

# Chapter 3

# Large Page

FUJITSU LIMITED

April 2016

- What Are Large Pages?
  - Functions of Large Pages
  - Purpose of Large Pages
  - Memory Address Conversion and TLB
  - What Is the TLB? (Details)
  - TLB Configuration
  - Default Large Page Size
  - Aspects of Each Page Size
  - Page Size Selection Criteria

## ■ Functions of Large Pages

- Ipgparm Command
- Demand Paging
- Different Areas Allocated for Different Variable Types
- Arena Process
- Thread Heap
- Environment Variables Used for Tuning

## ■ Evaluation Functions for Large Pages

- List of Evaluation Functions for Large Pages
- Large Page Memory Usage Information
- Large Page Statistics

# What Are Large Pages?

- Functions of Large Pages
- Purpose of Large Pages
- Memory Address Conversion and TLB
- What Is the TLB? (Details)
- TLB Configuration
- Default Large Page Size
- Aspects of Each Page Size
- Page Size Selection Criteria

## ■ What are the functions of large pages?

- For applications that handle large-scale data, **memory is assigned with a larger page size** than the normal page size to...
  - reduce the overhead of OS address conversion, and
  - improve memory access performance.
  
- The size of a normal page is 8 KB, and the size of a large page may be 512 KB, 4 MB, or 32 MB.

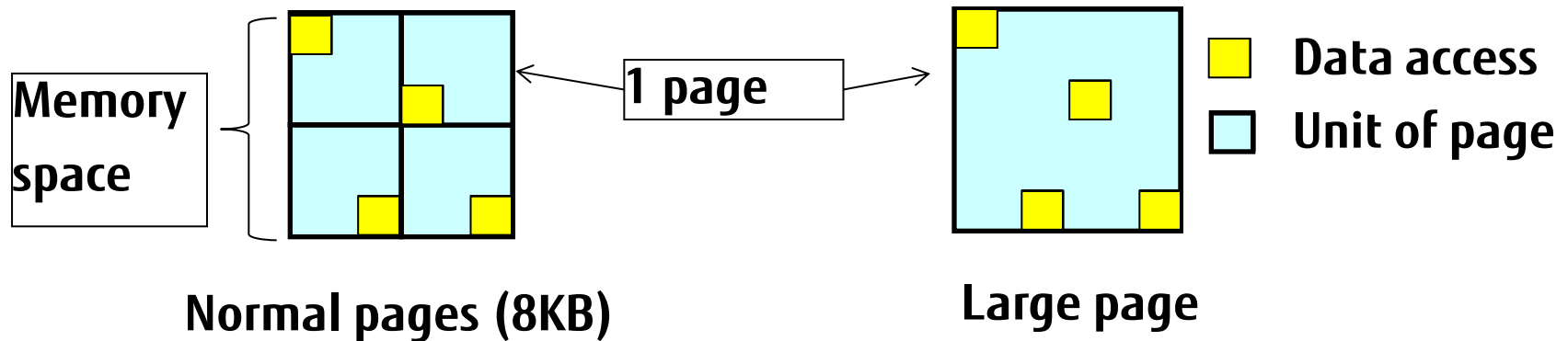
# Purpose of Large Pages

- Large pages expand the size of a page as the unit of memory management from 8 KB to a larger size (512 KB, 4 MB, 32 MB, etc.)

The larger size pages provide **wider coverage** of the CPU address conversion buffer (TLB), thereby **reducing TLB misses**.

→ **Improved execution performance**

## Conceptual diagram of normal and large pages



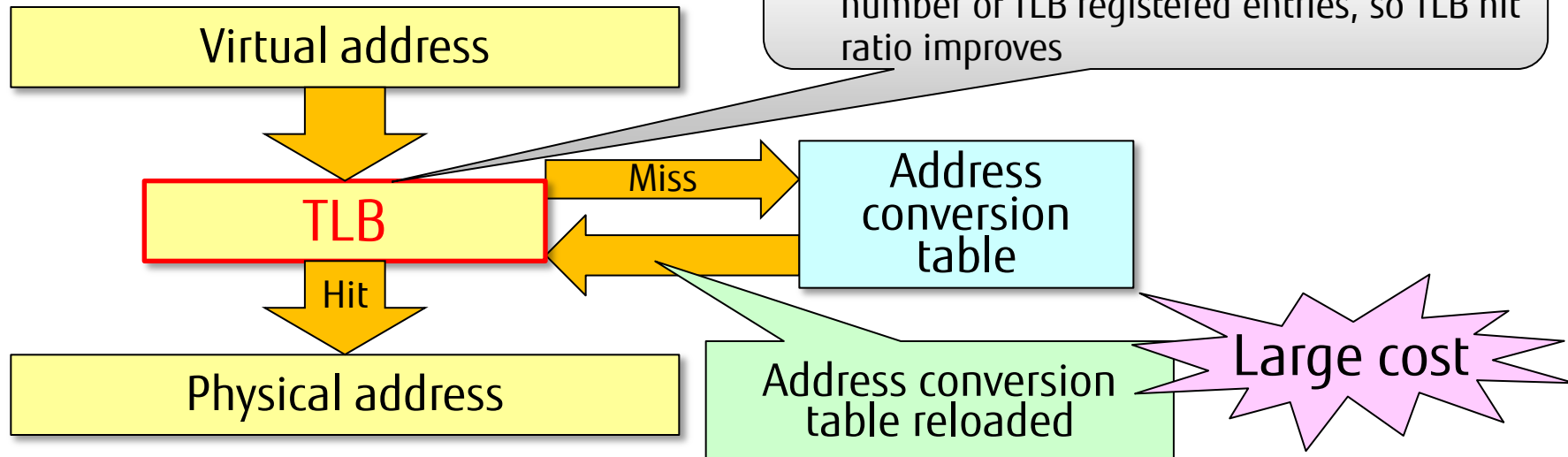
The normal pages have a TLB miss at each access of  (4 times).  
The large page has a TLB miss only at the initial access of  (1 time).

→ **Use of large pages reduces TLB misses**

# Memory Address Conversion and TLB

- For memory access by a program, **a virtual memory address must be converted to a physical memory address.**
- The address conversion **uses an address conversion table** residing on the main memory. It also **uses the TLB in the CPU for high-speed access.** TLB stands for Translation Look-aside Buffer, which is an address conversion buffer.
  - \* The TLB associates the virtual memory space used by a program, with a physical memory space.
- **The TLB reconfiguration process (reloading of the address conversion table)** is required when a TLB miss occurs. The TLB reconfiguration process has a significant cost.

## TLB mechanism



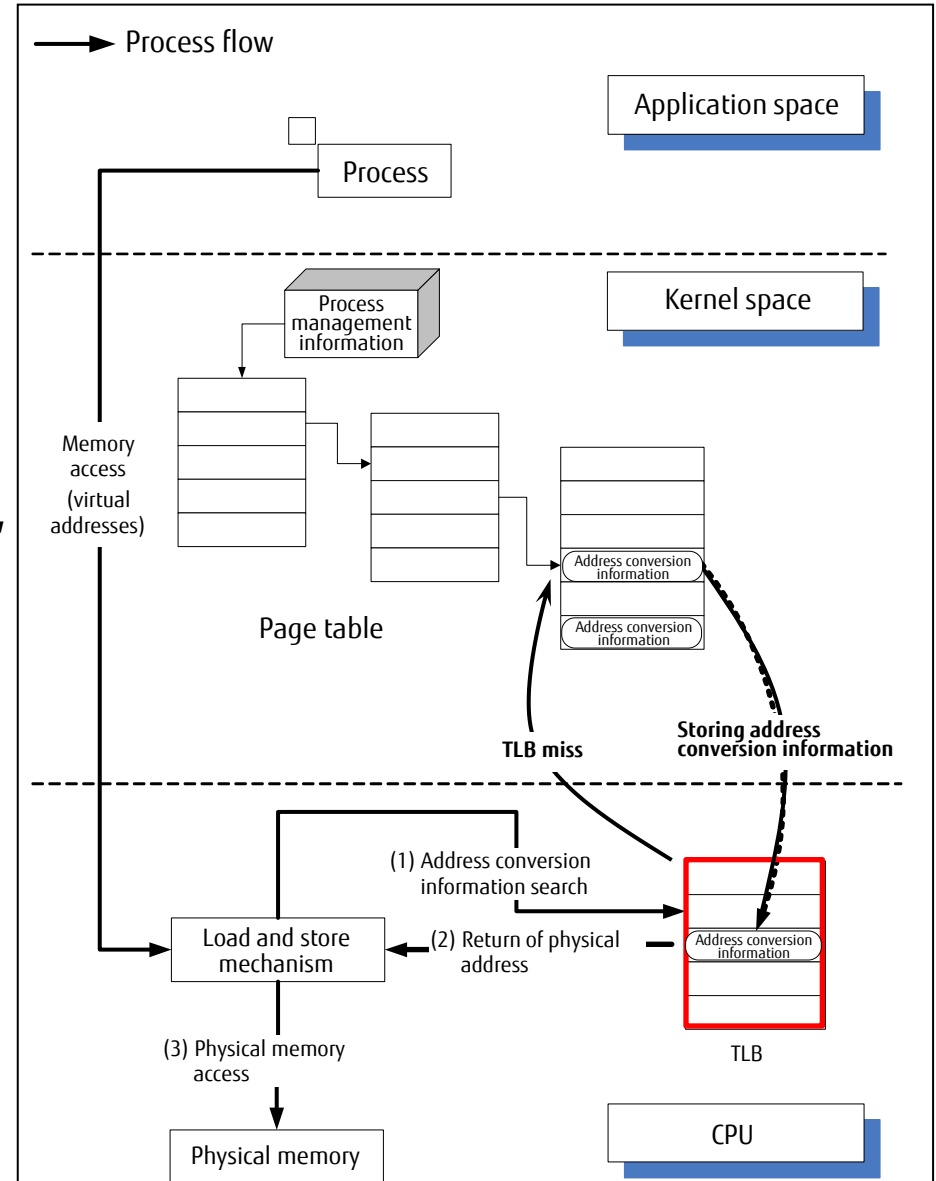
# What Is the TLB? (Details)

## ■ Address conversion by the TLB

The CPU receives a load and store request for memory from a running program.

- (1) The physical memory address corresponding to the virtual address specified as the load and store destination is determined from the TLB. The TLB initially contains no information, so (following TLB misses) the TLB will store corresponding address conversion information.
- (2) The relevant physical memory address is obtained from the TLB.
- (3) Physical memory access begins at the load and store destination address.

## Conversion from virtual address to physical address by the TLB





# TLB Configuration

## ■ sTLB and fTLB

There are two types of TLB: **sTLB** and **fTLB**.

**sTLB**: 512 entries, 4-way set associative

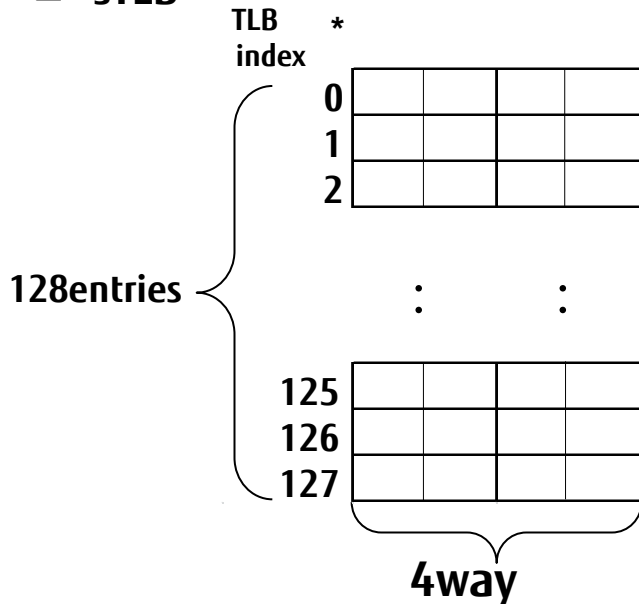
**fTLB**: 16 entries, full associative

## ■ A large number of entries in the sTLB reduces TLB misses and thus improves execution performance. (fTLB is also available.)

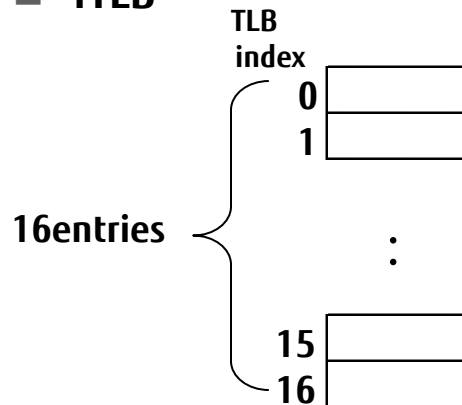
At the address conversion time, the sTLB of the set associative method is searched first. If the search has no hits, then the fTLB of the full associative method is searched.

The search sequentially browses the entries in the fTLB of the full associative method.

### ■ sTLB



### ■ fTLB



\* How to calculate the TLB index of the sTLB:  
 $\text{rounddown}(\text{mod}(\text{address} \div \text{page size}, \text{number of TLB entries}), 0)$

## ■ Default lpgparm parameters

If nothing is specified, the parameters are equivalent to:

```
lpgparm -s 4MB -t 4MB -d 4MB -h 4MB -p 4MB -S 4MB a.out
```

- You can make more TLB entries available by specifying that the large page size for the area used is the same as that for using the sTLB.

⇒ **Improved TLB hit ratio**

- A large page size of 4 MB can allocate a large amount of memory for use. (\*1)

⇒ **Improved memory usage ratio**

- The default large page size is 4 MB, which takes into consideration that **applications access memory continuously** with little (or no) performance degradation as a result. That large size provides high user convenience.

(\*1) For a large page size of 32 MB, an area of 32 MB is allocated. Even if only 1 MB is used for the thread stack, the remaining area is unused (31 MB). (If there are 16 threads, the unused area is multiplied by 16.)

You can reduce this unused, wasted area by changing the large page size to 4 MB.

# Aspects of Each Page Size

The following table shows advantages and disadvantages of page sizes of 8 KB, 4 MB, and 32 MB.

	8 KB	512 KB	4 MB (recommended)	32 MB
Execution performance (percentage of TLB misses)	<b>Poor</b>	<b>Good</b>	<b>Good to excellent</b>	<b>Excellent</b>
Overhead (memory initialization)	<b>Excellent</b>	<b>Good</b>	<b>Good</b>	<b>Fair</b>
Memory usage	<b>Excellent</b>	<b>Good</b>	<b>Good</b>	<b>Fair</b>

## ■ Page size of 8 KB

- This size neither reduces TLB misses nor improves execution performance.

## ■ Page size of 4 MB

- This size offers the best balance from the viewpoint of execution performance, memory usage, and overhead.

## ■ Page size of 32 MB

- This size uses a large amount of memory, so there is a possibility that some applications cannot be executed. Memory is likely to become insufficient, especially with Flat MPI.

# Page Size Selection Criteria

## ■ Default page size of 4 MB

- This size offers the best balance from the viewpoint of execution performance, memory usage, and overhead.

Aspects of each page size

	8 KB	512 KB	4 MB (recommended)	32 MB
Execution performance (percentage of TLB misses)	Poor	Good	Good to Excellent	Excellent
Overhead (memory initialization)	Excellent	Good	Good	Fair
Memory usage	Excellent	Good	Good	Fair

Reference values (relationship between execution performance and the percentage of TLB misses during sequential access)

Large page size	Execution performance ratio (32 MB/page ratio)	Percentage of TLB misses
512 KB	1.00	0.00139%
4 MB	1.00	0.00018%
32 MB	1.00	0.00001%

## ■ Page size expansion guideline

- If the percentage of TLB misses exceeds the following value, expand the large page size.

<b>Guideline value of percentage of mTLB misses</b>
1.5% or over

<b>Guideline value of percentage of <math>\mu</math>TLB misses</b>
2.5% or over

Generally, if the percentage of TLB misses exceeds the guideline value, even local tuning of the data of applications themselves should be reviewed.

# Functions of Large Pages

## How to Use Large Pages

- Ipgparm Command
- Demand Paging

## Area Management

- Different Areas Allocated for Different Variable Types
- Arena Process
- Thread Heap
- Binning (bin Expansion)

## Other

- Environment Variables Used for Tuning

## ■ lpgparm command

- You can specify a large page size from 8 KB, 512 KB, 4 MB, 32 MB, and default as appropriate to application characteristics, for high-speed execution of applications.
- You can specify a large page size when executing an application with the lpgparm command.
- How to use the command

```
lpgparm [-s process stack page size] [-t thread stack page size]
        [-d data page size] [-h heap page size]
        [-S shared memory page size]
        [-p sTLB page size]
        [-f floating-point register save and restore mode]
        [-l large page paging mode]
        [--help] program name program argument
```

- Default large page size values

The default value of -s, -t, -d, -h, -p, and -S is 4 MB.

Note: Executing only a.out is equivalent to executing lpgparm with the default value. a.out is assumed to be a load module created with -Klargepage specified at the compile time.

## ■ Specifying a page size for each area

■ Option: `-s pagesize -t pagesize -d pagesize -h pagesize -S pagesize`

- You can specify any of the values shown between the curly braces (default: 4MB):

`-s pagesize -t pagesize -d pagesize -h pagesize -S pagesize`

`{ 8KB | 512KB | 4MB | 32MB | default | auto }`

\* *auto* can be specified only with `-d`.

Function: Uses a stack segment area, thread stack area, data segment area, heap segment area, and shared memory area of the size specified in the respective *pagesize*.

Effect: Increases the size to reduce TLB misses and thus improve execution performance.

Note 1: Shared memory is related to the following interface.

System V IPC shared memory `shmget(2)/shmat(2)/shmdt(2)/shmctl(2)`

POSIX shared memory `shm_open(3)/shm_unlink(3)`

Note 2: For downward compatibility, you can specify that the page size be 256 MB.

When the specified page size is 256 MB, it works like it is 32 MB.

## ■ Specifying a page size for use with the sTLB

### ■ Option: `-p pagesize`

- You can specify any of the values shown between the curly braces (default: *4MB*):

`-p pagesize { 8KB | 512KB | 4MB | 32MB }`

Function: Uses the sTLB of the size specified in *pagesize*.

(Only a single page size can be used at a time.)

Effect: Uses the sTLB with many TLB entries to reduce TLB misses and thus improve execution performance.

Example: `lpgparm -s 32MB -t 32MB -d 32MB -h 32MB -p 32MB -S 32MB a.out`



## ■ Specifying paging when acquiring memory

### ■ Option: `-l pagingmode`

- You can specify any of the values shown between the curly braces (default: *use*):  
`-l pagingmode { none | use | demand | prepage }`

Function: If *use* or *prepage* is specified, use large pages and acquire memory by prepaging.

If *demand* is specified, use large pages and acquire memory by demand paging.

If *none* is specified, use normal pages and acquire memory by demand paging.

Effect: Improves execution performance through paging appropriate for memory acquisition.

Example: `lpgparm -l demand a.out`

\* For details on demand paging, see the next page (Demand Paging).

# Demand Paging

## ■ Example of demand paging when allocating a dynamic area (lpgparm -l demand)

- Prepaging (default)
  - Area acquired at the allocate time
- Demand paging (-l demand specified)
  - Area acquired at the initial area access time

```
program main
```

```
integer, parameter::N=(10*1024*1024+384)
```

```
real*8,allocatable::a(:),b(:),c(:)
```

```
...
```

```
allocate(a(N),b(N),c(N))
```

```
...
```

```
!$omp parallel do
```

```
do i=1,N
```

```
  c(i)=a(i)+b(i)*d
```

```
enddo
```

```
!$omp end parallel do
```

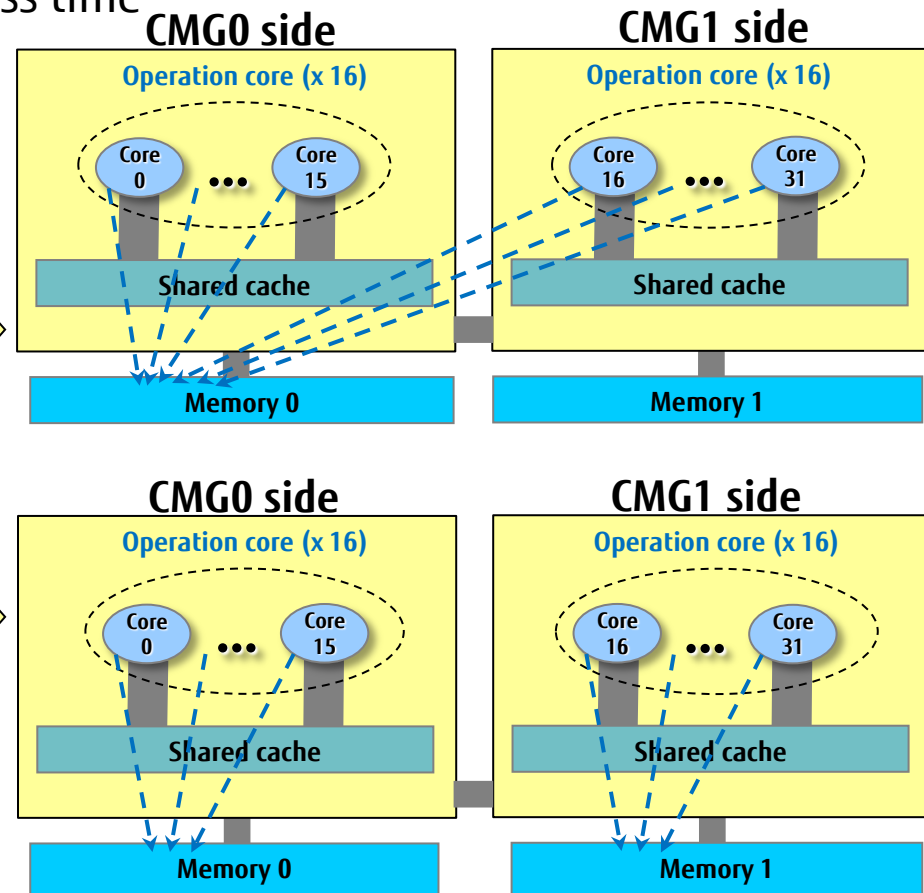
```
...
```

```
end
```

Prepaging allocates the array a, b, and c areas here. The areas are allocated on the CMG side where the allocate statement is executed.

Demand paging allocates the array a, b, and c areas here. Areas are also allocated on the CMG where each thread will be executed.

When dynamically acquiring an area, demand paging can be expected to improve performance for a 32-core thread parallelization program in some cases.



## ■ Different areas allocated for different variable types (Fortran)

```
program main
integer*8, parameter::N=(1024_8)
real*8 a(N)           !a is local array with no initial value

real*8 :: b (N)=1.0    !b is array with initial value
real*8,allocatable::c(:) !c is allocated array
allocate(c(N))
...
end
```

**"a" is in the data area (-d). However, for an executable program compiled with -Kauto or -Kthreadsafe specified, "a" is in the process stack area (-s).**

**"b" is in the data area (-d).**

**"c" is in the heap area (-h).**

Note: Data in the process stack area (-s) at the parallel execution time also uses the thread stack area (-t).

## ■ Different areas allocated for different variable types (C language)

```
#define N 1024
double a[N];      //a is global variable with no initial value
double b[N]={0.0}; //b is global variable with initial value
double *c;        //c is pointer variable

int main(void) {
    double d[N];    //d is local variable

    c=(double *)malloc(sizeof(double)*N);
    ...
}
```

**"a" is in the data area (-d).**

**"b" is in the data area (-d).**

**"c" is in the heap area (-h).**

**"d" is in the process stack area (-s).**

Note: Suppose you specify an executable program that is compiled with `array_private` specified together with `-Kparallel` or `-Kopenmp` at the parallel execution time. Then, data in the process stack area (-s) uses the thread stack (-t) as well.

## ■ Different areas allocated for different variable types (C++)

```
const int N = 1024;
struct Klass {
    double k;
    Klass() : k (0.0) {}
    Klass(double K) : k (K) {}
};

std::vector<Klass> a(N);           //Area reservation by vector class

int main(){
    Klass* b = new Klass[N];      //Area reservation by new operator
    std::vector<double> c(N);    //Area reservation by vector class
    return 0;
}
```

**"a," "b," and "c" are in the heap area (-h).**

Note: The areas allocated for variables are basically the same as in the C language, except that the vector class and other dynamically obtained areas are allocated in the heap area.

For example, variable a in the above example is allocated to the data area, but the content of "a" includes only information such as the beginning address and ending address of an array. The Klass-type array of the N element is allocated in the heap area.

## ■ What is an arena?

A memory area allocated in advance (memory pool) is used to improve processing performance for memory acquisition/deallocation requests. An arena is the combination of that memory pool and its management.

## ■ What is the arena process?

The arena process is the overall process for management (including reuse), etc. of the memory pool.

## ■ Effect of the arena process

- Reduction in the overhead of initialization
- Address tuning to prevent cache thrashing

## ■ Memory allocation process

- Memory is allocated from the memory pool (heap area) when there is a memory request from a user.
- For memory requested during thread execution, a memory pool (heap area) is allocated for every thread. (Thread heap)
- If the memory pool does not have an area of the requested size, the heap area is extended to allocate a sufficient area.

## ■ Arena deallocation process function

**The function instructs whether to deallocate an arena in response to a memory deallocation request.**

### ■ `XOS_MMM_L_ARENA_FREE=1` (default)

Function: Deallocates an arena in response to a memory deallocation request.

Effect: Can reduce memory usage.

### ■ `XOS_MMM_L_ARENA_FREE=2`

Function: Does not deallocate an arena in response to a memory deallocation request.

Effect: Can save the cost of memory acquisition/deallocation.

- \* To reduce memory usage,  
specify `XOS_MMM_L_ARENA_FREE=1`.
- To achieve acceleration,  
specify `XOS_MMM_L_ARENA_FREE=2`.

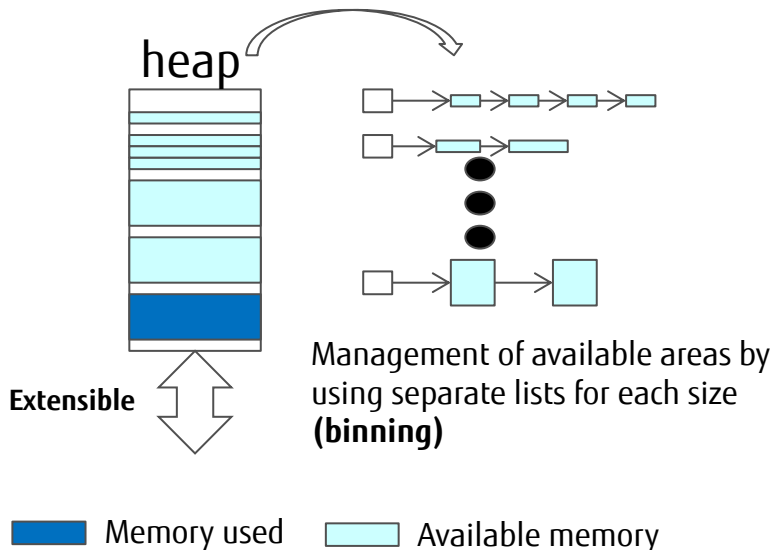
# Thread Heap (1/2)

## Basic operation of glibc malloc

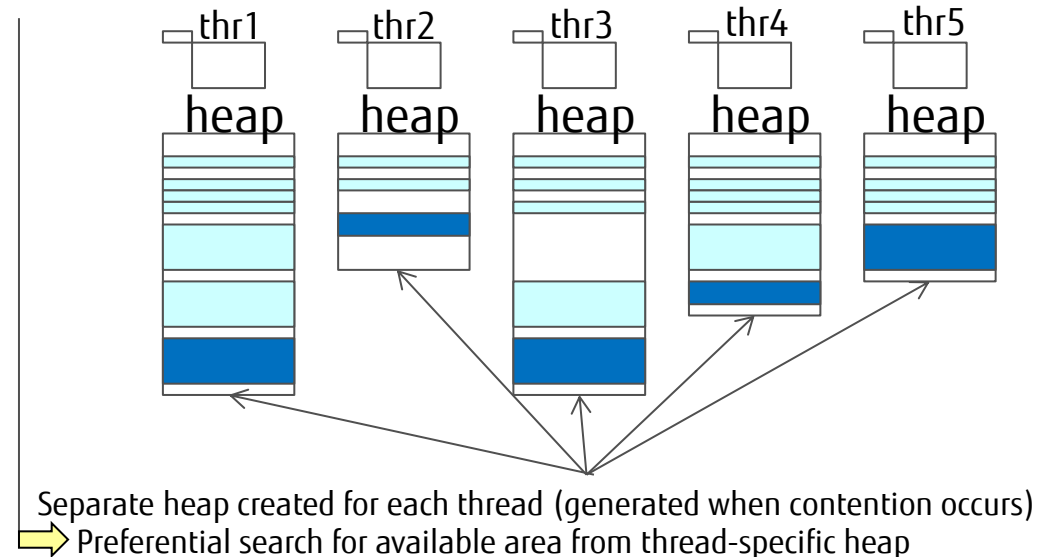
- Allocating available memory in a heap (extensible memory space)
- Managing deallocated memory areas with separate lists for each size. This results in high speed in searches for the next available memory of the same size (binning).
- Creating a separate heap area for each thread. The result is an on-board mechanism (thread heap) for parallel memory search even in cases of contention for malloc between threads.
- Large memory (128 MiB or more) is allocated by mmap, and the memory is instantly deallocated (mmap chunk) at the time of free.

\* With the environment variable `MALLOC_MMAP_THRESHOLD_`, you can specify the threshold of memory allocation by mmap. (The default is 134209536 (128 MiB - 8 KiB).)

Conceptual diagram of binning



Conceptual diagram of a thread heap





# Binning (bin Expansion)

- Large page libraries use memory areas of various sizes, including small areas for communication and large areas for computing.
- ➔ Sizes of available memory are categorized into nine steps (classes). Each class is provided with a memory management list of its own stride. This arrangement realizes high-speed search for available memory.

Size of available memory	Stride of available memory management list
32 B to 80 B	8 B
80 B to 1 KB	16 B
1 KB to 3 KB	64 B
3 KB to 10 KB	512 B
10 KB to 44 KB	4 KB
44 KB to 256 KB	32 KB
256 KB to 4 MB	256 KB
4 MB to 64 MB	4 MB
64 MB to 512 MB	64 MB
512 MB or more	None (management with single list)

# Thread Heap (2/2)

## ■ How to specify a thread heap

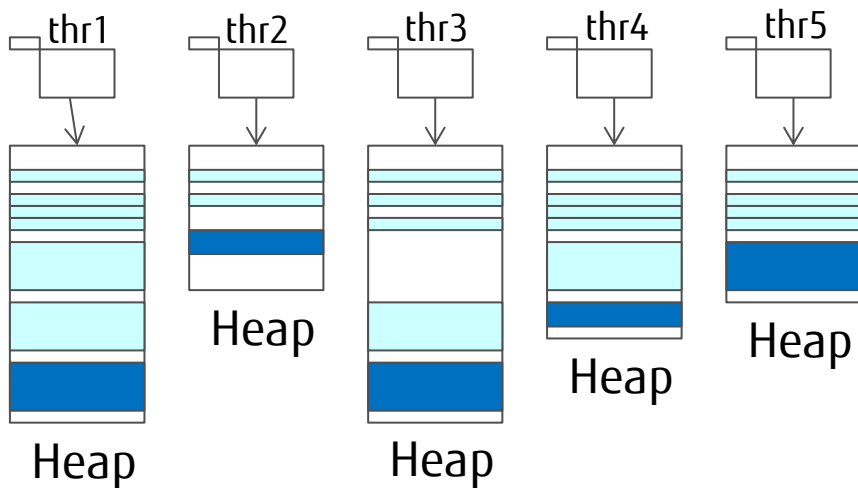
- The use of thread heaps by an HPC application as in (a) will produce many unused areas.

➡ There are fewer cases of simultaneous acquisition between threads (for an area for long-term use).  
(Most cases of contention between threads occur in temporary area acquisition.)

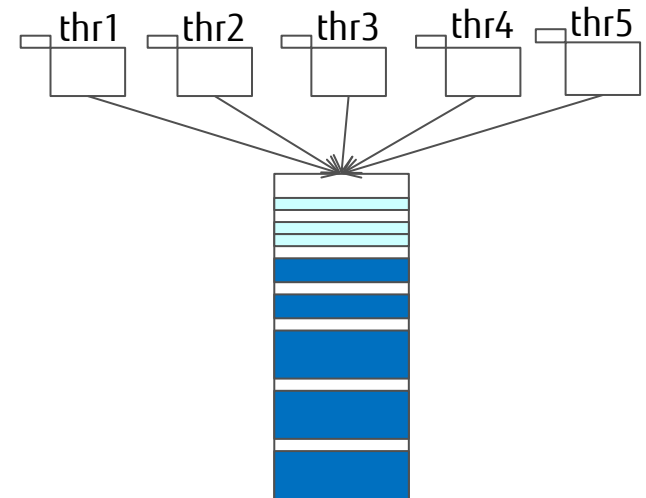
- By default, large page libraries are configured like in (b) to not generate a thread heap.

➡ This is because large page sizes significantly affect memory efficiency.

\* With the environment variable `XOS_MMM_L_ARENA_LOCK_TYPE`, you can specify whether to generate thread heaps. (The default is 1 (setting to not generate thread heaps).)



(a) Separate heaps allocated for individual threads  
`XOS_MMM_L_ARENA_LOCK_TYPE=0`



(b) Use of heap shared among threads  
`XOS_MMM_L_ARENA_LOCK_TYPE=1`

# Environment Variables Used for Tuning (1/3)

Variable name (default value)	Meaning	Purpose
MALLOC_MMAP_THRESHOLD_(134209536(128MiB-8KiB))	Threshold size for generating mmapped chunk	An mmapped chunk is created for a memory request to allocate memory of at least the size specified in this environment variable.
MALLOC_TRIM_THRESHOLD_(134217728(128MiB))	Specification of arena contraction baseline value	The arena contraction process takes place when the size of an available area retaining an arena (heap or thread heap) exceeds the value specified in this variable. If memory of 128 MiB or more is repeatedly acquired and deallocated, a setting value of 128 MiB or more in this variable can improve memory acquisition performance.
MALLOC_TOP_PAD_(131072)	Specification of extension size of heap area (in bytes)	An extension size is set for each extension of the heap area. The rolled-up value in the page size is applied. You can reduce page allocation and deallocation costs and improve performance by setting a value larger than the size of memory acquired and deallocated at one time by applications. (See also the description of MALLOC_TRIM_THRESHOLD_.)

# Environment Variables Used for Tuning (2/3)

Variable name (default value)	Meaning	Purpose
XOS_MMM_L_AR ENA_FREE (1)	1: Deallocatable memory pages instantly deallocated at the time of free  2: Memory pages not deallocated	The setting relates to the handling of heap areas deallocated by free or cfree. "1" gives priority to memory usage efficiency, and "2" gives priority to memory acquisition performance. The default value is 1. Values other than "1" and "2" are regarded as "1". For the value of 1, the area reserved in an mmaped chunk is instantly deallocated. A memory request for a size of less than 128 MiB has memory allocated from a heap area. That memory is not subject to instant deallocation. The deallocation process is executed for each available area of a size of more than 128 MiB retained in a heap area. For the value of "2", a memory request for a size of 128 MiB or more has memory allocated from a heap area. That memory will be retained as an available memory area even after being deallocated. The deallocation process is not executed for heap areas. Thread-specific heaps are not used because memory of all sizes must be allocated in a single heap area to prevent inefficient memory usage.
XOS_MMM_L_AR ENA_LOCK_TYPE (1)	Default value of 1 (sequential processing)	The setting relates to memory allocation policies. "0" gives priority to memory acquisition performance, and "1" gives priority to memory usage efficiency. The default value is "1". Values other than "0" and "1" are regarded as "1". For the value of "0", new mmaped arenas are created to process malloc requests in parallel when multiple malloc requests are called simultaneously. Consequently, memory usage efficiency may decrease. For the value of "1", the available search time of existing arenas may increase because of the sequential processing.
XOS_MMM_L_MA X_ARENA_NUM (1)	Specification of maximum number of arenas that can be generated (total number of process and thread heaps)	This variable is effective only when XOS_MMM_L_ARENA_LOCK_TYPE=1. You can set the number of arenas (the sum of the numbers of heaps and thread heaps) that can be generated. (The value of "1" specifies the use of only heap areas, so thread heaps are not generated.) Use this variable when you want to limit the number of thread heaps created. (The setting of XOS_MMM_L_ARENA_LOCK_TYPE=0 does not limit the number of thread heaps generated.) Note that using thread heaps improves memory acquisition performance through parallel processing of malloc but may degrade memory efficiency. Therefore, use this variable after studying the balance between performance and efficiency.

# Environment Variables Used for Tuning (3/3)

Variable name (default value)	Meaning	Purpose
XOS_MMM_L_H EAP_SIZE_MB (128)	Specification of expansion size of thread heaps	For use of thread heaps, this variable sets the size of memory acquired for thread heap generation and expansion. The default value is 128 MiB. A memory request for a size of 128 MiB or less has memory allocated from a thread heap. If the size of memory acquired from any thread is less than 128 MiB, you can improve the memory efficiency of the thread heap by setting a value of less than 128 in this environment variable.

# Evaluation Functions for Large Pages

- List of Evaluation Functions for Large Pages
- Large Page Memory Usage Information
- Large Page Statistics

# List of Evaluation Functions for Large Pages

- The advanced profiler (fapp) can collect and output the following large page information by:
  - Outputting large page memory usage information
  - Outputting large page statistics

Option name	Function and specification unit	Function overview
-l lpgusage   nolpgusage	<p>Gives an instruction to collect large page memory usage information.</p> <ul style="list-style-type: none"><li>· lpgusage: Collect large page memory usage information.</li><li>· nolpgusage: Do not collect large page memory usage information.</li></ul> <p>The default of this option is nolpgusage.</p> <p>This option cannot be specified together with the -lmpi, -lhwm, -llpgstats, or -H option.</p>	<p>Analyze the use conditions of memory in detail.</p> <p>Output the use conditions of the internal memory of each heap (arena).</p>
-l lpgstats   nolpgstats	<p>Gives an instruction to collect large page statistics.</p> <ul style="list-style-type: none"><li>· lpgstats: Collect large page statistics.</li><li>· nolpgstats: Do not collect large page statistics.</li></ul> <p>The default of this option is nolpgstats.</p> <p>This option cannot be specified together with the -lmpi, -lhwm, -llpgusage, or -H option.</p>	<p>Identify whether a performance problem is due to a large page.</p> <p>Output statistics, such as the malloc issuance count and processing time.</p>

# Large Page Memory Usage Information

- How to Use Large Page Memory Usage Information



- Function: Output the use conditions of memory for each process in detail.
- Purpose: You can use the information to check for disparities in memory usage ratio between processes, such as if an application fails only in a specific process.
- How to use
  - (1) Specify the `-l lpgusage` option in the `fapp` command to collect advanced profiling data.
  - (2) Specify the `-l lpg` option in the `fapppx` command to analyze the collected advanced profiling data.
- Collecting large page statistics (Write the following command in a job script.)

```
fapp -C -d data_lpgusage -l lpgusage a.out
```
- Outputting large page statistics (Execute the following command on a login node.)

```
fapppx -A -l lpg -d data_lpgusage
```

The output advanced profiling data is assumed to be `data_lpgusage`.

# Large Page Statistics

- How to Use Large Page Statistics
- Usage Example of Large Page Statistics

- Function: Output statistics, such as the malloc issuance count and processing time.
- Purpose: Identify whether a performance problem is due to a large page.
- How to use
  - (1) Specify the `-l lpgstats` option in the `fapp` command to collect advanced profiling data.
  - (2) Specify the `-l lpg` option in the `fapppx` command to analyze the collected advanced profiling data.
- Collecting large page statistics (Write the following command in a job script.)

```
fapp -C -d data_lpgstats -l lpgstats a.out
```
- Outputting large page statistics (Execute the following command on a login node.)

```
fapppx -A -l lpg -d data_lpgstats
```

The output advanced profiling data is assumed to be `data_lpgstats`.

# Usage Example of Large Page Statistics (1/2)

## ■ Phenomenon

Deterioration of application performance

## ■ Data to collect

Collect and analyze large page statistics.

## ■ Analysis step

- Check whether AVG of malloc is within several tens of microseconds.

Value of over 100 microseconds indicates potential deterioration in memory acquisition performance

Kind	Count	TimeMin	TimeMax	TimeAvg	SizeMin	SizeMax	SizeAvg		
AVG	100417	0	96101	174	1	405798912	1351723	malloc	all 0
MAX	100417	0	96101	174	1	405798912	1351723		
MIN	100417	0	96101	174	1	405798912	1351723		
AVG	0	0	0	0	0	0	0	calloc	
MAX	0	0	0	0	0	0	0		
MIN	0	0	0	0	0	0	0		
- snip -									
Kind	NewHeap	DelHeap	Expbrk	Exp mmap	Newmmap	Delmmap			
AVG	0	0	3903	0	0	0	0	all 0	
MAX	0	0	3903	0	0	0	0		
MIN	0	0	3903	0	0	0	0		

Possible frequent heap expansion/contraction

# Usage Example of Large Page Statistics (2/2)

## ■ Means of improvement

Thrashing may be occurring because of frequent heap expansion/contraction, so expand the available areas retained for the library. → Set the value of the environment variable `MALLOC_TRIM_THRESHOLD_` to 128 MB or more.

## ■ Execution example

- 256 MB is set in `MALLOC_TRIM_THRESHOLD_` with the following line added to the job script.

```
export MALLOC_TRIM_THRESHOLD_=268435456
```

- Check whether AVG of malloc is within several tens of microseconds.

Kind	Count	TimeMin	TimeMax	TimeAvg	SizeMin	SizeMax	SizeAvg		
AVG	100417	0	96101	10	1	405798912	1351723	malloc	all 0
MAX	100417	0	96101	10	1	405798912	1351723		
MIN	100417	0	96101	10	1	405798912	1351723		
-----									
AVG	0	0	0	0	0	0	0	calloc	
MAX	0	0	0	0	0	0	0		
MIN	0	0	0	0	0	0	0		
- snip -									
Kind	NewHeap	DelHeap	Expbrk	Expmmmap	Newmmap	Delmmap			
AVG	0	0	6	0	0	0			
MAX	0	0	6	0	0	0			
MIN	0	0	6	0	0	0			


Annotations: A red circle highlights the 'TimeAvg' value of 10 for the first 'malloc' row, with an arrow pointing to a box labeled 'Within 10 microseconds'. Another red circle highlights the 'Expbrk' value of 6 for the 'Heap' section, with an arrow pointing to a box labeled 'Heap expansion/contraction count decreased'.

## ■ Important point

- Raising the `MALLOC_TRIM_THRESHOLD_` setting value may increase memory usage. Perform tuning with consideration of the balance with acquisition performance.

# Revision History

Version	Date	Revised section	Details
2.0	April 25, 2016	-	- First published



**FUJITSU**

shaping tomorrow with you