construct the query plan for all the logical express of SL in SSQ and re-use these constructed plan to the similar sub-queries. Finally, TD_CJP algorithm is applied once more to solve the global solution based on the CSLQS and query plans as a result of the second phase in parallel. By implementing our framework and analyzing the experiment results, OTDP_CJP gracefully degrade the complexity degree for top-down join enumeration with large number, impressive gains in the performance.

Future work is still needed in expanding our multi-threaded cluster partition and join strategy to examine performance on other multithreaded processors and to support other operations.

References

[1] W.S. Han,W. Kwak, J. Lee, G. M. Lohman and V. Markl, Proc. the VLDB Endowment, 188 (2008).

[2] G. Moerkotte and T. Neumann, Proc. the VLDB Endowment, 930 (2006).

[3] G. Moerkotte and T. Neumann, Proc. of the ACM SIGMOD international conference on Management of data, 539 (2008).

[4] John L. Hennessy and J.L. Patterson, In: Computer Architecture: A Quantitative Approach, Nate McFadden (Eds.), 335 (ELSEVIER, New York, 2007).

[5] K. Donald and S. Konrad, Proc. ACM Trans. on Database Systems, 2000 (1998).

[6] Arun N. Swami and G. Anoop, Proc. of the ACM SIGMOD international conference on Management of data, 8 (1988).

[7] S. Michael, M. Guido and K.A lfons, Proc. the VLDB Endowment, 191 (1997).

[8] B. Kristin, C.F.Michael and E.I. Yannis, Proc. the Fourth International Conference on Genetic Algorithms, 400 (1991).

[9] Q. Zhu, Y.Y. Tao, and Z. Calisto, Knowledge and Information Systems, (Springer, New York, 2005).

[10] V.V. Meduri, http://scholarbank.nus.edu.sg/bitstream/handle/10635/20999/report.pdf?sequence=1, (2011).

[11] M. B. Pam, Proc. the ACM/IEEE on SuperComputing, 15 (2008).

[12] P. Stenstrom, Proc. Parallel and Distributed Symposium, 14 (2007).

[13] L. Shapiro, Proc. the International Database Engineering & Applications Symposium, 20 (2001).

[14] John W. Raymond and W. Peter, Computer-Aided Molecular Design, (Springer, New York, 2002).

First Author was born in Heilongjiang of China in Dec 1979 and received the Ph.D. degree at the Department of Computer Science and technology, Jilin University. His current main research interests include Query Optimization, Web Intelligence and Ontology Engineering and Information integration. He is a member of System Software Committee of China's Computer Federation. More than 20 papers of him were published in key Chinese journals or international conferences, 10 of which are cited by SCI/EI.

Second Author obtained her B.Sc. degree from Harbin Normal University. Currently she is a M.Sc. candidate at the Department of Computer Science and technology, Jilin University. Main research area covers Database Theory, Machine Learning, Data Mining and Web Mining, Web Search Engines, Web Intelligence.

© 2012 NSP
Natural Sciences Publishing Cor.