Nested Loop - Parallel Cursor, Parallel Processor techniques

Problem Statement: It is always a good practice to avoid nested loops.
I want to perform total quantities when sales orders match in VBAP & VBAK tables.
Below is the logic

```
" Select all data from VBAP
SELECT * FROM vbap INTO TABLE lt_vbap.

" Select all data from VBAK
SELECT * FROM vbak INTO TABLE lt_vbak.

" Initialize accumulators
CLEAR: lv_total_quantity.

" Nested Loop Processing
LOOP AT lt_vbap INTO wa_vbap.
  LOOP AT lt_vbak INTO wa_vbak WHERE vbeln = wa_vbap-vbeln.

    lv_total_quantity = lv_total_quantity + wa_vbap-kwmeng.

ENDLOOP.
```

Now, this is the main pgm, we will do nested loop, parallel cursor and parallel processor and find their execution time

```
REPORT znested_loop.

DATA: lv_start_time     TYPE i,
      lv_end_time       TYPE i,
      nested_exec_time  TYPE i,
      par_cur_exec_time TYPE i,
      pp_exec_time      TYPE i.

PERFORM nested_loop.
PERFORM parallel_cursor.
PERFORM parallel_processing.

WRITE: / '----------------------------------------------------|'.
WRITE: / '| Execution Method         | Time (ms)              |'.
WRITE: / '----------------------------------------------------|'.
WRITE: / '| Nested Loop              |', nested_exec_time,   '|'.
WRITE: / '| Parallel Cursor          |', par_cur_exec_time,  '|'.
WRITE: / '| Parallel Processing      |', pp_exec_time     ,  '|'.
WRITE: / '----------------------------------------------------|'.
```

This is the code for a nested loop.

```
FORM nested_loop .
```

```
* WAIT UP TO 2 SECONDS.
  " Start Timer
  GET RUN TIME FIELD lv_start_time.
  DATA: lt_vbap           TYPE TABLE OF vbap,           " Table for VBAP data
        lt_vbak           TYPE TABLE OF vbak,           " Table for VBAK data
        wa_vbap           TYPE vbap,                    " Work area for VBAP
        wa_vbak           TYPE vbak,                    " Work area for VBAK
        lv_total_quantity TYPE kwmeng.        " Total quantity accumulator

  " Select all data from VBAP
  SELECT * FROM vbap INTO TABLE lt_vbap.

  " Select all data from VBAK
  SELECT * FROM vbak INTO TABLE lt_vbak.

  " Initialize accumulators
  CLEAR: lv_total_quantity.

  " Nested Loop Processing
  LOOP AT lt_vbap INTO wa_vbap.
    LOOP AT lt_vbak INTO wa_vbak WHERE vbeln = wa_vbap-vbeln.

      lv_total_quantity = lv_total_quantity + wa_vbap-kwmeng.

    ENDLOOP.
  ENDLOOP.

  " Display Results After Processing
  WRITE: / 'NS Total Quantity:', lv_total_quantity.

  " End Timer
  GET RUN TIME FIELD lv_end_time.

  " Calculate execution time in milliseconds
  nested_exec_time = lv_end_time - lv_start_time.

ENDFORM.
```

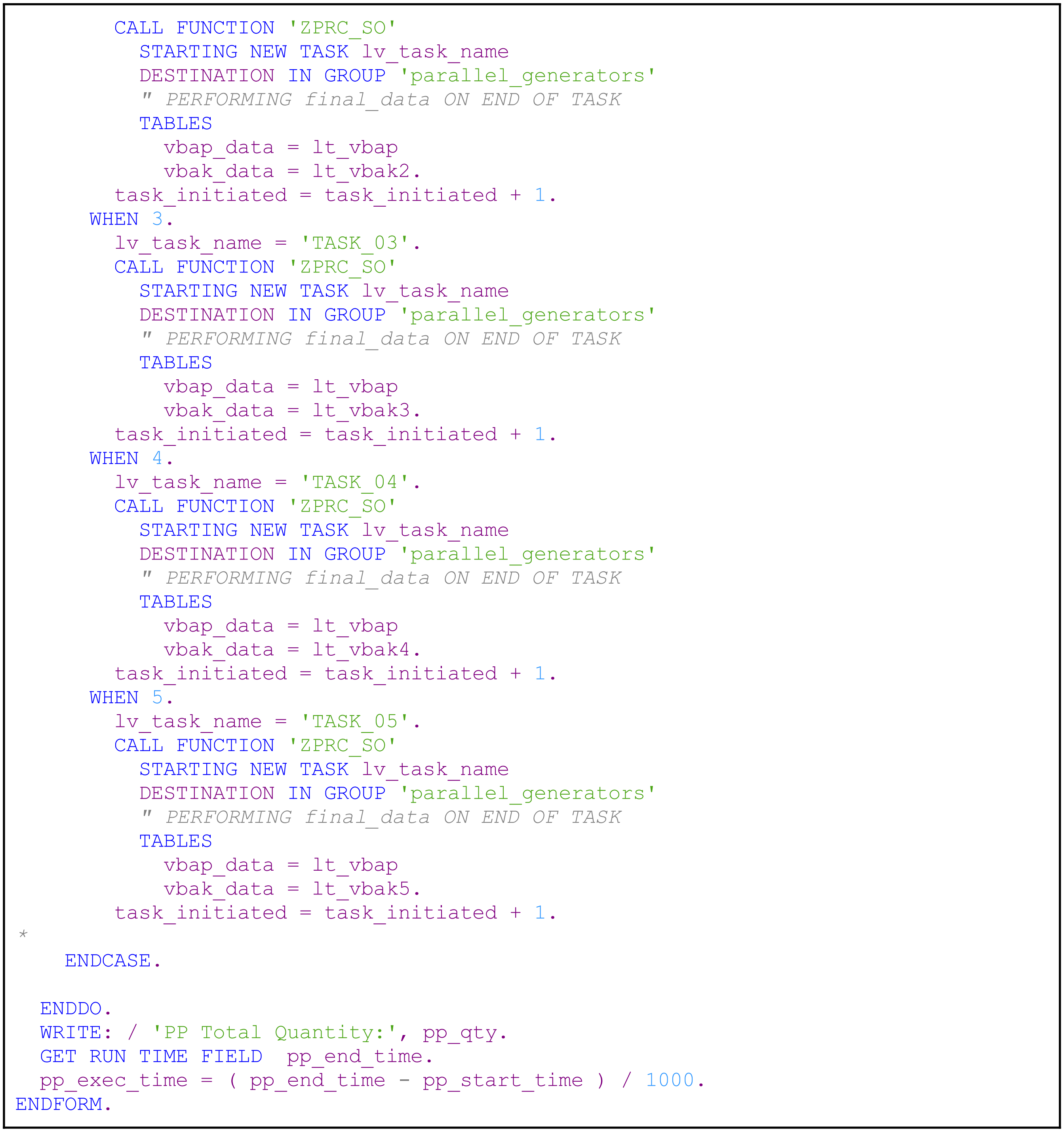
This is the code for parallel cursor. Kindly note, there is no second loop (or inside loop). Instead of the second loop, we are reading with a key. Before reading, both the tables VBAP & VBAK should be sorted on the key ascending.

Whenever the key matches only, the inside logic is performed.

```
FORM parallel_cursor .
  DATA: lt_vbap_data   TYPE TABLE OF vbap,       " Table for VBAP data
        lt_vbak_data   TYPE TABLE OF vbak,       " Table for VBAK data
        ls_vbap        TYPE vbap,                     " Work area for VBAP
        ls_vbak        TYPE vbak,                     " Work area for VBAK
        total_quantity TYPE kwmeng,            " Total quantity accumulator
        start_time     TYPE i,                    " Start time
        end_time       TYPE i,                      " End time
        vbak_pointer   TYPE sy-tabix.             " Pointer for VBAK.

  " Select all data from VBAP and VBAK
  GET RUN TIME FIELD start_time.
  SELECT * FROM vbap INTO TABLE lt_vbap_data.
  SELECT * FROM vbak INTO TABLE lt_vbak_data.

  " Sort both tables by VBELN (required for parallel cursor)
  SORT lt_vbap_data BY vbeln.
  SORT lt_vbak_data BY vbeln.

  " Initialize accumulators
  CLEAR: total_quantity.

  " Process using Parallel Cursor
  LOOP AT lt_vbap_data INTO ls_vbap.
    " Use READ TABLE with KEY to find matching record in VBAK
    READ TABLE lt_vbak_data INTO ls_vbak WITH KEY vbeln = ls_vbap-vbeln.

    IF sy-subrc = 0.
      " Matching record found in VBAK
      " 1. Accumulate Total Quantities
      total_quantity = total_quantity + ls_vbap-kwmeng.
    ENDIF.
  ENDLOOP.
  " Display Results After Processing
  WRITE: / 'PC Total Quantity:', total_quantity.
  GET RUN TIME FIELD end_time.

  " Calculate execution time in milliseconds
  par_cur_exec_time = ( end_time - start_time ) / 1000.

ENDFORM.
```

Now, coming to Parallel Processing. PP is splitting the data into smaller chunks (say 5 here) and submitting 5 different jobs. Once all the 5 jobs complete, we have to consolidate the result. Below is the logic for the code.

```
FORM parallel_processing .
  TYPES:
    BEGIN OF ty_vbak_vbap,
      vbeln  TYPE vbak-vbeln,
      kwmeng TYPE vbap-kwmeng,
    END OF ty_vbak_vbap.

  DATA: lt_vbak         TYPE TABLE OF vbak,
        lt_vbap         TYPE TABLE OF vbap,
        ls_vbak         TYPE vbak,
        total_quantity TYPE kwmeng,      " Accumulated quantity
        match_count     TYPE i,              " Match counter
        task_count      TYPE i,               " Counter for the background tasks
        chunk_size     TYPE i VALUE 10,     " Size of each chunk
        start_index    TYPE i VALUE 1,     " Index for chunking
        end_index      TYPE i,               " End index for chunking
        pp_start_time  TYPE i,                     " Start time
        pp_end_time    TYPE i.                       " End time


  GET RUN TIME FIELD  pp_start_time.

  DATA: lt_vbak1        TYPE TABLE OF vbak,
        lt_vbak2        TYPE TABLE OF vbak,
        lt_vbak3        TYPE TABLE OF vbak,
        lt_vbak4        TYPE TABLE OF vbak,
        lt_vbak5        TYPE TABLE OF vbak,
        lv_task_name    TYPE c LENGTH 20,
        lv_chunk_size   TYPE i,
        lv_counter      TYPE i,
        task_processed TYPE i,
        task_initiated TYPE i.

  " Fetch records into lt_vbap and lt_vbak
  SELECT * FROM vbap INTO TABLE lt_vbap UP TO 100 ROWS.
  SELECT * FROM vbak INTO TABLE lt_vbak UP TO 100 ROWS.

  " Split vbak into 10 subsets
  lv_chunk_size = ceil( lines( lt_vbak ) / 5 ). " Divide total rows into 10 parts

  LOOP AT lt_vbak INTO ls_vbak.
    lv_counter = lv_counter + 1.

    " Append to the appropriate subset
    CASE ( lv_counter - 1 ) / lv_chunk_size + 1.
      WHEN 1. APPEND ls_vbak TO lt_vbak1.
      WHEN 2. APPEND ls_vbak TO lt_vbak2.
      WHEN 3. APPEND ls_vbak TO lt_vbak3.
      WHEN 4. APPEND ls_vbak TO lt_vbak4.
      WHEN 5. APPEND ls_vbak TO lt_vbak5.
    ENDCASE.
  ENDLOOP.

  " Submit 10 background jobs
  DO 10 TIMES.
    CASE sy-index.
      WHEN 1.
        lv_task_name = 'TASK_01'.
        CALL FUNCTION 'ZPRC_SO'
          STARTING NEW TASK lv_task_name
          DESTINATION IN GROUP 'parallel_generators'
          " PERFORMING final_data ON END OF TASK
          TABLES
            vbap_data = lt_vbap
            vbak_data = lt_vbak1.
        task_initiated = task_initiated + 1.

      WHEN 2.
        lv_task_name = 'TASK_02'.
```



```
        CALL FUNCTION 'ZPRC_SO'
          STARTING NEW TASK lv_task_name
          DESTINATION IN GROUP 'parallel_generators'
          " PERFORMING final_data ON END OF TASK
          TABLES
            vbap_data = lt_vbap
            vbak_data = lt_vbak2.
        task_initiated = task_initiated + 1.
      WHEN 3.
        lv_task_name = 'TASK_03'.
        CALL FUNCTION 'ZPRC_SO'
          STARTING NEW TASK lv_task_name
          DESTINATION IN GROUP 'parallel_generators'
          " PERFORMING final_data ON END OF TASK
          TABLES
            vbap_data = lt_vbap
            vbak_data = lt_vbak3.
        task_initiated = task_initiated + 1.
      WHEN 4.
        lv_task_name = 'TASK_04'.
        CALL FUNCTION 'ZPRC_SO'
          STARTING NEW TASK lv_task_name
          DESTINATION IN GROUP 'parallel_generators'
          " PERFORMING final_data ON END OF TASK
          TABLES
            vbap_data = lt_vbap
            vbak_data = lt_vbak4.
        task_initiated = task_initiated + 1.
      WHEN 5.
        lv_task_name = 'TASK_05'.
        CALL FUNCTION 'ZPRC_SO'
          STARTING NEW TASK lv_task_name
          DESTINATION IN GROUP 'parallel_generators'
          " PERFORMING final_data ON END OF TASK
          TABLES
            vbap_data = lt_vbap
            vbak_data = lt_vbak5.
        task_initiated = task_initiated + 1.
*
    ENDCASE.

  ENDDO.
  WRITE: / 'PP Total Quantity:', pp_qty.
  GET RUN TIME FIELD  pp_end_time.
  pp_exec_time = ( pp_end_time - pp_start_time ) / 1000.
ENDFORM.
```

The DESTINATION can be got from Table RZLLITAB as below

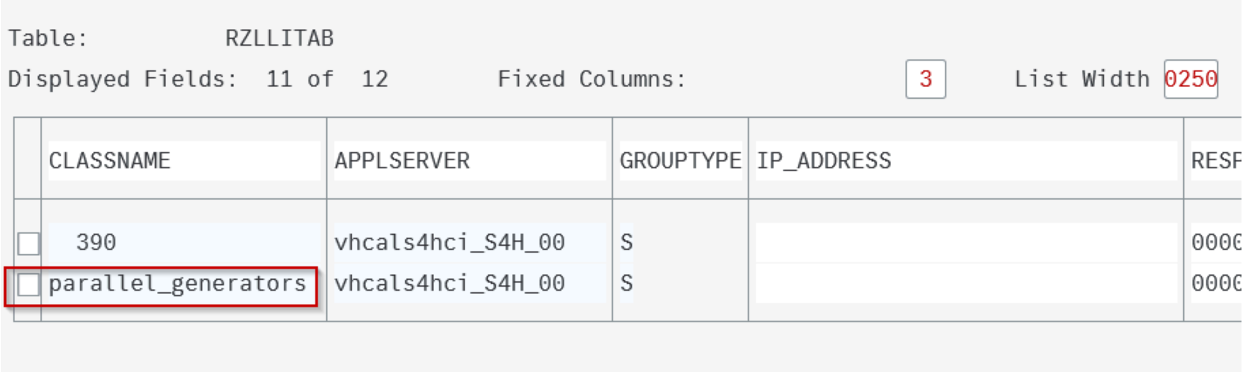
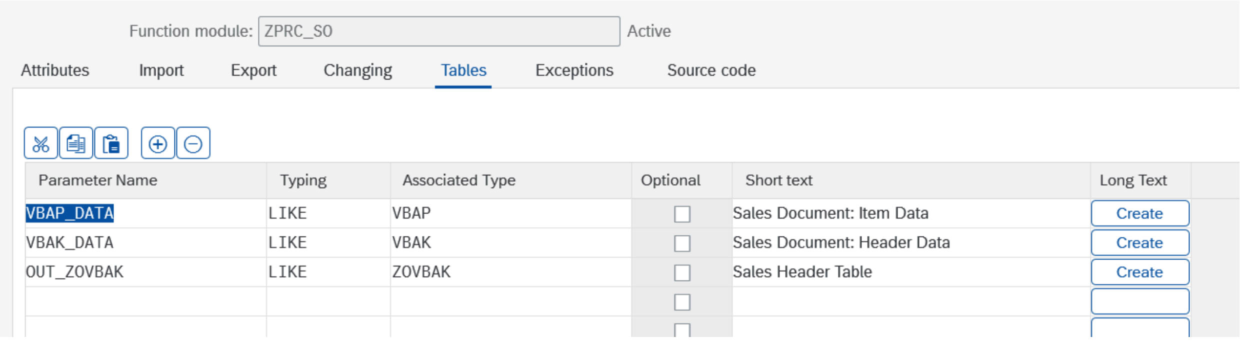


Table: RZLLITAB

Displayed Fields: 11 of 12 Fixed Columns: 3 List Width 0250

	CLASSNAME	APPLSERVER	GROUPTYPE	IP_ADDRESS	RESP
□	390	vhcals4hci_S4H_00	S		0000
□	parallel_generators	vhcals4hci_S4H_00	S		0000

Now, let us create a function ZPRC_SO



Function module: ZPRC_SO Active

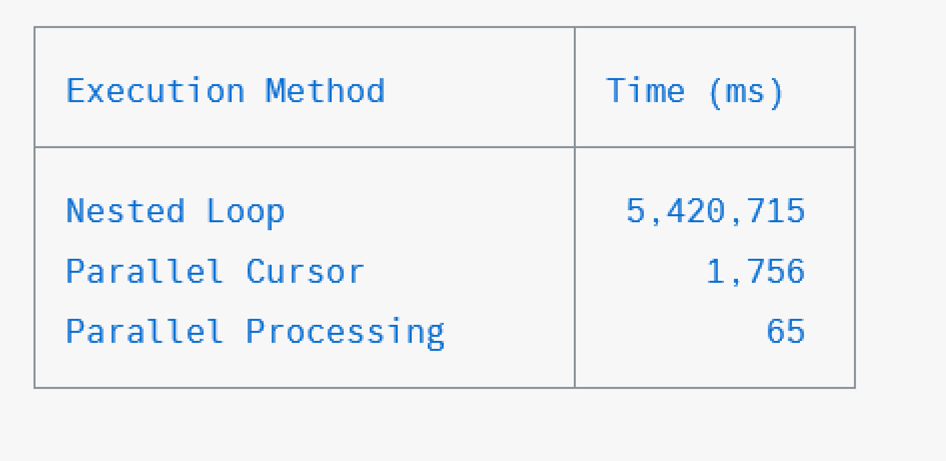
Parameter Name	Typing	Associated Type	Optional	Short text	Long Text
VBAP_DATA	LIKE	VBAP	□	Sales Document: Item Data	Create
VBAK_DATA	LIKE	VBAK	□	Sales Document: Header Data	Create
OUT_ZOVBAK	LIKE	ZOVBAK	□	Sales Header Table	Create
			□		
			□		

I have used VBAP_DATA & VBAK_DATA as Input tables and Table OUT_ZOVBAK as my output table. But, there are many ways to get your output. Here, I have written my output to a record in a configure kind of table.

```
FUNCTION zprc_so.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  TABLES
*"      VBAP_DATA STRUCTURE  VBAP
*"      VBAK_DATA STRUCTURE  VBAK
*"      OUT_ZOVBAK STRUCTURE  ZOVBAK
*"----------------------------------------------------------------------

  DATA: ls_vbap        TYPE vbap,
        ls_vbak        TYPE vbak,
        total_quantity TYPE kwmeng.

  " Copy the input data into local variables
  DATA(lt_vbap_data) = vbap_data[].
  DATA(lt_vbak_data) = vbak_data[].

  SORT lt_vbap_data BY vbeln DESCENDING.
  SORT lt_vbaK_data BY vbeln DESCENDING.

  CLEAR: total_quantity.
  LOOP AT lt_vbap_data INTO ls_vbap.

    READ TABLE lt_vbak_data INTO ls_vbak WITH KEY vbeln = ls_vbap-vbeln.
    IF sy-subrc = 0.
      total_quantity = total_quantity + ls_vbap-kwmeng.
    ENDIF.

  ENDLOOP.

  DATA: ls_kwmeng TYPE kwmeng.
  SELECT SINGLE kwmeng FROM zovbak_fk INTO @ls_kwmeng WHERE vbeln = '0000000007'.
  IF sy-subrc = 0.
    ls_kwmeng = ls_kwmeng + total_quantity.

    UPDATE zovbak_fk
    SET kwmeng = ls_kwmeng
    WHERE vbeln = '0000000007'.
  ENDIF.

ENDFUNCTION.
```

To find the execution of the Parallel Process (Tasks), use TCODE SM50 or SM66

Results:



Execution Method	Time (ms)
Nested Loop	5,420,715
Parallel Cursor	1,756
Parallel Processing	65

Parallel Processing is far more efficient, but coding is cumbersome (but worth it).

Happy learning 🙂