Hacking Oracle's Memory - About Internals & Troubleshooting

Stefan Koehler
Freelance Consultant (Soocs)
Coburg, Germany

Keywords
Oracle memory, SGA, Granules, Shared memory, X$ table, Shared pool, Sub-Pool, Duration, PGA, Heap memory, DTrace, Memory allocation, Oradebug

Abstract of the corresponding DOAG talk
Oracle externalizes parts of its memory structures via X$ tables and provides various memory dump possibilities. This session provides insights into Oracle's SGA / PGA memory structure internals and points out important changes with 12c. It includes several deep-dive demos and a real life case about troubleshooting a PGA memory leak (analyzing, capturing and sourcing PGA memory allocations).

About the author
Stefan Koehler follows his passion about Oracle since +13 years and specialized in Oracle performance tuning, especially in Oracle cost based optimizer (CBO) and database internal related topics. Nowadays he primarily works as a freelance Oracle database performance consultant in large mission critical environments (e.g. +10 TB databases) for market-leading companies or multi-billion dollar enterprises in various industries all over the world. He usually supports his clients in understanding and solving complex Oracle performance issues and troubleshooting nontrivial database issues on short-term contracting basis. He is also a proud member of the OakTable Network.

Disclaimer
Almost everything in this paper is based on research and testing. Test it yourself – with your release and operating system – always! Do not trust Stefan or anybody else! ☺

X$ tables - A window into Oracle’s memory structure

Oracle’s X$ tables are basically C memory structures which are read by using a fixed table row source function in the SQL execution plan. However some X$ tables are not just a plain representation of C memory structures but also have some associated additional (helper) functions that retrieve the data (e.g. from the control file) first and then copy the needed data into a memory format corresponding to the X$ table – afterwards the usual fixed table row source function kicks in.

Plain representation of C memory structures

The X$ table x$ksuse (that is used by gv$session) is an example of a plain representation of a C memory structure. The following demo is run on Oracle 12.1.0.2 and SunOS SOL 5.11 11.2 (x86).

```
SYS@S12DB:8> select view_definition from v$fixed_view_definition where view_name =
'GV$SESSION';
VIEW_DEFINITION
---------------------
select s.inst_id,s.addr,s.indx,s.ksuseser,s.ksuudses,s.ksusepro,s.ksuudlui
…
decode(bitand(s.ksuseflg2,64),64,'TRUE','FALSE'),
```

The problem with view v$fixed_view_definition is that column view_definition is defined as VARCHAR2(4000) and may not represent the whole view definition. However the fixed view definitions are hard-coded into the Oracle binary and can also be extracted in full version from there.

```
shell> strings $ORACLE_HOME/bin/oracle | grep "select
s.inst_id,s.addr,s.indx,s.ksuseser,s.ksuudses,s.ksusepro,s.ksuudlui"
select s.inst_id,s.addr,s.indx,s.ksuseser,s.ksuudses,s.ksusepro,s.ksuudlui
…
from x$ksuse s, x$ksled e, x$kslwt w where bitand(s.ksspaflg,1)!=0 and bitand(s.ksuseflg,1)!=0
and s.indx=w.kslwtsid and w.kslwtevt=e.indx
```

The fixed view gv$session is based on the X$ tables x$ksuse, x$ksled, x$kslwt. The following demonstration is based on X$ table x$ksuse which is basically an acronym for "[K]ernel Layer, [S]ervice Layer, [U]ser Management, [SE]ssions Info" (MOS ID #22241.1). [1]

```
SYS@S12DB:8> show parameter sessions
NAME        TYPE        VALUE
----------- ----------- ------
sessions    integer     474

SYS@S12DB:8> select s.addr, s.indx, s.ksuseser, s.ksuudsna from x$ksuse s;
ADDR                   INDX   KSUSESER KSUUDSNA
---------------- ---------- ---------- ------------------------------
00000001B0C83A80          1      42091 SYS
00000001B0C819B0          2      16106 SYS
…
00000001B0DC9BB0        474          0
474 rows selected.

-------------------------------------------------------------------
| Id  | Operation         | Name     | E-Rows |E-Bytes| Cost (%CPU)|
-------------------------------------------------------------------
|   0 | SELECT STATEMENT |          |        |       |     1 (100)|
|   1 |  FIXED TABLE FULL| X$KSUSE  |474     |  8532 |     0    (0)|
-------------------------------------------------------------------
```

The X$ table x$ksuse is just a memory array of session information with a defined size by the init parameter sessions. The memory address of each array entry can be obtained with help of column ADDR. Tanel Poder has written a script called fcha.sql ([F]ind [CH]unk [A]ddress) [2] that determines in which heap (e.g. UGA, PGA, SGA) a memory address is located, but be careful because this script is based on the X$ table x$ksmsp, which can cause severe shared pool latch contention.

```
SYS@S12DB:8> @/Users/iStefan/SQLs/10_talks/DOAG2016_161115/tanel_find_heap_mem.sql 1B0C83A80
LOC KSMCHPTR          KSMCHIDX   KSMCHDUR   KSMCHCOM          KSMCHSIZ   KSMCHCLS KSMCHTYP  KSM
--- ---------------- ---------- ---------- ---------------- ---------- -------- --------- ---
SGA 00000001B00CD000 1          1          permanent memor   15707088   perm     0         00
```

The array entries of the X$ table x$ksuse are stored in a permanent memory chunk in the SGA. Permanent memory areas are allocated at instance startup and contain structures for processes, sessions or segment arrays. The array entry memory addresses are fixed and do not change regardless of how often a query is run on the X$ table x$ksuse.

Additional (helper) functions and representation of C memory structures

The X$ table x$kccle (that is used by gv$log) is an example of a X$ table that is based on additional (helper) functions. The following demo is run on Oracle 12.1.0.2 and SunOS SOL 5.11 11.2 (x86).

```
SYS@S12DB:8> select view_definition from v$fixed_view_definition where view_name = 'GV$LOG';
VIEW_DEFINITION
---------------------
select le.inst_id, le.lenum, le.lethr, le.leseq
…
from x$kccle le, x$kccrt rt where le.ledup!=0 and le.lethr=rt.rtnum and le.inst_id =
rt.inst_id
```

The fixed view gv$log is based on the X$ tables x$kccle and x$kccrt. The following demonstration is based on X$ table x$kccle which is basically an acronym for "[K]ernel Layer, [C]ache Layer, [C]ontrol File Management, [L]og Files, Log File [E]ntries" (MOS ID #22241.1). [1]

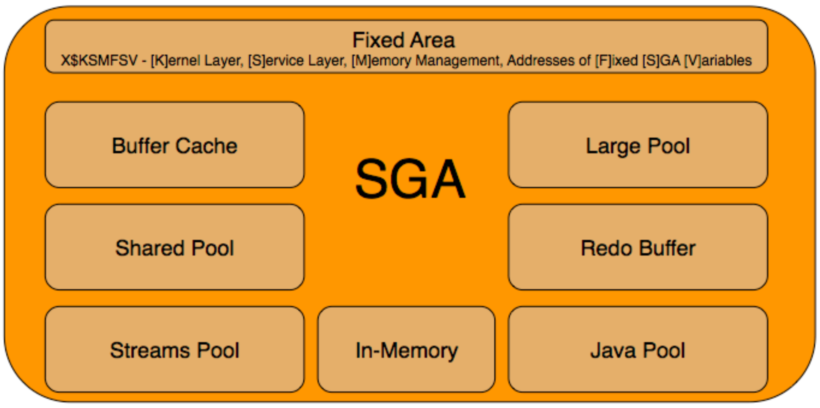
```
SYS@S12DB:8> select * from x$kccle;
ADDR                  INDX    INST_ID     CON_ID      LENUM     LESIZ       LESEQ     LEHWS
---------------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
FFFF80FFBDB25390          0          1          0          1     307200         55         40
FFFF80FFBDB25390          1          1          0          2     307200         53         43
FFFF80FFBDB25390          2          1          0          3     307200         54         67

-----------------------------------------------------------------
| Id  | Operation       | Name       | E-Rows |E-Bytes| Cost (%CPU)|
-----------------------------------------------------------------
|   0 | SELECT STATEMENT |           |        |       |     1 (100)|
|   1 |  FIXED TABLE FULL| X$KCCLE |    3     |   366 |     0   (0)|
-----------------------------------------------------------------

SYS@S12DB:8> @/Users/iStefan/SQLs/10_talks/DOAG2016_161115/tanel_find_heap_mem.sql
FFFF80FFBDB25390
LOC KSMCHPTR         KSMCHIDX   KSMCHDUR   KSMCHCOM          KSMCHSIZ   KSMCHCLS KSMCHTYP  KSM
--- ---------------- ---------- ---------- ---------------- ---------- -------- --------- ---
UGA FFFF80FFBDB24F90                       kxsFrame4kPage    4152       freeabl  0         00

SYS@S12DB:8> select * from x$kccle;
ADDR                  INDX    INST_ID     CON_ID      LENUM     LESIZ       LESEQ     LEHWS
---------------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
FFFF80FFBDD032B0          0          1          0          1     307200         55         40
FFFF80FFBDD032B0          1          1          0          2     307200         53         43
FFFF80FFBDD032B0          2          1          0          3     307200         54         67

SYS@S12DB:8> @/Users/iStefan/SQLs/10_talks/DOAG2016_161115/tanel_find_heap_mem.sql
FFFF80FFBDD032B0
LOC KSMCHPTR         KSMCHIDX   KSMCHDUR   KSMCHCOM          KSMCHSIZ   KSMCHCLS KSMCHTYP  KSM
--- ---------------- ---------- ---------- ---------------- ---------- -------- --------- ---
UGA FFFF80FFBDD02EB0                      kxsFrame4kPage    4152       freeabl  0         00
```

The execution plan looks the same as for the query on a plain representation of a C memory structure, like shown above with the X$ table x$ksuse, but the memory address of the X$ table x$kccle entries resides in the UGA now. In this case the additional (helper) function reads the control file, allocates memory in the UGA (kxsFrame4kPage chunk) and copies the requested data from the control file into this piece of memory. After the helper function has been executed the usual fixed table row source function kicks in and presents the requested output. The memory address is not fixed and changes each time a query is run on the X$ table x$kccle as the memory in UGA (kxsFrame4kPage chunk) is allocated from various memory locations for each execution.

The following content is an extended SQL trace of a query on the X$ table x$kccle.

```
…
PARSING IN CURSOR #18446604434621016928 len=21 dep=0 uid=0 oct=3 lid=0 tim=4847314211
hv=26881378 ad='19b52f070' sqlid='4nkm8x40tnbb2'
select * from x$kccle
END OF STMT
…
WAIT #18446604434621016928: nam='control file sequential read' ela= 63 file#=0 block#=1
blocks=1 obj#=662 tim=4847316598
WAIT #18446604434621016928: nam='control file sequential read' ela= 59 file#=0 block#=16
blocks=1 obj#=662 tim=4847316805
WAIT #18446604434621016928: nam='control file sequential read' ela= 70 file#=0 block#=18
blocks=1 obj#=662 tim=4847317206
WAIT #18446604434621016928: nam='control file sequential read' ela= 50 file#=0 block#=21
blocks=1 obj#=662 tim=4847317407
…
```

SGA memory structure overview


Fixed Area
X$KSMFSV - [K]ernel Layer, [S]ervice Layer, [M]emory Management, Addresses of [F]ixed [S]GA [V]ariables
Buffer Cache
SGA
Large Pool
Shared Pool
Redo Buffer
Streams Pool
In-Memory
Java Pool

The SGA consists of various components like the “fixed size area”, the “variable size area” (java pool, large pool, shared pool, streams pool), the buffer cache (which is also part of the “variable size area” from the OS perspective), the redo buffer and the in-memory area.

Fixed size area
The fixed size area is a component of the SGA that varies in size based on the platform and release. It is computed when Oracle is compiled and linked and it is located at the start of the first granule (more about granules later on). Basically it contains all fixed variables (e.g. like SCN), parent latches and pointers to all other structures in SGA.

The X$ table x$ksmfsv which is basically an acronym for “[K]ernel Layer, [S]ervice Layer, [M]emory Management, Addresses of [F]ixed [S]GA [V]ariables” (MOS ID #22241.1) [1] can be used to list all fixed variables. The following demo is run on Oracle 12.1.0.2 and SunOS SOL 5.11 11.2 (x86).

```
SYS@S12DB:327> select addr, ksmfsnam, ksmfsadr, ksmfssiz, ksmfstyp from x$ksmfsv where
ksmfsnam = 'kcsgscn_';
ADDR             KSMFSNAM         KSMFSADR          KSMFSSIZ   KSMFSTYP
---------------- ---------------- ---------------- ---------- ----------
0000000011152080 kcsgscn_         0000000060023F20 48         kcslf
```

The variable “kcsgscn_” (the SCN) is located at memory address 0x11152080 and holds the value 0x60023F20 (pointer), which is an item of data that is 48 bytes long and of type “kcslf”.
The content of memory address 0x60023F20 can be examined via the X$ table x$ksmmem which is basically an acronym for “[K]ernel Layer, [S]ervice Layer, [M]emory Management, SGA [MEM]ory” (MOS ID #22241.1) [1] or with help of oradebug.

```
SYS@S12DB:327> oradebug setmypid
SYS@S12DB:327> oradebug dumpvar sga kcsgscn_
kcslf kcsgscn_ [060023F20, 060023F50) = 002ADDDE 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 60023A20 00000000

SYS@S12DB:327> select KSMMMVAL from X$KSMMEM where ADDR = hextoraw('0000000060023F20');
KSMMMVAL
----------------
00000000002ADDDE
```

The SCN can also be verified with help of the fixed view v$database.

```
SYS@S12DB:339> select (select KSMMMVAL from X$KSMMEM where ADDR =
hextoraw('0000000060023F20')) as SCN_HEX_KSMMEM, to_char(CURRENT_SCN,'xxxxxxxxxxxxxxxx') as
SCN_HEX_DATABASE, CURRENT_SCN as SCN_DECIMAL from V$DATABASE;

SCN_HEX_KSMMEM   SCN_HEX_DATABASE  SCN_DECIMAL
---------------- ----------------- -----------
00000000002ADE52           2ade52     2809426
```

<u>SGA from OS perspective (Shared memory segments)</u>
Most Unix systems support three types of inter-process communication mechanisms, which first appeared in Unix System V. These are message queues, semaphores and shared memory. These System V IPC mechanisms all share common authentication methods. Processes may access these resources only by passing a unique reference identifier to the kernel via system calls. Access to these System V IPC objects is checked using access permissions, much like accesses to files are checked. The access rights to the System V IPC object is set by the creator of the object via system calls. The object's reference identifier is used by each mechanism as an index into a table of resources.

Oracle's SGA is implemented by several System V shared memory segments on most Unix systems (disregarding the combination of AMM - Automatic Memory Management and Linux).

The following demo is run on Oracle 12.1.0.2 and SunOS SOL 5.11 11.2 (x86).

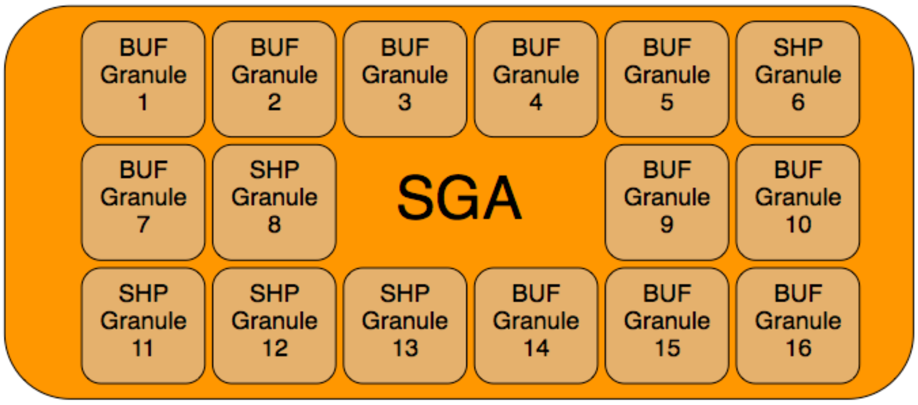
```
SYS@S12DB:336> show parameter
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_cache_size                        big integer 2000M
inmemory_size                        big integer 128M
memory_target                        big integer 0
sga_target                           big integer 4000M
shared_pool_size                     big integer 1200M

SYS@S12DB:336> select * from v$sga;
NAME                      VALUE CON_ID
-------------------- ---------- ----------
Fixed Size              3011592      0
Variable Size        1560284152      0
Database Buffers     2483027968      0
Redo Buffers           13762560      0
In-Memory Area        134217728      0

shell> ipcs -ma
IPC status from <running system> as of Sunday, September 18, 2016 10:10:31 AM CEST
T         ID      KEY        MODE        OWNER    GROUP  CREATOR    CGROUP NATTCH      SEGSZ
Shared Memory:
m    67108909   0xa73fde28 --rw-r-----   oracle      dba   oracle      dba     49      20480
m    67108908   0x0        --rw-r-----   oracle      dba   oracle      dba     49   14680064
m    67108907   0x0        --rw-r-----   oracle      dba   oracle      dba     49 4076863488
m    67108906   0x0        --rw-r-----   oracle      dba   oracle      dba     49  100663296
m    67108905   0x0        --rw-r-----   oracle      dba   oracle      dba     49    4194304
```

```
SYS@S12DB:336> oradebug setmypid
SYS@S12DB:336> oradebug ipc

Area #0 `Fixed Size' containing Subareas 4-4
  Total size 00000000002df408 Minimum Subarea size 00000000
   Area  Subarea    Shmid     Segment Addr     Stable Addr     Actual Addr
      0        4 67108905 0000000060000000 0000000060000000 0000000060000000
               Subarea size     Segment size    Req_Protect  Cur_protect
                          00000000002e0000 0000000000400000 default       readwrite
 Area #1 `Variable Size' containing Subareas 1-1
  Total size 00000000f1000000 Minimum Subarea size 01000000
   Area  Subarea    Shmid     Segment Addr     Stable Addr     Actual Addr
      1        1 67108907 00000000c0000000 00000000c0000000 00000000c0000000
               Subarea size     Segment size    Req_Protect  Cur_protect
                          00000000f1000000 00000000f3000000 default       readwrite
 Area #2 `Redo Buffers' containing Subareas 3-3
  Total size 0000000000d20000 Minimum Subarea size 00001000
   Area  Subarea    Shmid     Segment Addr     Stable Addr     Actual Addr
      2        3 67108908 00000001c0000000 00000001c0000000 00000001c0000000
               Subarea size     Segment size    Req_Protect  Cur_protect
                          0000000000d20000 0000000000e00000 default       readwrite
 Area #3 `imc area default 0' containing Subareas 2-2
  Total size 0000000002000000 Minimum Subarea size 01000000
   Area  Subarea    Shmid     Segment Addr     Stable Addr     Actual Addr
      3        2 67108907 00000000c0000000 00000001b1000000 00000001b1000000
               Subarea size     Segment size    Req_Protect  Cur_protect
                          0000000002000000 00000000f3000000 default       readwrite
 Area #4 `imc area rdonly 0' containing Subareas 0-0
  Total size 0000000006000000 Minimum Subarea size 01000000
   Area  Subarea    Shmid     Segment Addr     Stable Addr     Actual Addr
      4        0 67108906 0000000080000000 0000000080000000 0000000080000000
               Subarea size     Segment size    Req_Protect  Cur_protect
                          0000000006000000 0000000006000000 default       read
 Area #5 `skgm overhead' containing Subareas 5-5
  Total size 0000000000005000 Minimum Subarea size 00000000
   Area  Subarea    Shmid     Segment Addr     Stable Addr     Actual Addr
      5        5 67108909 0000000200000000 0000000200000000 0000000200000000
               Subarea size     Segment size    Req_Protect  Cur_protect
                          0000000000005000 0000000000005000 default       readwrite
```

At startup Oracle has created 5 shared memory segments on OS level for different purposes:

- Shared memory segment ID 67108905 for the fixed size area (fixed variables and pointers to all other structures in SGA)
- Shared memory segment ID 67108907 for the variable size area which contains the java pool, large pool, shared pool, streams pool, buffer cache and the in-memory component IMCA_RW (imc area default 0) which is the in-memory journal
- Shared memory segment ID 67108908 for the redo buffer
- Shared memory segment ID 67108906 for the in-memory component IMCA_RO (imc area rdonly 0) which is the in-memory column data
- Shared memory segment ID 67108909 for the skgm overhead which is OS specific and contains the index for/into the shared memory structures

SGA granules



Oracle has re-engineered the SGA in Oracle 9i to move memory between various memory areas like the shared pool, buffer cache, java pool, streams pool and large pool in an easy and "standardized" way. The memory could be moved manually between the key areas in Oracle 9i and starting with Oracle 10g it could be done automatically by the instance itself.

Oracle uses fixed-size memory chunks called "granules" for that purpose. The granule size varies based on the operating system, Oracle version and SGA size. MOS ID #947152.1 describes the different algorithms for determining the corresponding granule size. Each configured component consists of at least one granule and the minimum size of some components can be greater than one granule (and rounded up to the nearest granule boundary). The minimum size of one granule per component becomes very important in case of multiple shared pool sub-pools (and durations) – more information about the shared pool sub-pools and durations is in the "SGA - Shared pool implementation" chapter.

In reality the memory relocation between the key areas (e.g. from shared pool to buffer cache) may be a little bit more complex ("KGH: NO ACCESS") as granules can also be partly relocated but these cases are not considered in the following section.

The following demo is run on Oracle 12.1.0.2 and SunOS SOL 5.11 11.2 (x86).

```
SYS@S12DB:336> select * from v$sgainfo order by name;
NAME                                  BYTES RES CON_ID
-------------------------------- ---------- --- ----------
Buffer Cache Size                2483027968 Yes          0
Data Transfer Cache Size                  0 Yes          0
Fixed SGA Size                      3011592 No           0
Free SGA Memory Available                 0              0
Granule Size                       16777216 No           0
In-Memory Area Size               134217728 No           0
Java Pool Size                    134217728 Yes          0
Large Pool Size                    33554432 Yes          0
Maximum SGA Size                 4194304000 No           0
Redo Buffers                       13762560 No           0
Shared IO Pool Size               150994944 Yes          0
Shared Pool Size                 1258291200 Yes          0
Startup overhead in Shared Pool   178760832 No           0
Streams Pool Size                 134217728 Yes          0
```

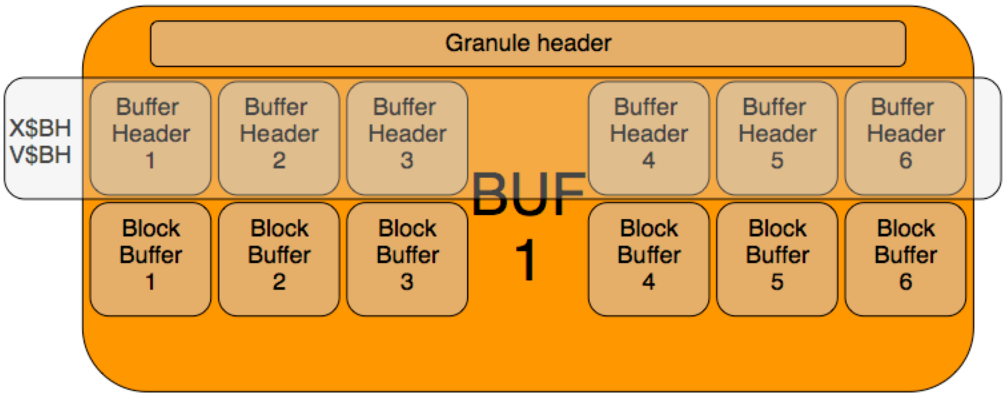
The granule size is 16 MB in this system and various components are resizable.

The X$ table x$ksmge externalizes the list of all the granules and the X$ table x$kmgsct externalizes the description to the corresponding granule type.

```
SYS@S12DB:336> select ge.grantype, ct.component, ge.granprev, ge.grannum, ge.grannext from
x$ksmge ge, x$kmgsct ct where ct.grantype = ge.grantype order by ge.grantype;

  GRANTYPE COMPONENT                        GRANPREV   GRANNUM    GRANNEXT
---------- -------------------------------- ---------- ---------- ----------
         1 shared pool                      232        231        230
         1 shared pool                      233        232        231
…
         2 large pool                       166        158        0
         2 large pool                       0          166        158
…
         3 java pool                        151        150        0
         3 java pool                        152        151        150
…
         4 streams pool                     143        142        0
         4 streams pool                     144        143        142
…
         7 DEFAULT buffer cache             2          1          141
         7 DEFAULT buffer cache             3          2          1
…
        15 Shared IO Pool                   133        132        0
        15 Shared IO Pool                   136        133        132
…
```

Every granule of every resizable SGA component is listed in the X$ table x$ksmge. This table represents the granule array as a linked list that points forward and backward to other elements (columns GRANPREV and GRANNEXT) in the array. It links the granules that make up each of the different components in the SGA.
The forward and backward pointers can also have gaps (e.g. large pool in the query output above) in the linked list - it indicates that (dynamic) resize operations have occurred.

SGA - Buffer pool / DB_CACHE



Each buffer cache granule consists mainly of three parts – the granule header, an array of buffer headers and an array of buffers that are used for holding copies of the data blocks. The buffer headers are related very closely to the block buffers – it is a permanent one to one link. Buffer headers do not contain any block data, only information about the block, the state of the corresponding buffer and a lot of pointers to other buffer headers (e.g. LRU list, etc.).

The following demo is run on Oracle 12.1.0.2 and SunOS SOL 5.11 11.2 (x86).

```
SYS@S12DB:177> select current_size, granule_size from v$sga_dynamic_components where component
= 'DEFAULT buffer cache';

CURRENT_SIZE GRANULE_SIZE
------------ ------------
  2332033024     16777216

SYS@S12DB:177> select name, block_size, buffers, block_size*buffers as buffer_size from
v$buffer_pool;

NAME                 BLOCK_SIZE    BUFFERS BUFFER_SIZE
-------------------- ---------- ---------- -----------
DEFAULT                    8192     273413  2239799296
```

There is a discrepancy between the current total size (2332033024) of the buffer cache and how many blocks are stored in this buffer cache (2239799296). This gap is caused by the overhead for the granule header and the buffer headers. Some simple arithmetic can be used to cross-check this:

```
SYS@S12DB:177> ACCEPT nbuff NUMBER PROMPT  'Enter number of buffers: '
Enter number of buffers: 273413
SYS@S12DB:177> ACCEPT ngran NUMBER PROMPT  'Enter number of buffer cache granules: '
Enter number of buffer cache granules: 139
SYS@S12DB:177> ACCEPT ngsiz NUMBER PROMPT  'Enter granule size: '
Enter granule size: 16777216
SYS@S12DB:177> select &nbuff/&ngran as buff_per_granule, &ngsiz-(&nbuff/&ngran)*8192 as
overhead_per_granule from dual;
old   1: select &nbuff/&ngran as buff_per_granule, &ngsiz-(&nbuff/&ngran)*8192 as
overhead_per_granule from dual
new   1: select     273413/    139 as buff_per_granule,   16777216-(    273413/   139)*8192 as
overhead_per_granule from dual

BUFF_PER_GRANULE OVERHEAD_PER_GRANULE
---------------- --------------------
            1967               663552
```

1967 buffers can be stored in one 16 MB buffer cache granule and there is an overhead of 648 kB per 16 MB buffer cache granule. A dump of the granules (with help of oradebug) confirms the calculation.

```
SYS@S12DB:177> oradebug setmypid
SYS@S12DB:177> oradebug dump dump_all_comp_granules 2

shell> grep -i ghdr_bufs
/oracle/S12DB/oratrace/diag/rdbms/s12db/S12DB/trace/S12DB_ora_1288.trc
129: gptr=0x140000000, ghdr_numbh=1978, ghdr_bufs=1967, ghdr=0x140ffe000,
spc=16777216numbh=1978
128: gptr=0x13f000000, ghdr_numbh=1978, ghdr_bufs=1967, ghdr=0x13fffe000,
spc=16777216numbh=1978
…

shell> grep -i ghdr_bufs
/oracle/S12DB/oratrace/diag/rdbms/s12db/S12DB/trace/S12DB_ora_1288.trc | wc -l
139
```

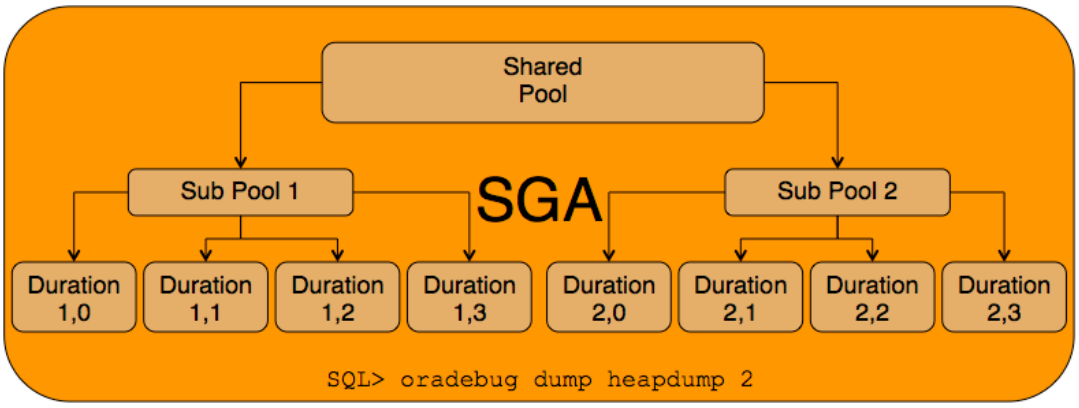
```
SYS@S12DB:177> select HLADDR,
decode(STATE,0,'free',1,'xcur',2,'scur',3,'cr',4,'read',5,'mrec',6,'irec',7,'write',8,'pi',9,'
memory',10,'mwrite',11,'donated',12,'protected',13,'securefile',14,'siop',15,'recckpt',16,'fla
shfree',17,'flashcur',18,'flashna') as STATE, PRV_HASH, NXT_HASH, BA, DBARFIL, DBABLK from
X$BH where rownum < 10;

HLADDR           STATE   PRV_HASH         NXT_HASH         BA               DBARFIL    DBABLK
---------------- ------- ---------------- ---------------- ---------------- ---------- -------
00000001AE6C0C88 xcur    00000001AE6C7598 00000001AE6C7598 000000014C412000 1          95651
00000001AE6C0D48 xcur    00000001AE6C7748 00000001AE6C7748 000000015E2EA000 1          77940
00000001AE6C11C8 xcur    00000001AE6C8338 00000001AE6C8338 000000014CA7A000 1          28988
00000001AE6C14C8 xcur    00000001AE6C8BE8 00000001AE6C8BE8 000000014C98C000 1          15458
00000001AE6C1648 xcur    00000001AE6C8EF8 00000001AE6C8EF8 000000015E20C000 1          101429
00000001AE6C1708 xcur    00000001AE6C9098 00000001AE6C9098 000000014C6BE000 1          83718
00000001AE6C1888 xcur    00000001AE6C9598 00000001AE6C9598 000000014B2D0000 1          30585
00000001AE6C1AC8 xcur    00000001AE6C9A48 00000001AE6C9A48 000000014A44C000 1          98845
```

The X$ table x$bh which is basically an acronym for "[B]uffer Management, Buffer [H]ash" (MOS ID #22241.1) [1] externalizes the buffer headers. The previous query output contains information about the latch address that protects the corresponding hash bucket (column HLADDR), the state of the data block/buffer (column STATE), the address of the previously and following attached buffer header in one double linked list (column PRV_HASH and NXT_HASH) and the address of the corresponding data block buffer (column BA).

SGA - Shared pool implementation
The shared pool is also made up of a number of granules or better said that each duration is made up of at least one granule. Basically there were two reasons why Oracle split-up the shared pool in this way. The first reason is scalability - Oracle has one shared pool latch per sub-pool which makes various operations like shared pool memory allocation or linked list modifications much more scalable. The second reason is to avoid ORA-04031 errors as a consequence of memory fragmentation. Oracle requires a lot of different memory (chunk) sizes in the shared pool, e.g. between 512 and 8432 bytes for the dictionary cache (with Oracle 12.1.0.2). Oracle uses the durations to "group" these different kinds of memory requests, e.g. the dictionary cache is usually stored in duration 0, the SQL area allocations (usually 4 KB) are stored in duration 3 and KGLH0 allocations (usually 4 KB or more) are stored in duration 0 (all sizes specific to Oracle 12.1.0.2).

Starting with Oracle 9.2 the shared pool can be divided into sub-pools and starting with Oracle 10g R2 each sub-pool may also be split-up into four durations (which has been reduced to two durations with Oracle 12c).
Oracle has an internal algorithm (which, as far as I know, is based on cpu count and shared pool size) to determine how many sub-pools should be created. The hidden parameter "_kghdsidx_count" controls how much sub-pools should be created. Durations are created automatically if ASMM or AMM is used, which sets the hidden parameter "_enable_shared_pool_durations" to TRUE.

Shared pool implementation with Oracle version >= 10.2 and < 12.1



The following demo is run on Oracle 11.2.0.3.6 and Linux OEL 3.8.13-118.3.2.el6uek.x86_64.

The X$ table x$kghlu which is basically an acronym for "[K]ernel Layer, [G]eneric Layer, [H]eap Manager, State (summary) of [L]R[U] heap - defined in ksmh.h" (MOS ID #22241.1) [1] lists the number of sub-pools.

```
SYS@T11DB:94> select decode(kghlushrpool,0,'Java Pool',1,'Shared Pool') as pool, kghluidx
num_subpool from x$kghlu;
POOL         NUM_SUBPOOL
------------ -----------
Shared Pool            1

SYS@T11DB:94> select x.ksppinm NAME, y.ksppstvl VALUE, ksppdesc DESCRIPTION from x$ksppi x,
x$ksppcv y where x.inst_id = userenv('Instance') AND y.inst_id = userenv('Instance') AND
x.indx = y.indx AND x.ksppinm in ('_kghdsidx_count','_enable_shared_pool_durations');
NAME                              VALUE                  DESCRIPTION
--------------------------------- ---------------------- -----------------------------------------------
_enable_shared_pool_durations     TRUE                   temporary to disable/enable kgh policy
_kghdsidx_count                   1                      max kghdsidx count
```

The shared pool is not divided into multiple sub-pools (due to the internal algorithm based on cpu_count and shared pool size) but shared pool durations are enabled (because the instance uses ASMM).

```
SYS@T11DB:94> select child#, gets from v$latch_children where name = 'shared pool' order by
child#;
    CHILD#       GETS
---------- ----------
         1     112121
         2         19
         3         19
         4         19
         5         19
         6         19
         7         19
```

The number of shared pool latches is hardcoded into Oracle but the ones that are actually used depends on the number of "real" sub-pools. In this case only one latch is active as Oracle created only one sub-pool.

```
SYS@T11DB:94> SELECT 'shared pool ('||NVL(DECODE(TO_CHAR(ksmdsidx),'0','0 - Unused',ksmdsidx),
'Total')||'):' subpool , SUM(ksmsslen) bytes FROM x$ksmss WHERE ksmsslen > 0 GROUP BY ROLLUP (
ksmdsidx ) ORDER BY subpool ASC;
SUBPOOL                                                         BYTES
--------------------------------------------------------------- ----------
shared pool (0 - Unused):                                       75497472
shared pool (1):                                                163577856
shared pool (Total):                                            239075328

SYS@T11DB:94> select current_size from v$sga_dynamic_components where component = 'shared
pool';
CURRENT_SIZE
------------
   239075328
```

The X$ table x$ksmss which is basically an acronym for “[K]ernel Layer, [S]ervice Layer, [M]emory Management, [S]GA Objects, Statistics (lengths) of SGA objects” (MOS ID #22241.1) [1] provides statistics about the shared pool. As expected shared pool sub-pool 1 is listed but there is also a shared pool sub-pool 0 which is a feature that was introduced with Oracle 10g. Not all the memory is immediately allocated by the various shared pool sub-pools - some memory is reserved for individual sub-pool growth that allows some sub-pools to grab more memory than others.

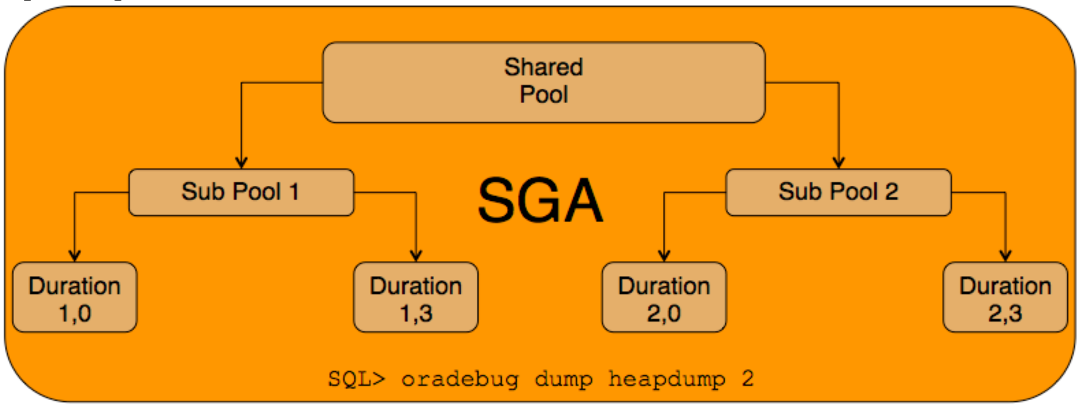
```
SYS@T11DB:94> oradebug setmypid
Statement processed.
SYS@T11DB:94> oradebug tracefile_name
/oracle/oratrace/T11DB/diag/rdbms/t11db/T11DB/trace/T11DB_ora_4213.trc
SYS@T11DB:94> oradebug dump heapdump 2

shell> grep -i "sga heap"
/oracle/oratrace/T11DB/diag/rdbms/t11db/T11DB/trace/T11DB_ora_4213.trc
HEAP DUMP heap name="sga heap"  desc=0x60001190
HEAP DUMP heap name="sga heap(1,0)"  desc=0x60054720
HEAP DUMP heap name="sga heap(1,1)"  desc=0x60055f78
HEAP DUMP heap name="sga heap(1,2)"  desc=0x600577d0
HEAP DUMP heap name="sga heap(1,3)"  desc=0x60059028
```

A SGA heap dump provides more detailed information about all the shared sub-pools and durations. In this case we have one shared sub-pool (1) with 4 durations (0-3). A detailed drill-down into each duration is provided in the “Shared pool implementation with Oracle version >= 12.1” section.

```
SYS@T11DB:94> oradebug setmypid
Statement processed.
SYS@T11DB:94> oradebug tracefile_name
/oracle/oratrace/T11DB/diag/rdbms/t11db/T11DB/trace/T11DB_ora_4213.trc
SYS@T11DB:94> oradebug dump dump_all_comp_granules 2

shell> grep -i "sga heap("
/oracle/oratrace/T11DB/diag/rdbms/t11db/T11DB/trace/T11DB_ora_4213.trc
Address 0x85400000 to 0x85800000 in sga heap(1,0) (idx=1, dur=1)
Address 0x85000000 to 0x85400000 in sga heap(1,1) (idx=1, dur=2)
Address 0x84c00000 to 0x85000000 in sga heap(1,0) (idx=1, dur=1)
Address 0x84800000 to 0x84c00000 in sga heap(1,0) (idx=1, dur=1)
…
Address 0x7fc00000 to 0x80000000 in sga heap(1,1) (idx=1, dur=2)
Address 0x7f800000 to 0x7fc00000 in sga heap(1,2) (idx=1, dur=3)
Address 0x7f400000 to 0x7f800000 in sga heap(1,3) (idx=1, dur=4)
Address 0x7f000000 to 0x7f400000 in sga heap(1,3) (idx=1, dur=4)
Address 0x7ec00000 to 0x7f000000 in sga heap(1,1) (idx=1, dur=2)
Address 0x7e800000 to 0x7ec00000 in sga heap(1,2) (idx=1, dur=3)
Address 0x7e400000 to 0x7e800000 in sga heap(1,3) (idx=1, dur=4)
Address 0x7e000000 to 0x7e400000 in sga heap(1,3) (idx=1, dur=4)
…
```

A dump of the granules shows that each duration consists at least of one non-overlapping granule.

Shared pool implementation with Oracle version >= 12.1



The shared pool structure has changed with Oracle 12.1 once again. Changes for unpublished bug #8857940 are included in 12.1.0.1 and permit to group the shared pool durations in 2 groups, which should allow better shareability of the memory and should avoid unnecessary ORA-4031 errors (MOS ID #1675470.1).

The following demo is run on Oracle 12.1.0.2 and SunOS SOL 5.11 11.2 (x86).

```
SYS@S12DB:9> select decode(kghlushrpool,0,'Java Pool',1,'Shared Pool') as pool, kghluidx
num_subpool from x$kghlu;
POOL         NUM_SUBPOOL
----------- -----------
Shared Pool          1

SYS@S12DB:9> select x.ksppinm NAME, y.ksppstvl VALUE, ksppdesc DESCRIPTION from x$ksppi x,
x$ksppcv y where x.inst_id = userenv('Instance') AND y.inst_id = userenv('Instance') AND
x.indx = y.indx AND x.ksppinm in ('_kghdsidx_count','_enable_shared_pool_durations');
NAME                           VALUE                DESCRIPTION
------------------------------ -------------------- ------------------------------------------
_enable_shared_pool_durations  TRUE                 temporary to disable/enable kgh policy
_kghdsidx_count                1                    max kghdsidx count
```

The shared pool is not divided into multiple sub-pools (due to the internal algorithm based on cpu_count and shared pool size) but shared pool durations are enabled (because of the instance uses ASMM).

```
SYS@S12DB:9> select child#, gets from v$latch_children where name = 'shared pool' order by
child#;
    CHILD#       GETS
---------- ----------
 1             121723
 2                 22
 3                 22
 4                 22
 5                 22
 6                 22
 7                 22
```

The number of shared pool latches is hardcoded into Oracle but the ones that are actually used depends on the number of “real” sub-pools. In this case only one latch is active as Oracle created only one sub-pool.

```
SYS@S12DB:9> SELECT 'shared pool ('||NVL(DECODE(TO_CHAR(ksmdsidx),'0','0 - Unused',ksmdsidx),
'Total')||'):' subpool , SUM(ksmsslen) bytes FROM x$ksmss WHERE ksmsslen > 0 GROUP BY ROLLUP (
ksmdsidx ) ORDER BY subpool ASC;
SUBPOOL                                                  BYTES
-------------------------------------------------------- ----------
shared pool (0 - Unused):                                 905969664
shared pool (1):                                          352321536
shared pool (Total):                                     1258291200

SYS@S12DB:9> select current_size from v$sga_dynamic_components where component = 'shared
pool';
CURRENT_SIZE
------------
  1258291200
```

The X$ table x$ksmss which is basically an acronym for "[K]ernel Layer, [S]ervice Layer, [M]emory Management, [S]GA Objects, Statistics (lengths) of SGA objects" (MOS ID #22241.1) [1] provides statistics about the shared pool. As expected shared pool sub-pool 1 is listed but there is also a shared pool sub-pool 0 which is a feature that was introduced with Oracle 10g. Not all the memory is immediately allocated by the various shared pool sub-pools - some memory is reserved for individual sub-pool growth that allows some sub-pools to grab more memory than others. The X$ table x$ksmss externalizes these "reserved" granules as sub-pool 0, but in reality the corresponding granules are just marked as "sga heap (reserved)" and not assigned to a sub-pool.

```
SYS@S12DB:9> oradebug setmypid
SYS@S12DB:9> oradebug tracefile_name
/oracle/S12DB/oratrace/diag/rdbms/s12db/S12DB/trace/S12DB_ora_1270.trc
SYS@S12DB:9> oradebug dump heapdump 2

shell> grep -i "sga heap("
/oracle/S12DB/oratrace/diag/rdbms/s12db/S12DB/trace/S12DB_ora_1270.trc
HEAP DUMP heap name="sga heap(1,0)"  desc=60076a38
HEAP DUMP heap name="sga heap(1,3)"  desc=6007b340
```

A SGA heap dump provides more detailed information about all the shared sub-pools and durations. In this case we have one shared sub-pool (1) with 2 durations (0 and 3). This is the previously mentioned Oracle 12c enhancement (MOS ID #1675470.1).

```
SYS@S12DB:9> oradebug setmypid
SYS@S12DB:9> oradebug tracefile_name
/oracle/S12DB/oratrace/diag/rdbms/s12db/S12DB/trace/S12DB_ora_1270.trc
SYS@S12DB:9> oradebug dump dump_all_comp_granules 2

shell> grep -i "sga heap("
/oracle/S12DB/oratrace/diag/rdbms/s12db/S12DB/trace/S12DB_ora_1270.trc
Address 1b0000000 to 1b1000000 in sga heap(1,0) (idx=1, dur=1)
Address 1af000000 to 1b0000000 in sga heap(1,0) (idx=1, dur=1)
…
Address 1a0000000 to 1a1000000 in sga heap(1,3) (idx=1, dur=4)
Address 19f000000 to 1a0000000 in sga heap(1,0) (idx=1, dur=1)
Address 19e000000 to 19f000000 in sga heap(1,3) (idx=1, dur=4)
Address 19d000000 to 19e000000 in sga heap(1,0) (idx=1, dur=1)
Address 19c000000 to 19d000000 in sga heap(1,3) (idx=1, dur=4)
```

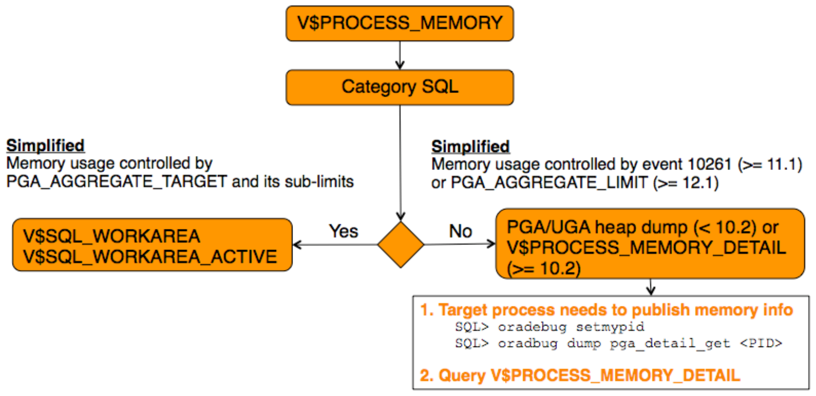
A dump of the granules shows that each duration consists at least of one non-overlapping granule.

A closer look in the heap dump trace file (or a query on X$ table x$ksmsp) also provides insights about why and where shared pool memory was allocated. Each of the shared pool durations consist of at least one extent and one extent is related to one granule. Since Oracle 11g every chunk for a library cache object is associated with a comment (hash value of the corresponding library cache object) when it is allocated.

```
Processing Oradebug command 'dump heapdump 2'
…
HEAP DUMP heap name="sga heap(1,0)"  desc=60076a38
…
EXTENT 0 addr=19d000000
…
  Chunk        19d0b8e18 sz=     4096    freeableU "KGLH0^8da600d0 "  ds=19f495938
  Chunk        19d0b9e18 sz=     4096    freeableU "KGLH0^8da600d0 "  ds=19f495938
  Chunk        19d0bae18 sz=     4096    freeableU "KGLH0^8da600d0 "  ds=19f495938
…
  Chunk        19d0cdef8 sz=     4096    recreatUC "KGLH0^8da600d0 "  latch=0
     ds        19f495938 sz=    16384 ct=        4
               19d0b8e18 sz=     4096
               19d0b9e18 sz=     4096
               19d0bae18 sz=     4096
…
  Chunk        19d410990 sz=     1072    recreatUR "KQR PO          "  latch=1a4919290
  Chunk        19d410dc0 sz=     1824    recreatPT "KQR SO          "  latch=0
…
HEAP DUMP heap name="sga heap(1,3)"  desc=6007b340
…
EXTENT 0 addr=19b000000
…
  Chunk        19bf1cf00 sz=     4096    freeableU "SQLA^8da600d0  "  ds=19d0cf910
  Chunk        19bf1cf00 sz=     4096    freeableU "SQLA^8da600d0  "  ds=19d0cf910
…
  Chunk        19bf2df00 sz=     4096    recreatUC "SQLA^8da600d0  "  latch=0
     ds        19d0cf910 sz=    12288 ct=        3
               19bf1bf00 sz=     4096
               19bf1cf00 sz=     4096
…
EXTENT 1 addr=19c000000
…

Processing Oradebug command 'dump dump_all_comp_granules 2'
GRANULE SIZE is 16777216
COMPONENT NAME : shared pool
…
Granule addr is 19d000000
Dumping layout
Address 19d000000 to 19e000000 in sga heap(1,0) (idx=1, dur=1)
…
Granule addr is 19b000000
Dumping layout
Address 19b000000 to 19c000000 in sga heap(1,3) (idx=1, dur=4)
…
Granule addr is 19c000000
Dumping layout
Address 19c000000 to 19d000000 in sga heap(1,3) (idx=1, dur=4)
…

SQL> select child_number,child_address,sql_text from v$sql where hash_value =
to_number('8da600d0', 'XXXXXXXXXXXXXXXX');
CHILD_NUMBER CHILD_ADDRESS    SQL_TEXT
------------ ---------------- ----------------------------------------------------------------
0            000000019F5712C8 select current_size from v$sga_dynamic_components where
                              component = 'shared pool'
```

The lines that start with "Chunk" identify chunks of memory in the extent (= granule) and each chunk shows its starting address and size in bytes. Most chunks have a comment in double quotes.
For example there are chunks (KQR - Kernel Query Rowcache) stored in shared sub-pool 1 / duration 0 which are related to rowcache objects (PO = parent object, SO = subordinate object) with a size of 1072 and 1824 bytes. There are also chunks (KGLH0 - Kernel General Library Heap 0 = environment, statistics and bind variables = SQL Heap 0) stored in shared sub-pool 1 / duration 0 with a size of 4 x 4096 bytes which are related to library cache objects and there are chunks (SQLA = SQL area – executable part of a cursor / execution plan = SQL Heap 6) in shared sub-pool 1 / duration 3 with a size of 3 x 4096 bytes which are related to library cache objects.

PGA - Analyze PGA memory usage on Oracle / SQL level


V$PROCESS_MEMORY
Category SQL
Simplified
Memory usage controlled by PGA_AGGREGATE_TARGET and its sub-limits
Simplified
Memory usage controlled by event 10261 (>= 11.1) or PGA_AGGREGATE_LIMIT (>= 12.1)
Yes
No
V$SQL_WORKAREA
V$SQL_WORKAREA_ACTIVE
PGA/UGA heap dump (< 10.2) or V$PROCESS_MEMORY_DETAIL (>= 10.2)
1. Target process needs to publish memory info
SQL> oradebug setmypid
SQL> oradbug dump pga_detail_get <PID>
2. Query V$PROCESS_MEMORY_DETAIL

Troubleshooting PGA memory usage can be pretty easy or a little bit more complicated depending on if the memory usage is caused by (SQL) work areas or by other allocations (e.g. PL/SQL code). The most common troubleshooting approach looks like this:

1. Query view V$PROCESS (PGA_USED_MEM / PGA_ALLOC_MEM) to identify the process(es) that causes high PGA memory usage
2. Query view V$PROCESS_MEMORY to identify category
3. Check views V$SQL_WORKAREA or V$SQL_WORKAREA_ACTIVE to identify work areas in case of SQL category
4. Check view V$PROCESS_MEMORY_DETAIL or create a PGA/UGA heap dump in case of of a non-SQL category

PGA memory can be controlled to a certain degree depending on the used Oracle version. PGA_AGGREGATE_TARGET can be considered as a soft limit as PGA can become larger than this init parameter value and it only controls the memory usage caused by SQL category. It has no impact on memory usage / allocations for PL/SQL collections, etc..
Since Oracle 11.1 the PGA memory limit for a non-SQL category can be controlled on process level by event 10261 [3] or since Oracle 12.1 via init parameter PGA_AGGREGATE_LIMIT on instance level. [5]

<u>SQL category</u>
The following demo is run on Oracle 12.1.0.2 and SunOS SOL 5.11 11.2 (x86).

```
TEST@S12DB:181> create table TEST as select * from DBA_SOURCE;
TEST@S12DB:181> select /*+ use_hash(t2) */ count(*) from TEST t1, TEST t2 where t1.owner =
t2.owner;

SYS@S12DB:339> select s.sid,p.spid,pm.* from v$session s, v$process p, v$process_memory pm
where s.paddr = p.addr and p.pid = pm.pid and s.sid = 181 order by category;
SID   SPID   PID   SERIAL#  CATEGORY         ALLOCATED  USED       MAX_ALLOCATED CON_ID
----- ------ ----- -------- ---------------- ---------- ---------- ------------- ----------
181   1228    55     2       Freeable        327680     0                             0
181   1228    55     2       Other           3697956               3697956            0
181   1228    55     2       PL/SQL          897472     883248     990512             0
181   1228    55     2       SQL             26248968   26100088   26248968           0

SYS@S12DB:339> select SID, SQL_ID, round(ACTUAL_MEM_USED/1024/1024,2) ACTUAL_MEM_USED_MB,
NUMBER_PASSES, OPERATION_TYPE, OPERATION_ID from V$SQL_WORKAREA_ACTIVE where SID = 181;
       SID SQL_ID          ACTUAL_MEM_USED_MB  NUMBER_PASSES      OPERATION_TYPE   OPERATION_ID
---------- ------------- ------------------ ------------------ ---------------- ---------------
       181 0w1v4xg8dwzf0              22.62                  0 HASH-JOIN        2
```

In this case most PGA memory is used for SQL category and the memory usage is caused by SQL ID "0w1v4xg8dwzf0" due to a hash join operation in execution plan step 2.

Non-SQL category

The following demo is run on Oracle 12.1.0.2 and SunOS SOL 5.11 11.2 (x86). The code is using UTL_HTTP to demonstrate a PGA memory leak caused by Oracle bug #22861143.

```
TEST@S12DB:180> begin
  dbms_network_acl_admin.append_host_ace (
    host => '*',
    ace => xs$ace_type(
    privilege_list => xs$name_list( 'http' ),
    principal_name => upper( 'TEST' ),
    principal_type => xs_acl.ptype_db
    )
  );
end;
/

TEST@S12DB:180> create or replace package test_pga_leak_http_only_s
is
procedure process_request;
end;
/

TEST@S12DB:180> create or replace package body test_pga_leak_http_only_s is
procedure process_request is
  loc_http_req utl_http.req;
  glo_http_url varchar2( 1000 ) := 'http://192.168.56.1:32400/web/index.html';
begin
  utl_http.set_transfer_timeout( 60 );
  loc_http_req := utl_http.begin_request( url => glo_http_url, method => 'POST' );
  utl_http.end_request( loc_http_req );
end;
end;
/

TEST@S12DB:180> begin
  for i in 1 .. 20000 loop
    test_pga_leak_http_only_s.process_request();
  end loop;
end;
/
```

The demo code calls the PL/SQL procedure 20000 times and the PL/SQL procedure code itself just calls a URL via UTL_HTTP.

```
SYS@S12DB:17> select s.sid,p.spid,pm.* from v$session s, v$process p, v$process_memory pm
where s.paddr = p.addr and p.pid = pm.pid and s.sid = 180 order by category;
SID   SPID   PID   SERIAL#  CATEGORY        ALLOCATED  USED       MAX_ALLOCATED CON_ID
----- ------ ----- -------- --------------- ---------- ---------- ------------- ----------
180  1261     64    3        Freeable        851968              0                       0
180  1261     64    3        Other           5957764                        5957764        0
180  1261     64    3        PL/SQL          147720           2256          198312        0
180  1261     64    3        SQL             162912              0         2191944        0
```

After the PL/SQL procedure is called 20000 times it still has allocated 5957764 bytes in category "Other". Each additional PL/SQL procedure call would increase the memory usage by 80 bytes in category "Other" until the server is out of memory or the memory limits (e.g. event 10261, PGA_AGGREGATE_LIMIT or operating system limits) are hit.

```
SYS@S12DB:17> select SID, SQL_ID, round(ACTUAL_MEM_USED/1024/1024,2) ACTUAL_MEM_USED_MB,
NUMBER_PASSES, OPERATION_TYPE, OPERATION_ID from V$SQL_WORKAREA_ACTIVE where SID = 180;
no rows selected
```

As expected view V$SQL_WORKAREA_ACTIVE does not list any non-SQL categories.

```
SYS@S12DB:17> select * from v$process_memory_detail where pid = 64 order by bytes desc;
no rows selected
```

By default view V$PROCESS_MEMORY_DETAIL does not list any memory details. The query process is not allowed to read some private memory content of other processes.
In consequence the query process needs to ask the target process to publish its memory details into view V$PROCESS_MEMORY_DETAIL.

```
SYS@S12DB:17> oradebug setmypid
SYS@S12DB:17> oradebug dump pga_detail_get 64

SYS@S12DB:17> select * from v$process_memory_detail where pid = 64 order by bytes desc;
no rows selected
```

The query process asked the other process to publish its information with help of oradebug, but unfortunately the information is still not available. PGA_DETAIL_GET will not immediately make the target process to report its usage – it will merely set a flag and the target process itself checks it when it is active. In my test case the target process (PID 64 / SID 180) was just in idle mode after it executed the PL/SQL procedure 20000 times.

```
TEST@S12DB:180> select 1 from dual;
1
----------
1

SYS@S12DB:17> select pid, serial#, category, name, heap_name, bytes, allocation_count from
v$process_memory_detail where pid = 64 order by bytes desc;
PID    SERIAL#    CATEGORY  NAME                       HEAP_NAME       BYTES      ALLOC_COUNT
----- ---------- --------- -------------------------- --------------- ---------- -------------
64     3          Other     kzxascPut: xs session cac  kzxauscini:hea  1600840         20001
64     3          Other     kxsFrame8kPage             session heap    791808             96
64     3          Other     qmushtCreate               qmtmInit        393504              6
64     3          Other     permanent memory           pga heap        305704             33
64     3          Other     qmtmltAlloc                qmtmInit        293272           1793
64     3          Other     permanent memory           session heap    218912             37
64     3          Other     free memory                pga heap        172576             31
64     3          Other     kxsFrame4kPage             session heap    153664             37
64     3          Other     kopo object                koh-kghu sessi  136816             10
…
```

Most of the PGA memory in category “Other” is used in heap “kzxauscini:hea” due to memory allocation reason “kzxascPut: xs session cac”. The names are cryptic, but some can be translated with help of oradebug.

```
SYS@S12DB:17> oradebug setmypid
SYS@S12DB:17> oradebug doc component
…
Components in library RDBMS:
--------------------------
…
  XS                                        XS Fusion Security (kzx)
…
    XSACL                                   XS ACL (kzxa)
…
```

Memory allocations due to “kzxa” are related to ACLs which also makes sense in this application scenario with UTL_HTTP requests.

The view V$PROCESS_MEMORY_DETAIL is available since Oracle version 10.2. A PGA/UGA heap dump and Tanel Poder’s heapdump_analyzer script [4] can be used in previous Oracle versions.

```
SYS@S12DB:17> oradebug setospid 1261
SYS@S12DB:17> oradebug tracefile_name
/oracle/S12DB/oratrace/diag/rdbms/s12db/S12DB/trace/S12DB_ora_1261.trc
SYS@S12DB:17> oradebug dump heapdump 1

shell> /oracle/talk/DOAG2016_161115/tanel_heapdump_analyzer.ksh
/oracle/S12DB/oratrace/diag/rdbms/s12db/S12DB/trace/S12DB_ora_1261.trc

  -- Heapdump Analyzer v1.00 by Tanel Poder ( http://www.tanelpoder.com )
  Total_size #Chunks  Chunk_size,        From_heap,       Chunk_type,  Alloc_reason
  ---------- ------- ------------ ----------------- ----------------- -----------------
     1768824   27      65512 ,     top uga heap,         freeable,    session heap
     1179072   18      65504 ,     top uga heap,         freeable,    session heap
      852072   13      65544 ,     top uga heap,         freeable,    session heap
      393984    6      65664 ,     top uga heap,         freeable,    session heap
      326920    5      65384 ,     top uga heap,         freeable,    session heap
      140880    3      46960 ,     top uga heap,         freeable,    session heap
…
```

A heap dump at level 1 dumps all top-level private heaps (PGA, UGA and call), but does not list the sub-heaps (e.g. like view V$PROCESS_MEMORY_DETAIL does). Adding power (2,29) (= 0x20000000 = 536870912) to the base heap number dumps the top 5 sub-heaps in any heap plus the top 5 sub-heaps inside any sub-heap (= 2 levels of sub-heap dump recursion) in detail as well.

```
SYS@S12DB:17> oradebug setospid 1261
SYS@S12DB:17> oradebug tracefile_name
/oracle/S12DB/oratrace/diag/rdbms/s12db/S12DB/trace/S12DB_ora_1261.trc
SYS@S12DB:17> oradebug dump heapdump 536870913

shell> /oracle/talk/DOAG2016_161115/tanel_heapdump_analyzer.ksh
/oracle/S12DB/oratrace/diag/rdbms/s12db/S12DB/trace/S12DB_ora_1261.trc
 -- Heapdump Analyzer v1.00 by Tanel Poder ( http://www.tanelpoder.com )
  Total_size #Chunks  Chunk_size,        From_heap,       Chunk_type,  Alloc_reason
  ---------- ------- ------------ ----------------- ----------------- -----------------
     1768824   27      65512 ,      top uga heap,         freeable,     session heap
     1597360 19967        80 ,    kzxauscini:hea,         freeable,     kzxascPut: xs s
     1401640  335       4184 ,      session heap,         freeable,     kzxauscini:hea
     1179072   18      65504 ,      top uga heap,         freeable,     session heap
      852072   13      65544 ,      top uga heap,         freeable,     session heap
…
        1248   12        104 ,    kzxauscini:hea,         freeable,     kzxascPut: xs s
…
        1008    9        112 ,    kzxauscini:hea,         freeable,     kzxascPut: xs s
…
         960   10         96 ,    kzxauscini:hea,         freeable,     kzxascPut: xs s
…
         264    3         88 ,    kzxauscini:hea,         freeable,     kzxascPut: xs s
…
```

For verification: The sum of chunks (20001) and the sum of bytes (1600840) that are allocated due to “kzxascPut: xs s” are identical with the content in view V$PROCESS_MEMORY_DETAIL.

A closer look into the raw heap dump (with level 536870913) reveals that heap "kzxauscini:hea" is allocated from "session heap" which is allocated from "top uga heap" in turn. This completes the picture of the heap dumps at level 1 and 536870913.

```
HEAP DUMP heap name="kzxauscini:hea"  desc=ffff80ffbdd8f9d8
 extent sz=0x1040 alt=32767 het=32767 rec=0 flg=2 opc=2
 parent=ffff80ffbdd6d728 owner=0 nex=0 xsz=0x1040 heap=0
…
EXTENT 0 addr=ffff80ffbcd21d58
…
  Chunk ffff80ffbcd22618 sz=       80    freeable  "kzxascPut: xs s"
  Chunk ffff80ffbcd22668 sz=       80    freeable  "kzxascPut: xs s"
  Chunk ffff80ffbcd226b8 sz=       80    freeable  "kzxascPut: xs s"
…

HEAP DUMP heap name="session heap"  desc=ffff80ffbdd6d728
 extent sz=0xffc8 alt=32767 het=32767 rec=0 flg=2 opc=2
 parent=ffff80ffbf068390 owner=1b0d9c9d0 nex=0 xsz=0x4880 heap=0
…
EXTENT 0 addr=ffff80ffbcd1a038
…
  Chunk ffff80ffbcd21d40 sz=     4184    freeable  "kzxauscini:hea "  ds=ffff80ffbdd8f9d8
  Chunk ffff80ffbcd22d98 sz=     4184    freeable  "kzxauscini:hea "  ds=ffff80ffbdd8f9d8
…
 Chunk ffff80ffbdcb4f68 sz=     2096    recreate  "kzxauscini:hea "  latch=0
     ds ffff80ffbdd8f9d8 sz=  1650144 ct=      452
…

HEAP DUMP heap name="top uga heap"  desc=ffff80ffbf068390
 extent sz=0xffc0 alt=256 het=32767 rec=0 flg=2 opc=3
 parent=0 owner=0 nex=0 xsz=0x1fff8 heap=0
…
```

PGA - Capture and source PGA memory allocations
A root cause analysis by capturing and sourcing PGA memory allocations is described in one of my blog posts here: http://scn.sap.com/community/oracle/blog/2016/01/30/oracle-trace-back-to-responsible-sql-or-plsql-code-for-a-particular-pga-memory-request-by-intercepting-process-with-dtrace

References
[1] http://www.eygle.com/refer/Oracle_x$table_list.htm
[2] http://blog.tanelpoder.com/files/scripts/fcha.sql
[3] http://www.oracle.com/technetwork/database/features/availability/exadata-consolidation-522500.pdf
[4] http://blog.tanelpoder.com/files/scripts/tools/unix/heapdump_analyzer
[5] https://fritshoogland.files.wordpress.com/2015/12/oracle-exadata-and-database-memory.pdf

Contact address
Stefan Koehler
Freelance Consultant (Soocs)
Gustav-Freytag-Weg 29
D-96450 Coburg, Germany

Phone:	+49 (0) 172 / 1796830
E-Mail	contact@soocs.de
Internet:	http://www.soocs.de
Twitter:	@OracleSK

Acknowledgment
I want to thank Frits Hoogland (https://fritshoogland.wordpress.com) for the technical review and content validation in general.