Multiple Architecture compiles

Timothy H. Kaiser, Ph.D
tkaiser@mines.edu

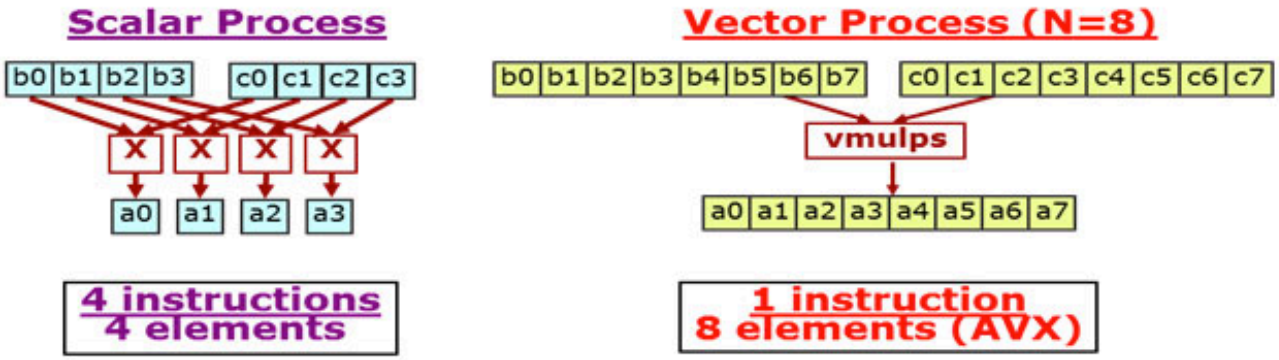
What and Why

- Successive generations of Intel Processors add CPU instructions
- Some important additions, vector instructions, improve performance, especially for loops.
- Mio has several generations of processor
- To get best performance you need to build for a specific processor
- Code built for a later processor may not run on an earlier processor

A simple loop

The generic multiplyValues example helps to illustrate the difference between scalar and vector process using Intel® AVX.

```
1  void multiplyValues(float *a, float *b, float * c, int size)
2  {
3      for (i = 0; i < size; i++) {
4          a[i] = b[i] * c[i];
5      }
6  }
```



Useful Commands

```
cat /proc/cpuinfo

/opt/utility/jlines

scontrol show nodes

/opt/utility/slurmnodes

[tkaiser@mio001 ~]$ /opt/utility/slurmnodes --help
/opt/utility/slurmnodes
     Options:
        Without arguments show full information for all nodes
       -fATTRIBUTE
           Show only the given attribute, without ATTRIBUTE just list the nodes
        list of nodes
           Show information for the given nodes
        -h
           Show this help

     Author:
        Timothy H. Kaiser, Ph.D.
        February 2014
[tkaiser@mio001 ~]$

slurmnodes -fActiveFeatures | /opt/utility/jlines 2 | grep broadwell

cat /proc/cpuinfo | grep flags | sort -u
```

srun -n 1 --constraint=broadwell ./eigen 3.0 7.0 8 < sort2.in

Mio nodes Generations and added Instructions

```
mio001 X5355
    [ssse3]

compute000 nehalem X5570
    [ rdtscp , xtopology , nonstop_tsc , sse4_1 , sse4_2 , popcnt , ida , ept ,
vpid ]

compute032 westmere X5670
    [ pdpe1gb , smx , pcid , arat , epb ]

compute102 sandybridge E5-2680 0
    [ pclmulqdq , x2apic , tsc_deadline_timer , aes , xsave , avx , xsaveopt ,
pln , pts ]

compute126 ivybridge E5-2680 v2
    [ f16c , rdrand , fsgsbase , smep , erms ]

compute132 haswell E5-2680 v3
    [ fma , movbe , abm , bmi1 , avx2 , bmi2 , invpcid , cqm , cqm_llc ,
cqm_occup_llc ]

compute180 broadwell E5-2680 v4
    [ aes , 3dnowprefetch , hle , rtm , rdseed , adx ]
```

srun -n 1 --constraint=broadwell ./eigen 3.0 7.0 8 < sort2.in

Compile Options for specific processors

-xtarget
Generates specialized code for any Intel® processor that supports the instruction set specified by target. The executable will not run on non-Intel processors or on Intel processors that support only lower instruction sets. Possible values of target, from highest to lowest instruction set: CORE-AVX512, MIC-AVX512, COMMON-AVX512, CORE-AVX2, AVX, SSE4.2, ATOM_SSE4.2, SSE4.1, ATOM_SSSE3, SSSE3, SSE3, SSE2

-axtarget
May generate specialized code for any Intel processor that supports the instruction set specified by target, while also generating a default code path. Possible values of target : CORE-AVX512, MIC-AVX512, COMMON-AVX512, CORE-AVX2, AVX, SSE4.2, SSE4.1, SSSE3, SSE3, SSE2
Multiple values, separated by commas, may be used to tune for additional Intel processors in the same executable, e.g. -axAVX,SSE4.2. The default code path will run on any Intel or compatible, non-Intel processor that supports at least SSE2, but may be modified by using in addition a (-x) switch.

examples/examples/eigen

examples/examples/stommel

Things to try…

- -xSSSE3
  - should run anywhere including head node
- -xSSE4.2
  - should run on any compute node but not head node
- -xAVX
  - should run on any compute node over compute102
- -xCORE-AVX2
  - should run on any compute node over compute132
- -xCORE-AVX512
  - will not run on any current Mio nodes, next week?

Our makefile

```
#make serial version of stommel with various settings for ${ARCH}
default: stc_00 stf_00
c:  stc_00
f:  stf_00

SFC=ifort
FFLAGS=-O3 ${ARCH}

SCC=icc
CFLAGS= -O3 ${ARCH}


stc_00:  stc_00.c
  $(SCC) $(CFLAGS) stc_00.c -lm -o stc_00
  ls -lt stc_00

stf_00:  stf_00.f90
  $(SFC) $(FFLAGS) stf_00.f90 -o stf_00
  ls -lt stf_00


clean:
  /bin/rm -f *mod stc_00 stf_00
```

We can build for a particular architecture by setting the environmental variable ARCH.

For example

export ARCH="-xSSSE3"

More things to try…

- Nothing
- -xSSSE3
- -xAVX2 -axSSSE3
- -axAVX2 -xSSSE3
- -axCORE-AVX512,COMMON-AVX512,CORE-AVX2,AVX,SSE4.2,SSE4.1,SSSE3,SSE3,SSE2
- -xAVX2 -axCORE-AVX512,COMMON-AVX512,CORE-AVX2,AVX,SSE4.2,SSE4.1,SSSE3,SSE3,SSE2

srun -n 1 --constraint=broadwell ./eigen 3.0 7.0 8 < sort2.in

Memory tracking

Timothy H. Kaiser, Ph.D
tkaiser@mines.edu

Why?

- Find out how much memory your program is using over time
- Why?
  - Find bugs
  - How big can you grow your problem size

How?

- Debuggers and trace packages (Forge ddt)
- Ganglia
  - http://tuyo.mines.edu/ganglia/
  - http://mindy.mines.edu/ganglia
- External tools
- We will discuss "In code"
  - Special subroutines
  - Special libraries
  - http://geco.mines.edu/prototype/How_do_I_track_memory_usage/
  - wget http://geco.mines.edu/prototype/How_do_I_track_memory_usage/stuff.tgz

mallinfo and malloc_info

- Similar
  - http://man7.org/linux/man-pages/man3/mallinfo.3.html
    - Subroutine that returns information
  - http://man7.org/linux/man-pages/man3/malloc_info.3.html
    - Subroutine that prints information in XML format
  - Our examples programs are similar to theirs

mallinfo

```
The mallinfo() function returns a copy of a structure containing
information about memory allocations performed by malloc(3) and
related functions.  This structure is defined as follows:

   struct mallinfo {
       int arena;     /* Non-mmapped space allocated (bytes) */
       int ordblks;   /* Number of free chunks */
       int smblks;    /* Number of free fastbin blocks */
       int hblks;     /* Number of mmapped regions */
       int hblkhd;    /* Space allocated in mmapped regions (bytes) */
       int usmblks;   /* Maximum total allocated space (bytes) */
       int fsmblks;   /* Space in freed fastbin blocks (bytes) */
       int uordblks;  /* Total allocated space (bytes) */
       int fordblks;  /* Total free space (bytes) */
       int keepcost;  /* Top-most, releasable space (bytes) */
   };
```

Program has a subroutine that calls malling and prints info

1. Gets number of blocks to allocate and size
2. Calls subroutine
3. Allocates memory
4. Calls subroutine
5. Frees memory
6. Calls subroutine

mallinfo example output

```
[tkaiser@mio001 linux]$ cat mallinfo.4008079
150 1000000
============== Before allocating blocks ==============
Total non-mmapped bytes (arena):       0
# of free chunks (ordblks):            1
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           0
Bytes in mapped regions (hblkhd):      0
Max. total allocated space (usmblks):  0
Free bytes held in fastbins (fsmblks): 0
Total allocated space (uordblks):      0
Total free space (fordblks):           0
Topmost releasable block (keepcost):   0

============== After allocating blocks ==============
Total non-mmapped bytes (arena):       0
# of free chunks (ordblks):            1
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           150
Bytes in mapped regions (hblkhd):      150528000
Max. total allocated space (usmblks):  0
Free bytes held in fastbins (fsmblks): 0
Total allocated space (uordblks):      0
Total free space (fordblks):           0
Topmost releasable block (keepcost):   0

============== After freeing blocks ==============
Total non-mmapped bytes (arena):       0
# of free chunks (ordblks):            1
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           0
Bytes in mapped regions (hblkhd):      0
Max. total allocated space (usmblks):  0
Free bytes held in fastbins (fsmblks): 0
Total allocated space (uordblks):      0
Total free space (fordblks):           0
Topmost releasable block (keepcost):   0
[tkaiser@mio001 linux]$
```

malloc_info

- Subroutine call - you pass it a FILE to write to
- “Prints” current state of memory as XML
- Cool thing - works on a thread level

malloc_info

```
malloc_info(0, stdout);
```

Program uses malloc_info

1. Gets number of threads to run and and size
2. Calls subroutine
3. Creates threads
4. Allocates memory
5. Calls subroutine

malloc_info

```
thr = calloc(numThreads, sizeof(pthread_t));
  if (thr == NULL)
     errExit("calloc");

  printf("============ Before allocating blocks ============\n");
  malloc_info(0, stdout);

  /* Create threads that allocate different amounts of memory */

  for (tn = 0; tn < numThreads; tn++) {
     errno = pthread_create(&thr[tn], NULL, thread_func,
                         (void *) tn);
     if (errno != 0)
        errExit("pthread_create");

     /* If we add a sleep interval after the start-up of each
*      thread, the threads likely won't contend for malloc
*           mutexes, and therefore additional arenas won't be
*                allocated (see malloc(3)). */

     if (sleepTime > 0)
        fsleep((float)sleepTime);
  }

  /* The main thread also allocates some memory */

  for (j = 0; j < numBlocks; j++)
     if (malloc(blockSize) == NULL)
        errExit("malloc");

  fsleep(2.0);           /* Give all threads a chance to
                   complete allocations */

  printf("\n============ After allocating blocks ============\n");
  malloc_info(0, stdout);
```

malloc_info - 16 threads

```
[tkaiser@mio001 linux]$ cat minfo.4008079
============ Before allocating blocks ============
<malloc version="1">
<heap nr="0">
<sizes>
</sizes>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<system type="current" size="135168"/>
<system type="max" size="135168"/>
<aspace type="total" size="135168"/>
<aspace type="mprotect" size="135168"/>
</heap>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<total type="mmap" count="0" size="0"/>
<system type="current" size="135168"/>
<system type="max" size="135168"/>
<aspace type="total" size="135168"/>
<aspace type="mprotect" size="135168"/>
</malloc>

============ After allocating blocks ============
<malloc version="1">
<heap nr="0">
<sizes>
</sizes>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<system type="current" size="135168"/>
<system type="max" size="135168"/>
<aspace type="total" size="135168"/>
<aspace type="mprotect" size="135168"/>
</heap>
<heap nr="1">
<sizes>
</sizes>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<system type="current" size="2134016"/>
<system type="max" size="2134016"/>
<aspace type="total" size="2134016"/>
<aspace type="mprotect" size="2134016"/>
</heap>
************************************************
<heap nr="15">
<sizes>
</sizes>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<system type="current" size="6135808"/>
<system type="max" size="6135808"/>
<aspace type="total" size="6135808"/>
<aspace type="mprotect" size="6135808"/>
</heap>
<heap nr="16">
<sizes>
</sizes>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<system type="current" size="5136384"/>
<system type="max" size="5136384"/>
<aspace type="total" size="5136384"/>
<aspace type="mprotect" size="5136384"/>
</heap>
<total type="fast" count="0" size="0"/>
<total type="rest" count="0" size="0"/>
<total type="mmap" count="154" size="1378316288"/>
<system type="current" size="154296320"/>
<system type="max" size="154296320"/>
<aspace type="total" size="154296320"/>
<aspace type="mprotect" size="154296320"/>
</malloc>
```

malloc_info - 16 threads

```
[tkaiser@mio001 linux]$ cat minfo.4008079 | egrep "nr=|current"
<heap nr="0">
<system type="current" size="135168"/>
<system type="current" size="135168"/>
<heap nr="0">
<system type="current" size="135168"/>
<heap nr="1">
<system type="current" size="2134016"/>
<heap nr="2">
<system type="current" size="3133440"/>
<heap nr="3">
<system type="current" size="4136960"/>
<heap nr="4">
<system type="current" size="17133568"/>
<heap nr="5">
<system type="current" size="16134144"/>
<heap nr="6">
<system type="current" size="15134720"/>
<heap nr="7">
<system type="current" size="14135296"/>
<heap nr="8">
<system type="current" size="13135872"/>
<heap nr="9">
<system type="current" size="12136448"/>
<heap nr="10">
<system type="current" size="11137024"/>
<heap nr="11">
<system type="current" size="10133504"/>
<heap nr="12">
<system type="current" size="9134080"/>
<heap nr="13">
<system type="current" size="8134656"/>
<heap nr="14">
<system type="current" size="7135232"/>
<heap nr="15">
<system type="current" size="6135808"/>
<heap nr="16">
<system type="current" size="5136384"/>
<system type="current" size="154296320"/>
[tkaiser@mio001 linux]$
```

Replacing malloc realloc, calloc, & free

- Replaces standard memory routine with tracking versions
- Most versions have utility routines to print information
- Most will automatically print a report at the end of a run
- We look at:
  - https://panthema.net/2013/malloc_count/
- Have not been able to get this to work on Mc2
- With minor mods works with Fortran

User Functions

```
/* user function to return the currently allocated amount of memory */
    extern size_t malloc_count_current(void)
    extern size_t malloc_count_current_(void)

/* user function to return the peak allocation */
    extern size_t malloc_count_peak(void)
    extern size_t malloc_count_peak_(void)

/* user function to reset the peak allocation to current */
    extern void malloc_count_reset_peak(void)
    extern void malloc_count_reset_peak_(void)

/* "clear" the stack by writing a sentinel value into it. */
void* stack_count_clear(void)
void* stack_count_clear_(void)

/* checks the maximum usage of the stack since the last clear call. */
size_t stack_count_usage(void* lastbase)
size_t stack_count_usage_(void* lastbase)


/* user function which prints current and peak allocation to stderr */
extern void malloc_count_print_status(void)
```

Fortran Example

Program
1. Gets a base memory
2. Allocates a big block
3. Prints information
4. Deallocates
5. Prints information
6. Allocates a smaller block
7. Prints information
8. Tracks stack information while recursively calling a subroutine
9. Tracks stack information while unwinding recursive calls
10. One more call to print information
11. Prints report on exit

Fortran Example

```
module tracking
    integer, parameter:: i8 = selected_int_kind(14)
end module
program xyz
    use tracking
    implicit none
    integer(i8) malloc_count_peak,malloc_count_current
    integer(i8) stack_count_usage,stack_count_clear
    integer(i8)base
    integer n
    real x
    real, allocatable :: block(:)
    base=stack_count_clear()
    write(*,'(20x,"            ",2a16)')"peak","current"
    write(*,'(20x,"      before",2i16)')malloc_count_peak(),malloc_count_current()
    allocate(block(10000000))
    block=1
    write(*,'(20x,"   allocated",2i16)')malloc_count_peak(),malloc_count_current()
    deallocate(block)
    write(*,'(20x," deallocated",2i16)')malloc_count_peak(),malloc_count_current()
    allocate(block(1000))
    write(*,'(20x," reallocated",2i16)')malloc_count_peak(),malloc_count_current()
    n=1
    x=0
    base=stack_count_clear()
    write(*,'("stack before",i8)')stack_count_usage(base)
    call sumit(n,x,base)
! note: the stack_count_usage call gives max stack size, not current
    write(*,'(" stack after",i8)')stack_count_usage(base)
    write(*,'(20x,"  after call",2i16)')malloc_count_peak(),malloc_count_current()
    write(*,*)x

end program
```

Fortran Example

```
recursive subroutine sumit(n,x,base)
    use tracking
    integer(i8) malloc_count_peak,malloc_count_current
    integer(i8) stack_count_usage,stack_count_clear
    integer n
    real x
    integer(i8) base
    real  :: sblock(10000)
    sblock=1
    x=sum(sblock)+x
    write(*,'("stack inside",i8,1x,i2)')stack_count_usage(base),n
    n=n+1
    if(n < 10 )then
      call sumit(n,x,base)
      base=stack_count_clear()
      write(*,'("   unwinding",i8,1x,i2)')stack_count_usage(base),n
    endif
end
```

Fortran example

```
[tkaiser@mio001 test-malloc_count]$ cat ftest.out.4008084
                                          peak         current
                        before            8771            8771
                     allocated        40008811        40008811
                   deallocated        40008811            8771
                   reallocated        40008811           12811
stack before     224
stack inside    4296  1
stack inside   40272  2
stack inside   80320  3
stack inside  120368  4
stack inside  160416  5
stack inside  200464  6
stack inside  240512  7
stack inside  280560  8
stack inside  320608  9
   unwinding  280560 10
   unwinding  240512 10
   unwinding  200464 10
   unwinding  160416 10
   unwinding  120368 10
   unwinding   80320 10
   unwinding   40272 10
   unwinding     224 10
 stack after    4224
                    after call        40008811           12811
   90000.00
malloc_count ### exiting, total: 40013331, peak: 40008811, current: 4320
[tkaiser@mio001 test-malloc_count]$
```

Blue Gene (Mc2) Kernel_GetMemorySize

- Blue Gene has its own routine Kernel_GetMemorySize
  - Takes as input a type of query
  - Returns values
  - IBM provides a nice wrapper

Kernel_GetMemorySize wrapper

```
/* compile instructions:
 * mpicc -std=gnu99 -c -g -O0 -Wall fortran_memory.c */

#include <spi/include/kernel/memory.h>
#include <spi/include/kernel/location.h>

/* u = used
 * a = available */

void memory_info(double * heapu, double * stacku, double * heapa, double * stacka)
{
  //uint64_t shared, persist, guard, mmap;
  //Kernel_GetMemorySize(KERNEL_MEMSIZE_SHARED,   &shared);
  //Kernel_GetMemorySize(KERNEL_MEMSIZE_PERSIST,  &persist);
  //Kernel_GetMemorySize(KERNEL_MEMSIZE_GUARD,    &guard);
  //Kernel_GetMemorySize(KERNEL_MEMSIZE_MMAP,     &mmap);

  uint64_t heap_used, stack_used, heap_avail, stack_avail;
  Kernel_GetMemorySize(KERNEL_MEMSIZE_HEAP,       &heap_used);
  Kernel_GetMemorySize(KERNEL_MEMSIZE_STACK,      &stack_used);
  Kernel_GetMemorySize(KERNEL_MEMSIZE_HEAPAVAIL,  &heap_avail);
  Kernel_GetMemorySize(KERNEL_MEMSIZE_STACKAVAIL, &stack_avail);

  *heapu  = (double) heap_used;
  *stacku = (double) stack_used;
  *heapa  = (double) heap_avail;
  *stacka = (double) stack_avail;

  return;
}

void memory_info_(double * heapu, double * stacku, double * heapa, double * stacka)
{
    memory_info(heapu, stacku, heapa, stacka);
    return;
}
```

Kernel_GetMemorySize Example

Program

1. Allocates bigger and bigger blocks of memory until we run out while tracking usage
2. Tracks stack information while recursively calling a subroutine
3. Tracks stack information while unwinding recursive calls

Kernel_GetMemorySize Example

```
 IBM XL Fortran for Blue Gene, V14.1 (5799-AH1) Version 14.01.0000.0013
                          heap used   stack used   heap avail   stack avail
check   10                     4096   290.816E+03    19.231E+03    17.079E+09    17.079E+09
check   11                     8192   290.816E+03    19.231E+03    17.079E+09    17.079E+09
check   12                    16384   290.816E+03    19.231E+03    17.079E+09    17.079E+09
check   13                    32768   290.816E+03    19.231E+03    17.079E+09    17.079E+09
check   14                    65536   290.816E+03    19.231E+03    17.079E+09    17.079E+09
check   15                   131072   430.080E+03    19.231E+03    17.079E+09    17.079E+09
check   16                   262144   430.080E+03    19.231E+03    17.079E+09    17.079E+09
check   17                   524288   823.296E+03    19.231E+03    17.078E+09    17.078E+09
check   18                  1048576     2.101E+06    19.231E+03    17.077E+09    17.077E+09
check   19                  2097152     3.150E+06    19.231E+03    17.076E+09    17.076E+09
check   20                  4194304     5.247E+06    19.231E+03    17.074E+09    17.074E+09
check   21                  8388608     9.441E+06    19.231E+03    17.070E+09    17.070E+09
check   22                 16777216    17.830E+06    19.231E+03    17.061E+09    17.061E+09
check   23                 33554432    34.607E+06    19.231E+03    17.045E+09    17.045E+09
```

Kernel_GetMemorySize Example

```
check   24       67108864    68.162E+06    19.231E+03    17.011E+09    17.011E+09
check   25      134217728   135.270E+06    19.231E+03    16.944E+09    16.944E+09
check   26      268435456   269.488E+06    19.231E+03    16.810E+09    16.810E+09
check   27      536870912   537.924E+06    19.231E+03    16.541E+09    16.541E+09
check   28     1073741824     1.075E+09    19.231E+03    16.004E+09    16.004E+09
check   29     2147483648     2.149E+09    19.231E+03    14.931E+09    14.931E+09
check   30     3221225472     3.222E+09    19.231E+03    13.857E+09    13.857E+09
check   31     4294967296     4.296E+09    19.231E+03    12.783E+09    12.783E+09
check   32     5368709120     5.370E+09    19.231E+03    11.709E+09    11.709E+09
check   33     6442450944     6.444E+09    19.231E+03    10.636E+09    10.636E+09
check   34     7516192768     7.517E+09    19.231E+03     9.562E+09     9.562E+09
check   35     8589934592     8.591E+09    19.231E+03     8.488E+09     8.488E+09
check   36     9663676416     9.665E+09    19.231E+03     7.414E+09     7.414E+09
check   37    10737418240    10.738E+09    19.231E+03     6.341E+09     6.341E+09
check   38    11811160064    11.812E+09    19.231E+03     5.267E+09     5.267E+09
check   39    12884901888    12.886E+09    19.231E+03     4.193E+09     4.193E+09
check   40    13958643712    13.960E+09    19.231E+03     3.119E+09     3.119E+09
check   41    15032385536    15.033E+09    19.231E+03     2.046E+09     2.046E+09
check   42    16106127360    16.107E+09    19.231E+03   972.007E+06   972.007E+06
 allocation error 17179869184
```

Kernel_GetMemorySize Example

```
stack inside   59711  0
stack inside   99999  1
stack inside  140287  2
stack inside  180575  3
stack inside  220863  4
stack inside  261151  5
stack inside  301439  6
stack inside  341727  7
stack inside  382015  8
stack inside  422303  9
   unwinding  382015 10
   unwinding  341727 10
   unwinding  301439 10
   unwinding  261151 10
   unwinding  220863 10
   unwinding  180575 10
   unwinding  140287 10
   unwinding   99999 10
   unwinding   59711 10
 100000.0000
```

NAME | SYNOPSIS | DESCRIPTION | ATTRIBUTES | CONFORMING TO | BUGS | EXAMPLE | SEE ALSO | COLOPHON

MALLINFO(3) Linux Programmer's Manual MALLINFO(3)

NAME top

mallinfo - obtain memory allocation information

SYNOPSIS top

```
#include <malloc.h>

struct mallinfo mallinfo(void);
```

DESCRIPTION top

The **mallinfo**() function returns a copy of a structure containing information about memory allocations performed by malloc(3) and related functions. This structure is defined as follows:

```
struct mallinfo {
    int arena;     /* Non-mmapped space allocated (bytes) */
    int ordblks;   /* Number of free chunks */
    int smblks;    /* Number of free fastbin blocks */
    int hblks;     /* Number of mmapped regions */
    int hblkhd;    /* Space allocated in mmapped regions (bytes) */
    int usmblks;   /* Maximum total allocated space (bytes) */
    int fsmblks;   /* Space in freed fastbin blocks (bytes) */
    int uordblks;  /* Total allocated space (bytes) */
    int fordblks;  /* Total free space (bytes) */
    int keepcost;  /* Top-most, releasable space (bytes) */
};
```

The fields of the *mallinfo* structure contain the following information:

arena The total amount of memory allocated by means other than mmap(2) (i.e., memory allocated on the heap). This figure includes both in-use blocks and blocks on the free list.

ordblks The number of ordinary (i.e., non-fastbin) free blocks.

smblks The number of fastbin free blocks (see mallopt(3)).

hblks The number of blocks currently allocated using mmap(2). (See the discussion of **M_MMAP_THRESHOLD** in mallopt(3).)

hblkhd The number of bytes in blocks currently allocated using mmap(2).

usmblks The "highwater mark" for allocated space—that is, the maximum amount of space that was ever allocated. This field is maintained only in nonthreading environments.

fsmblks The total number of bytes in fastbin free blocks.

uordblks The total number of bytes used by in-use allocations.

fordblks The total number of bytes in free blocks.

keepcost The total amount of releasable free space at the top of the heap. This is the maximum number of bytes that could ideally (i.e., ignoring page alignment restrictions, and so on) be released by malloc_trim(3).

ATTRIBUTES top

For an explanation of the terms used in this section, see attributes(7).

Interface	Attribute	Value
mallinfo()	Thread safety	MT-Unsafe init const:mallopt

mallinfo() would access some global internal objects. If modify them with non-atomically, may get inconsistent results. The identifier *mallopt* in *const:mallopt* mean that **mallopt**() would modify the global internal objects with atomics, that make sure **mallinfo**() is safe enough, others modify with non-atomically maybe not.

CONFORMING TO top

This function is not specified by POSIX or the C standards. A similar function exists on many System V derivatives, and was specified in the SVID.

BUGS top

Information is returned for only the main memory allocation area. Allocations in other arenas are excluded. See malloc_stats(3) and malloc_info(3) for alternatives that include information about other arenas.

The fields of the *mallinfo* structure are typed as *int*. However, because some internal bookkeeping values may be of type *long*, the reported values may wrap around zero and thus be inaccurate.

EXAMPLE top

The program below employs **mallinfo**() to retrieve memory allocation statistics before and after allocating and freeing some blocks of memory. The statistics are displayed on standard output.

The first two command-line arguments specify the number and size of blocks to be allocated with malloc(3).

The remaining three arguments specify which of the allocated blocks should be freed with free(3). These three arguments are optional, and specify (in order): the step size to be used in the loop that frees blocks (the default is 1, meaning free all blocks in the range); the ordinal position of the first block to be freed (default 0, meaning the first allocated block); and a number one greater than the ordinal position of the last block to be freed (default is one greater than the maximum block number). If these three arguments are omitted, then the defaults cause all allocated blocks to be freed.

In the following example run of the program, 1000 allocations of 100 bytes are performed, and then every second allocated block is freed:

```
$ ./a.out 1000 100 2
============== Before allocating blocks ==============
Total non-mmapped bytes (arena):       0
# of free chunks (ordblks):            1
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           0
Bytes in mapped regions (hblkhd):      0
Max. total allocated space (usmblks):  0
Free bytes held in fastbins (fsmblks): 0
Total allocated space (uordblks):      0
Total free space (fordblks):           0
Topmost releasable block (keepcost):   0

============== After allocating blocks ==============
Total non-mmapped bytes (arena):       135168
# of free chunks (ordblks):            1
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           0
```

```
    Bytes in mapped regions (hblkhd):      0
    Max. total allocated space (usmblks):  0
    Free bytes held in fastbins (fsmblks): 0
    Total allocated space (uordblks):      104000
    Total free space (fordblks):           31168
    Topmost releasable block (keepcost):   31168

    ============== After freeing blocks ==============
    Total non-mmapped bytes (arena):       135168
    # of free chunks (ordblks):            501
    # of free fastbin blocks (smblks):     0
    # of mapped regions (hblks):           0
    Bytes in mapped regions (hblkhd):      0
    Max. total allocated space (usmblks):  0
    Free bytes held in fastbins (fsmblks): 0
    Total allocated space (uordblks):      52000
    Total free space (fordblks):           83168
    Topmost releasable block (keepcost):   31168
```

Program source

```
#include <malloc.h>
#include "tlpi_hdr.h"

static void
display_mallinfo(void)
{
    struct mallinfo mi;

    mi = mallinfo();

    printf("Total non-mmapped bytes (arena):       %d\n", mi.arena);
    printf("# of free chunks (ordblks):            %d\n", mi.ordblks);
    printf("# of free fastbin blocks (smblks):     %d\n", mi.smblks);
    printf("# of mapped regions (hblks):           %d\n", mi.hblks);
    printf("Bytes in mapped regions (hblkhd):      %d\n", mi.hblkhd);
    printf("Max. total allocated space (usmblks):  %d\n", mi.usmblks);
    printf("Free bytes held in fastbins (fsmblks): %d\n", mi.fsmblks);
    printf("Total allocated space (uordblks):      %d\n", mi.uordblks);
    printf("Total free space (fordblks):           %d\n", mi.fordblks);
    printf("Topmost releasable block (keepcost):   %d\n", mi.keepcost);
}

int
main(int argc, char *argv[])
{
#define MAX_ALLOCS 2000000
    char *alloc[MAX_ALLOCS];
    int numBlocks, j, freeBegin, freeEnd, freeStep;
    size_t blockSize;
```

```
        if (argc < 3 || strcmp(argv[1], "--help") == 0)
            usageErr("%s num-blocks block-size [free-step [start-free "
                    "[end-free]]]\n", argv[0]);

        numBlocks = atoi(argv[1]);
        blockSize = atoi(argv[2]);
        freeStep = (argc > 3) ? atoi(argv[3]) : 1;
        freeBegin = (argc > 4) ? atoi(argv[4]) : 0;
        freeEnd = (argc > 5) ? atoi(argv[5]) : numBlocks;

        printf("============== Before allocating blocks ==============\n");
        display_mallinfo();

        for (j = 0; j < numBlocks; j++) {
            if (numBlocks >= MAX_ALLOCS)
                fatal("Too many allocations");

            alloc[j] = malloc(blockSize);
            if (alloc[j] == NULL)
                errExit("malloc");
        }

        printf("\n============== After allocating blocks ==============\n");
        display_mallinfo();

        for (j = freeBegin; j < freeEnd; j += freeStep)
            free(alloc[j]);

        printf("\n============== After freeing blocks ==============\n");
        display_mallinfo();

        exit(EXIT_SUCCESS);
    }
```

SEE ALSO top

mmap(2), malloc(3), malloc_info(3), malloc_stats(3), malloc_trim(3), mallopt(3)

COLOPHON top

This page is part of release 4.13 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.

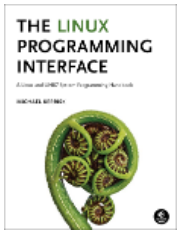
Linux **2017-09-15** **MALLINFO(3)**

Pages that refer to this page: malloc_hook(3), malloc_info(3), malloc_stats(3), mallopt(3)

HTML rendering created 2017-09-15 by Michael Kerrisk, author of *The Linux Programming Interface*, maintainer of the Linux *man-pages* project.

For details of in-depth **Linux/UNIX system programming training courses** that I teach, look here.

Hosting by jambit GmbH.



NAME | SYNOPSIS | DESCRIPTION | ATTRIBUTES | CONFORMING TO | BUGS | EXAMPLE | SEE ALSO | COLOPHON

MALLINFO(3) Linux Programmer's Manual MALLINFO(3)

NAME top

mallinfo - obtain memory allocation information

SYNOPSIS top

```
#include <malloc.h>

struct mallinfo mallinfo(void);
```

DESCRIPTION top

The **mallinfo**() function returns a copy of a structure containing information about memory allocations performed by malloc(3) and related functions. This structure is defined as follows:

```
struct mallinfo {
    int arena;     /* Non-mmapped space allocated (bytes) */
    int ordblks;   /* Number of free chunks */
    int smblks;    /* Number of free fastbin blocks */
    int hblks;     /* Number of mmapped regions */
    int hblkhd;    /* Space allocated in mmapped regions (bytes) */
    int usmblks;   /* Maximum total allocated space (bytes) */
    int fsmblks;   /* Space in freed fastbin blocks (bytes) */
    int uordblks;  /* Total allocated space (bytes) */
    int fordblks;  /* Total free space (bytes) */
    int keepcost;  /* Top-most, releasable space (bytes) */
};
```

The fields of the *mallinfo* structure contain the following information:

arena The total amount of memory allocated by means other than mmap(2) (i.e., memory allocated on the heap). This figure includes both in-use blocks and blocks on the free list.

ordblks The number of ordinary (i.e., non-fastbin) free blocks.

smblks The number of fastbin free blocks (see mallopt(3)).

hblks The number of blocks currently allocated using mmap(2). (See the discussion of **M_MMAP_THRESHOLD** in mallopt(3).)

hblkhd The number of bytes in blocks currently allocated using mmap(2).

usmblks The "highwater mark" for allocated space—that is, the maximum amount of space that was ever allocated. This field is maintained only in nonthreading environments.

fsmblks The total number of bytes in fastbin free blocks.

uordblks The total number of bytes used by in-use allocations.

fordblks The total number of bytes in free blocks.

keepcost The total amount of releasable free space at the top of the heap. This is the maximum number of bytes that could ideally (i.e., ignoring page alignment restrictions, and so on) be released by malloc_trim(3).

ATTRIBUTES top

For an explanation of the terms used in this section, see attributes(7).

Interface	Attribute	Value
mallinfo()	Thread safety	MT-Unsafe init const:mallopt

mallinfo() would access some global internal objects. If modify them with non-atomically, may get inconsistent results. The identifier *mallopt* in *const:mallopt* mean that **mallopt**() would modify the global internal objects with atomics, that make sure **mallinfo**() is safe enough, others modify with non-atomically maybe not.

CONFORMING TO top

This function is not specified by POSIX or the C standards. A similar function exists on many System V derivatives, and was specified in the SVID.

BUGS top

Information is returned for only the main memory allocation area. Allocations in other arenas are excluded. See malloc_stats(3) and malloc_info(3) for alternatives that include information about other arenas.

The fields of the *mallinfo* structure are typed as *int*. However, because some internal bookkeeping values may be of type *long*, the reported values may wrap around zero and thus be inaccurate.

EXAMPLE top

The program below employs **mallinfo**() to retrieve memory allocation statistics before and after allocating and freeing some blocks of memory. The statistics are displayed on standard output.

The first two command-line arguments specify the number and size of blocks to be allocated with malloc(3).

The remaining three arguments specify which of the allocated blocks should be freed with free(3). These three arguments are optional, and specify (in order): the step size to be used in the loop that frees blocks (the default is 1, meaning free all blocks in the range); the ordinal position of the first block to be freed (default 0, meaning the first allocated block); and a number one greater than the ordinal position of the last block to be freed (default is one greater than the maximum block number). If these three arguments are omitted, then the defaults cause all allocated blocks to be freed.

In the following example run of the program, 1000 allocations of 100 bytes are performed, and then every second allocated block is freed:

```
$ ./a.out 1000 100 2
============== Before allocating blocks ==============
Total non-mmapped bytes (arena):       0
# of free chunks (ordblks):            1
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           0
Bytes in mapped regions (hblkhd):      0
Max. total allocated space (usmblks):  0
Free bytes held in fastbins (fsmblks): 0
Total allocated space (uordblks):      0
Total free space (fordblks):           0
Topmost releasable block (keepcost):   0

============== After allocating blocks ==============
Total non-mmapped bytes (arena):       135168
# of free chunks (ordblks):            1
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           0
```

```
    Bytes in mapped regions (hblkhd):      0
    Max. total allocated space (usmblks):  0
    Free bytes held in fastbins (fsmblks): 0
    Total allocated space (uordblks):      104000
    Total free space (fordblks):           31168
    Topmost releasable block (keepcost):   31168

    ============== After freeing blocks ==============
    Total non-mmapped bytes (arena):       135168
    # of free chunks (ordblks):            501
    # of free fastbin blocks (smblks):     0
    # of mapped regions (hblks):           0
    Bytes in mapped regions (hblkhd):      0
    Max. total allocated space (usmblks):  0
    Free bytes held in fastbins (fsmblks): 0
    Total allocated space (uordblks):      52000
    Total free space (fordblks):           83168
    Topmost releasable block (keepcost):   31168
```

Program source

```
#include <malloc.h>
#include "tlpi_hdr.h"

static void
display_mallinfo(void)
{
    struct mallinfo mi;

    mi = mallinfo();

    printf("Total non-mmapped bytes (arena):       %d\n", mi.arena);
    printf("# of free chunks (ordblks):            %d\n", mi.ordblks);
    printf("# of free fastbin blocks (smblks):     %d\n", mi.smblks);
    printf("# of mapped regions (hblks):           %d\n", mi.hblks);
    printf("Bytes in mapped regions (hblkhd):      %d\n", mi.hblkhd);
    printf("Max. total allocated space (usmblks):  %d\n", mi.usmblks);
    printf("Free bytes held in fastbins (fsmblks): %d\n", mi.fsmblks);
    printf("Total allocated space (uordblks):      %d\n", mi.uordblks);
    printf("Total free space (fordblks):           %d\n", mi.fordblks);
    printf("Topmost releasable block (keepcost):   %d\n", mi.keepcost);
}

int
main(int argc, char *argv[])
{
#define MAX_ALLOCS 2000000
    char *alloc[MAX_ALLOCS];
    int numBlocks, j, freeBegin, freeEnd, freeStep;
    size_t blockSize;
```

```
        if (argc < 3 || strcmp(argv[1], "--help") == 0)
            usageErr("%s num-blocks block-size [free-step [start-free "
                    "[end-free]]]\n", argv[0]);

        numBlocks = atoi(argv[1]);
        blockSize = atoi(argv[2]);
        freeStep = (argc > 3) ? atoi(argv[3]) : 1;
        freeBegin = (argc > 4) ? atoi(argv[4]) : 0;
        freeEnd = (argc > 5) ? atoi(argv[5]) : numBlocks;

        printf("============== Before allocating blocks ==============\n");
        display_mallinfo();

        for (j = 0; j < numBlocks; j++) {
            if (numBlocks >= MAX_ALLOCS)
                fatal("Too many allocations");

            alloc[j] = malloc(blockSize);
            if (alloc[j] == NULL)
                errExit("malloc");
        }

        printf("\n============== After allocating blocks ==============\n");
        display_mallinfo();

        for (j = freeBegin; j < freeEnd; j += freeStep)
            free(alloc[j]);

        printf("\n============== After freeing blocks ==============\n");
        display_mallinfo();

        exit(EXIT_SUCCESS);
    }
```

SEE ALSO top

mmap(2), malloc(3), malloc_info(3), malloc_stats(3), malloc_trim(3), mallopt(3)

COLOPHON top

This page is part of release 4.13 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.

Linux **2017-09-15** **MALLINFO(3)**

Pages that refer to this page: malloc_hook(3), malloc_info(3), malloc_stats(3), mallopt(3)

HTML rendering created 2017-09-15 by Michael Kerrisk, author of *The Linux Programming Interface*, maintainer of the Linux *man-pages* project.

For details of in-depth **Linux/UNIX system programming training courses** that I teach, look here.

Hosting by jambit GmbH.