OS 2020

Bechmarking your computer black box

魏連興 Gary Wei

p76094224@gs.ncku.edu.tw

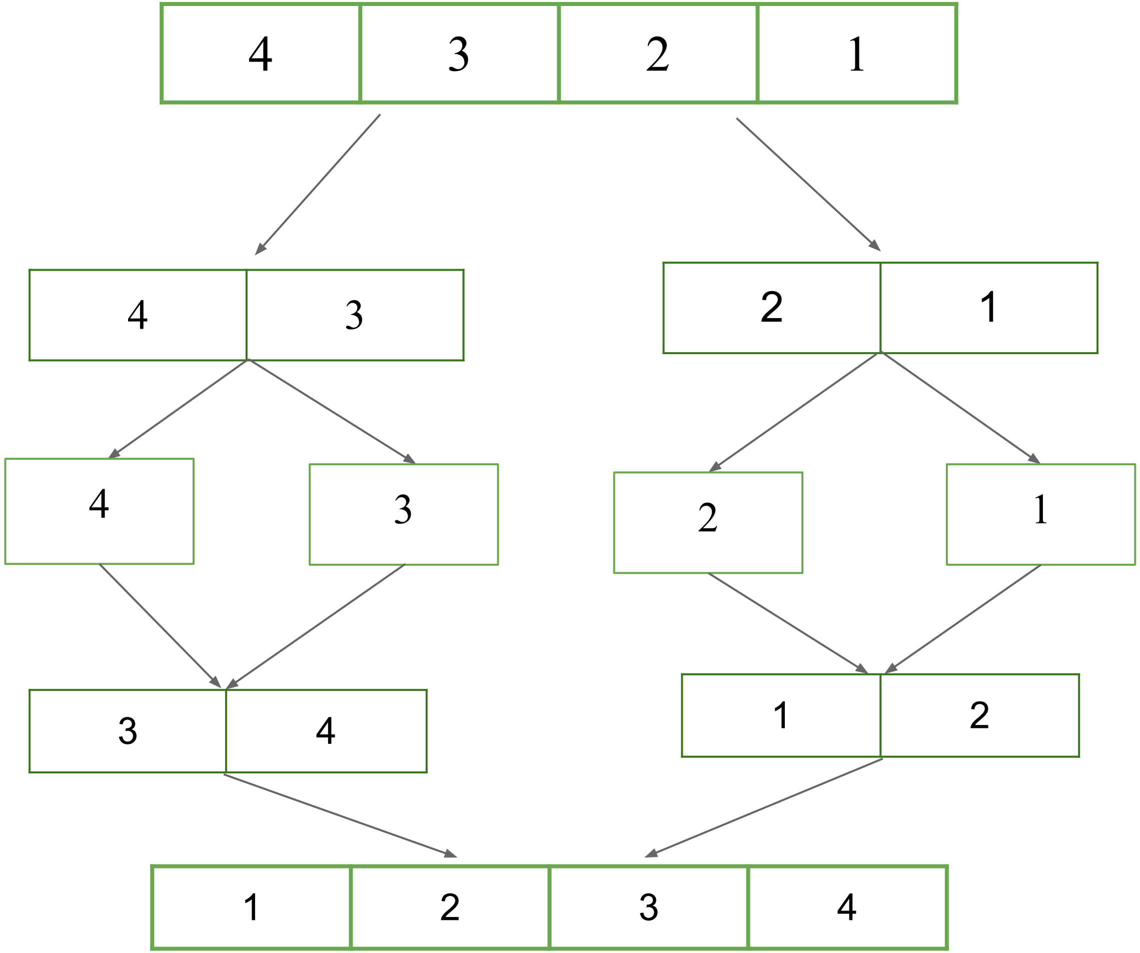
Objectives

- Be a good user.
- Know some detail with memory and caches(L1 caches, L2 ...).
- CPU utilization.
- RAM access rule of Thumb.
- Performance

Outline

- Merge Sort Algorithm
- Step-by-step tutorial
  - Linux Performance Event (perf)
  - Case Study
    - Matrix Multiplication
    - Computer Multitasking
- Homework Rules

Merge Sort



4	3	2	1

4	3

2	1

4

3

2

1

3	4

1	2

1	2	3	4

Divide-and-Conquer 技巧

- 分割(Divide)一個較大問題實例成為一個或多個較小的實例
- 解出每個較小實例的答案(Conquer)
- 除非實例已經分割到足夠小的地步，否則使用遞迴來解
- 必要的話，將兩個較小實體的解合併(Combine)以獲得原始問題實例的解

Merge list too long

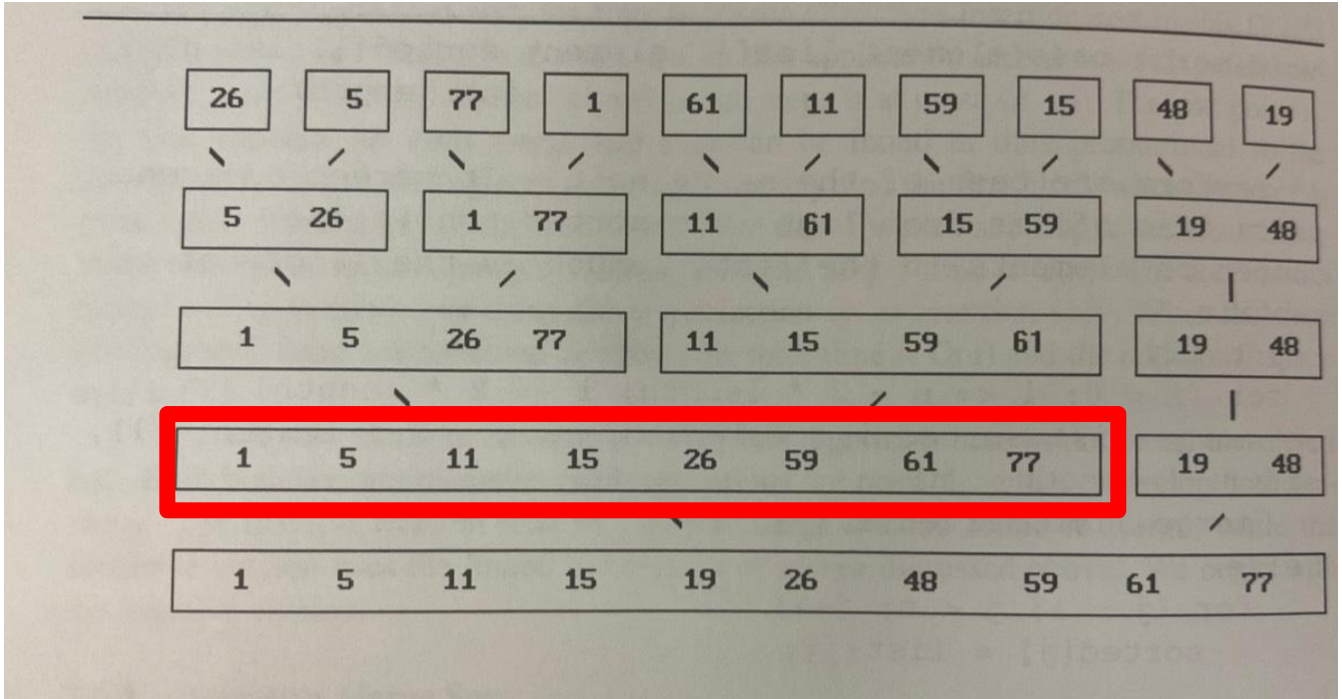
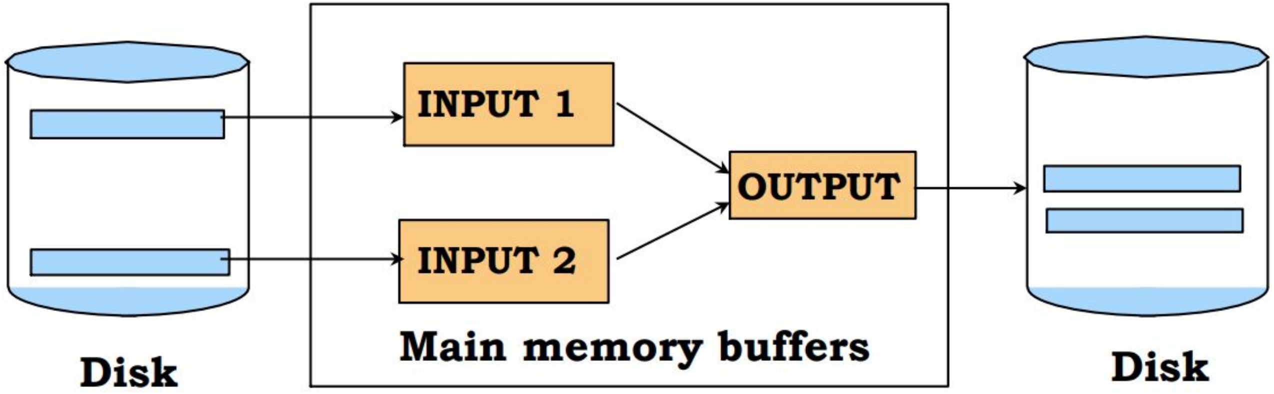


Figure 7.7: Merge tree for iterative merge sort

2-Way Merge Sort: Requires 3 Buffers

- Pass 1: Read a page, sort it, write it.
  - only one buffer page is used
- Pass 2, 3, …, N etc.:
  - Read two pages, merge them, and write merged page
  - Requires three buffer pages.



Two-Way External Merge Sort

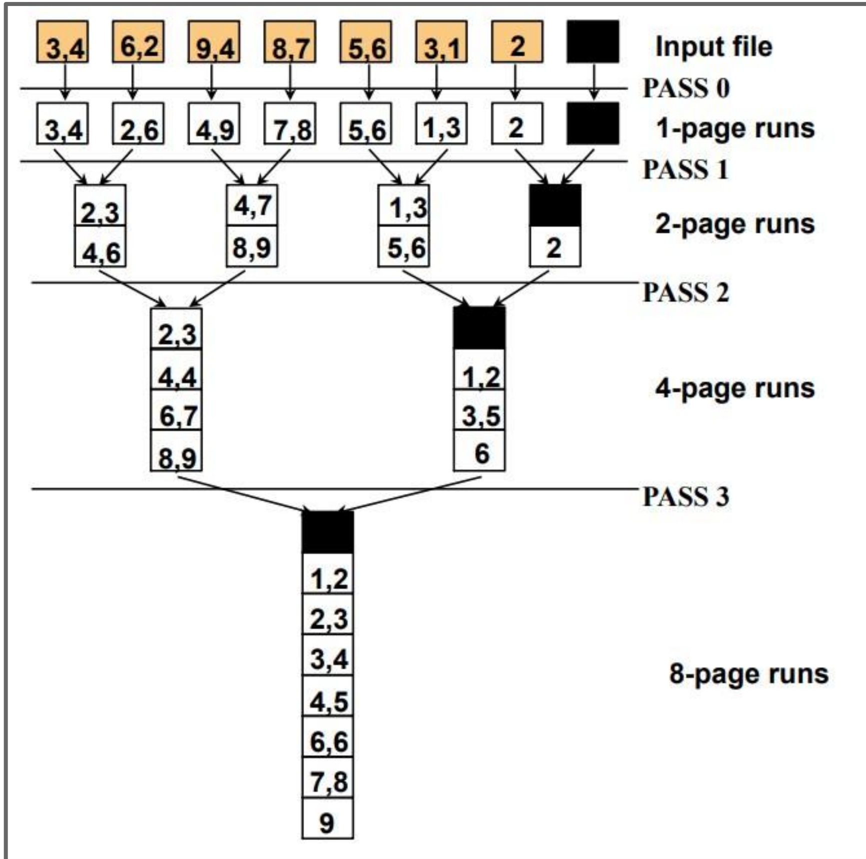
- Each pass we read + write each page in file.
- *N* pages in the file => the number of passes

$$= \lceil \log_2 N \rceil + 1$$

- So toal cost is:

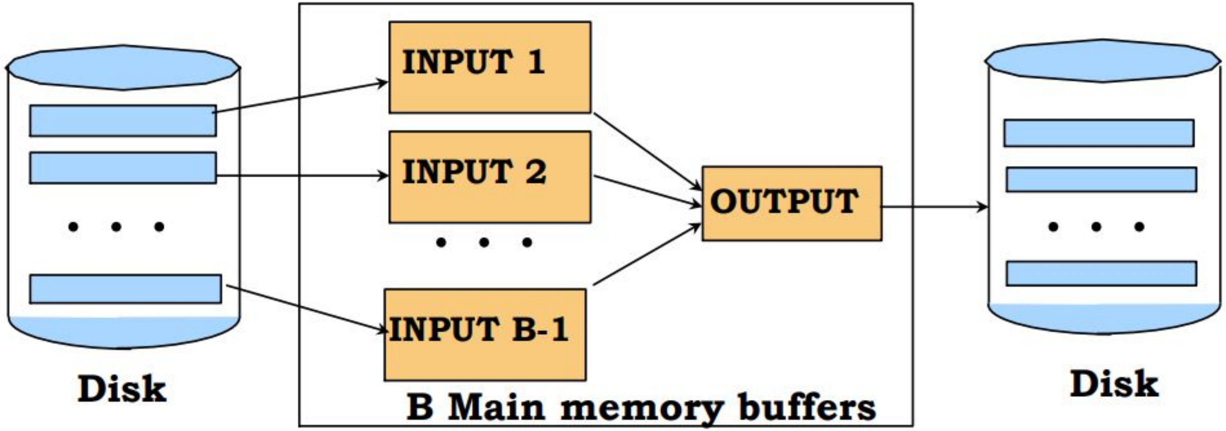
$$2N\left(\lceil \log_2 N \rceil + 1\right)$$

- Idea:
  - Divide and conquer:
    - sort pages and merge



General External Merge Sort

- More than 3 buffer pages. How can we utilize them?
- To sort a file with *N* pages using *B* buffer pages (including output):
    - Pass 0: use *B* buffer pages. Produce sorted runs of *B* pages each.
    - Pass 2, …, etc.: merge *B*-1 runs



General External Merge Sort

- More than 3 buffer pages. How can we utilize them?
- Key Insight #1: We can merge more than 2 input buffers at a time… affects fanout base of log!
- Key Insight #2: The output buffer is generated incrementally, so only one buffer page is needed for any size of run!
- To sort a file with ***N*** pages using ***B*** buffer pages:
  - Pass 0: use *B* buffer pages. Produce sorted runs of *B* pages each.
  - Pass 2, …, etc.: merge ***B-1*** runs, leaving one page for output.

General External Merge Sort : Pass 0

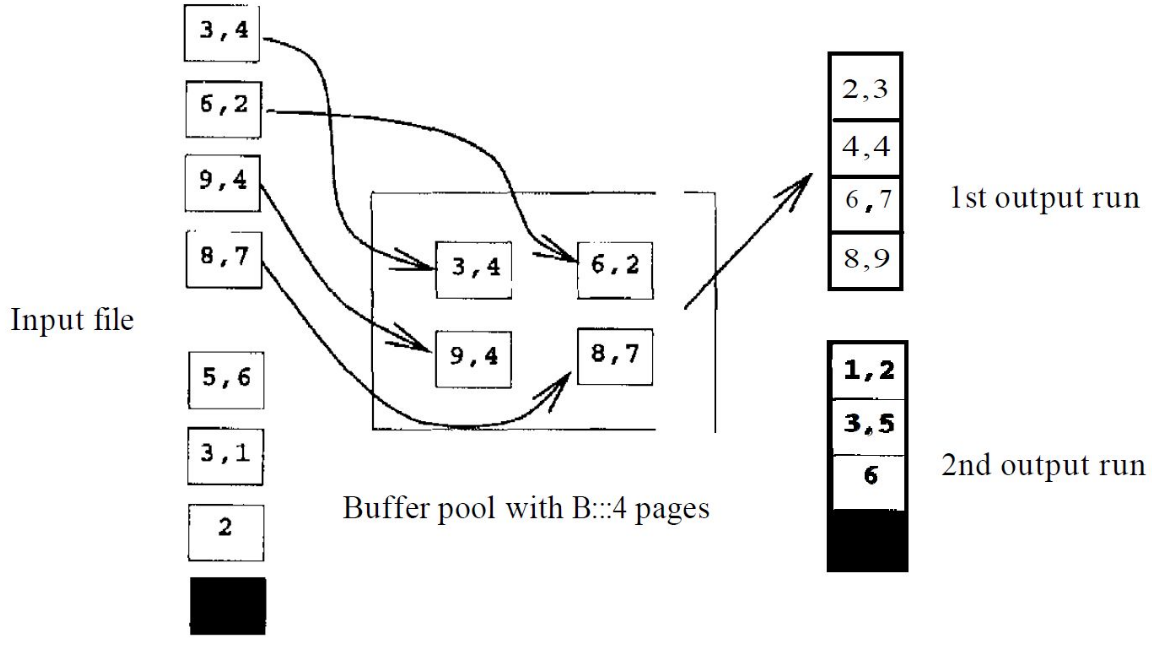


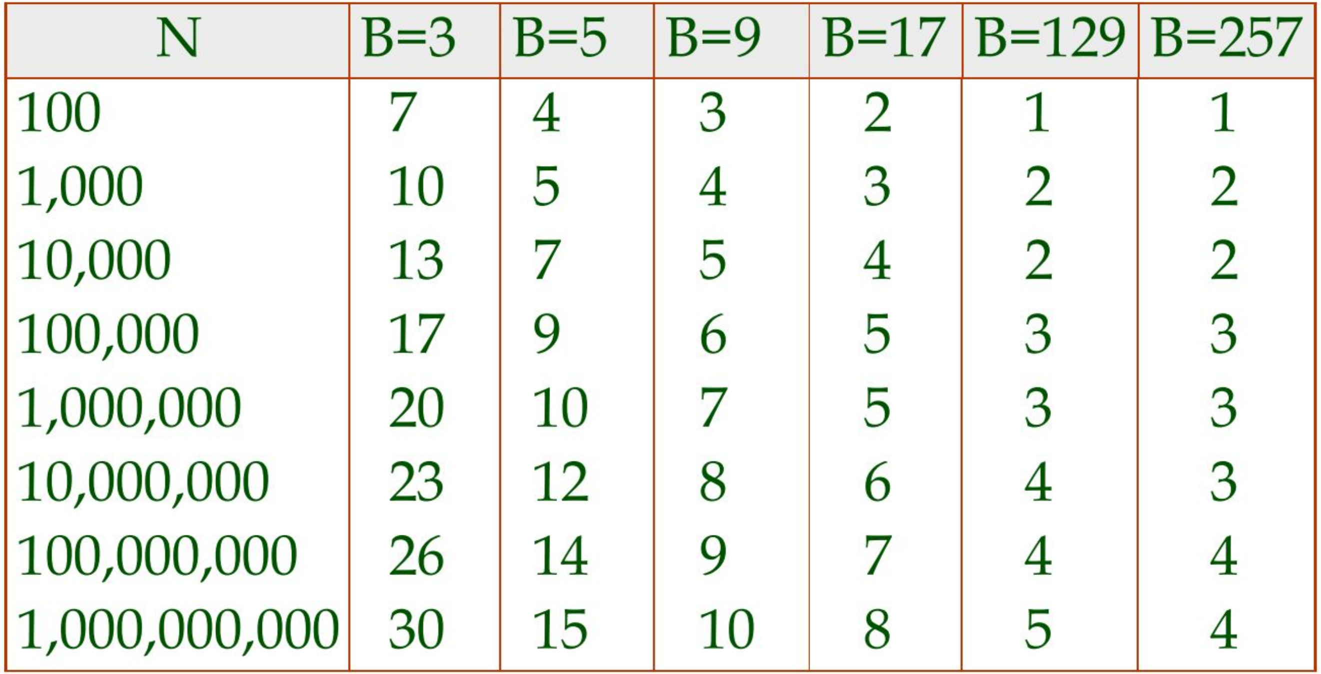
Figure 13.4 External Merge Sort with *B* Buffer Pages: Pass 0

Cost of External Merge Sort

- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$

- Cost = 2N * (# of passes)
- E.g., with 5 buffer pages, to sort 108 page file:
  - Pass 0: ceil(108/5)= 22 sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: Do four-way merge
    - ceil(22/4) = 6 sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: ceil(6/4)= 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

Number of Passes of External Sort



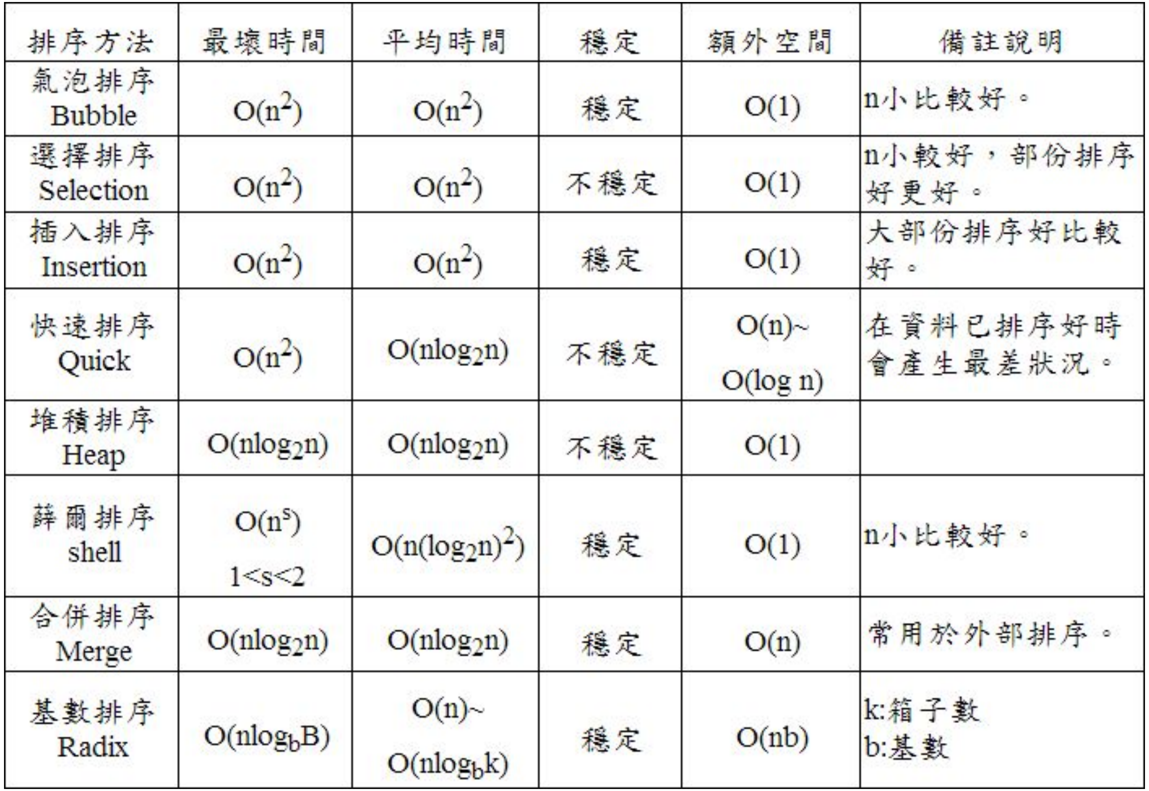
N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

Internal Sort Algorithm

- n個正整數的排列共有n!種順序
  - 以1, 2, 3為例

    [1, 2, 3]　[1, 3, 2]　[2, 1, 3]　[2, 3, 1]　[3, 1, 2]　[3, 2, 1]
- 倒置(inversion)
  - (k_i, k_j) 使得 $i < j$ 且 $k_i > k_j$
- 一組排列不含有倒置 若且唯若 該排列已按照順序排好
  - 對n個相異的key進行排序 ↔ 把排列中的倒置移除

Complexity



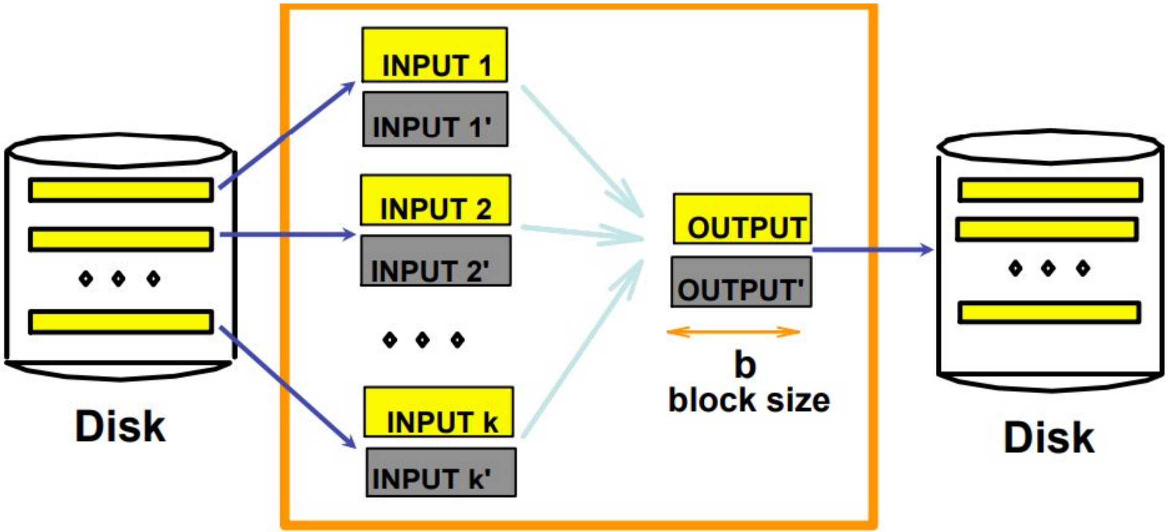
排序方法	最壞時間	平均時間	穩定	額外空間	備註說明
氣泡排序 Bubble	$O(n^2)$	$O(n^2)$	穩定	$O(1)$	n小比較好。
選擇排序 Selection	$O(n^2)$	$O(n^2)$	不穩定	$O(1)$	n小較好，部份排序好更好。
插入排序 Insertion	$O(n^2)$	$O(n^2)$	穩定	$O(1)$	大部份排序好比較好。
快速排序 Quick	$O(n^2)$	$O(n\log_2 n)$	不穩定	$O(n)$~ $O(\log n)$	在資料已排序好時會產生最差狀況。
堆積排序 Heap	$O(n\log_2 n)$	$O(n\log_2 n)$	不穩定	$O(1)$	
薛爾排序 shell	$O(n^s)$ $1<s<2$	$O(n(\log_2 n)^2)$	穩定	$O(1)$	n小比較好。
合併排序 Merge	$O(n\log_2 n)$	$O(n\log_2 n)$	穩定	$O(n)$	常用於外部排序。
基數排序 Radix	$O(n\log_b B)$	$O(n)$~ $O(n\log_b k)$	穩定	$O(nb)$	k:箱子數 b:基數

I/O for External Merge Sort

- Actually, do I/O a page at a time
- In fact, read a block of pages sequentially!
- Suggests we should make each buffer (input/output) be a block of pages.
  - But this will reduce fan-out during merge passes!
  - In practice, most files still sorted in 2-3 passes.

Double Buffering

- To reduce wait time for I/O request to complete, can prefetch into a ”shadow block”.
- Potentially, more passes; in practice, most files still sorted in 2-3 passes.



B main memory buffers, k-way merge

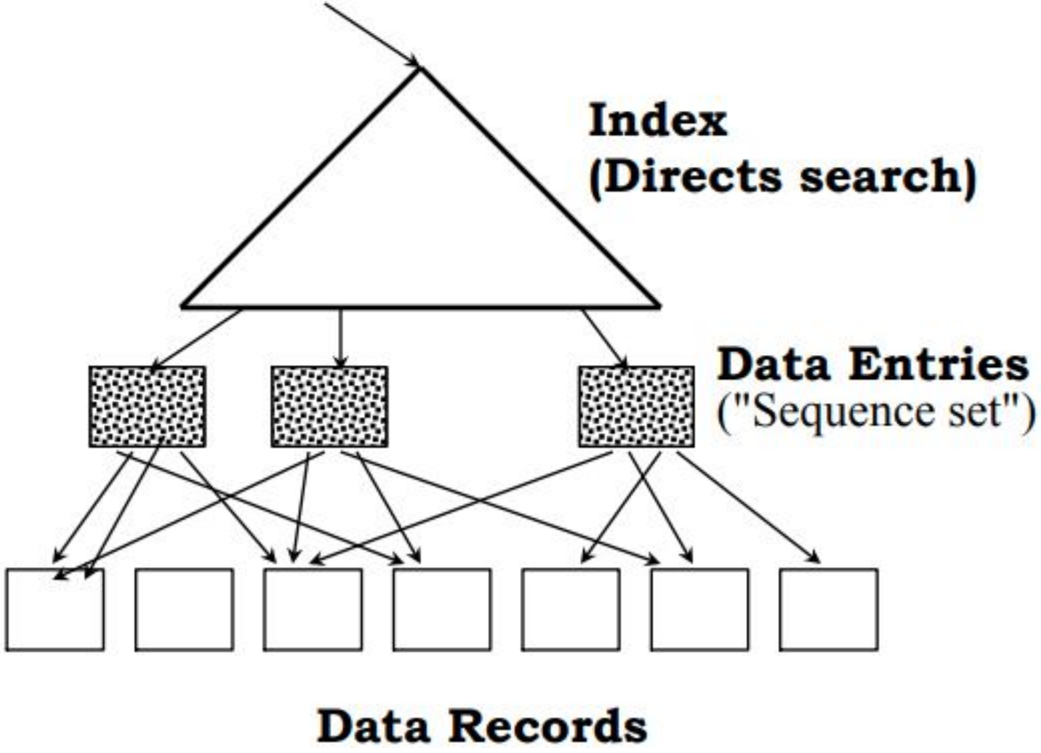
Sorting Records!

- Sorting has become a blood sport!
  - Parallel external sorting is the name of the game ...
- **Sort Benchmark Home Page**
    - created by computer scientist Jim Gray
    - http://sortbenchmark.org/
- New **benchmarks** proposed:
  - Minute Sort: How many can you sort in 1 minute?
  - Dollar Sort: How many can you sort for $1.00?

Using B+ Trees for Sorting

- Scenario: Table to be sorted has B+ tree index on sorting column(s).
- Idea: Can retrieve records in order by traversing leaf pages.
- Is this a good idea?
- Cases to consider:
    - B+ tree is clustered **Good idea**!
    - B+ tree is not clustered Could be a **very bad idea**!

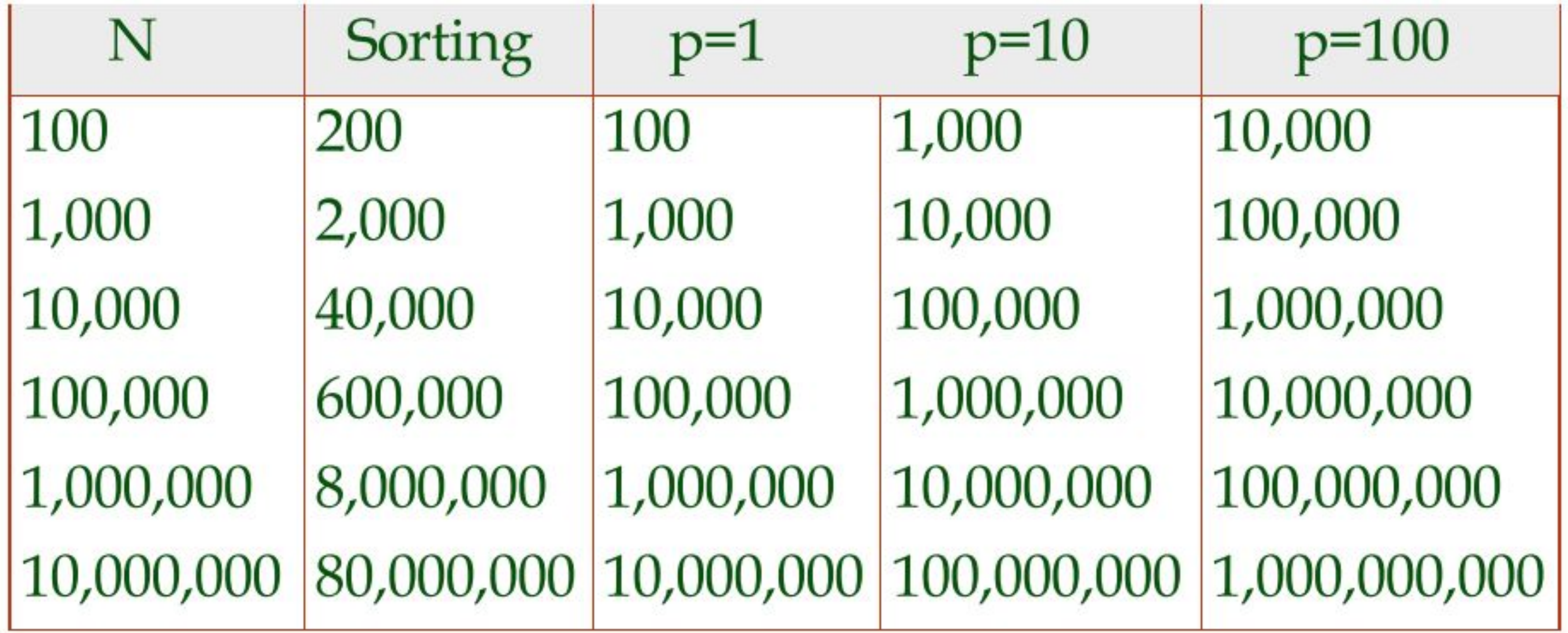
Clustered B+ Tree Used for Sorting (1/2)



Clustered B+ Tree Used for Sorting (2/2)

- Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)
- If Alternative 2 is used? Additional cost of retrieving data records: each page fetched just once.
- Fill factor of < 100% introduces a small overhead extra pages fetched
- Always better than external sorting!
- Alternative (2) for data entries; each data entry contains *rid* of a data record. In general, one I/O per data record!

External Sorting vs. Unclustered Index



N	Sorting	p=1	p=10	p=100
100	200	100	1,000	10,000
1,000	2,000	1,000	10,000	100,000
10,000	40,000	10,000	100,000	1,000,000
100,000	600,000	100,000	1,000,000	10,000,000
1,000,000	8,000,000	1,000,000	10,000,000	100,000,000
10,000,000	80,000,000	10,000,000	100,000,000	1,000,000,000

p: # of records per page
B=1,000 and block size=32 for sorting
p=100 is the more realistic value.

Summary (1/2)

- External sorting is important; DBMS may dedicate part of buffer pool just for sorting!
- External merge sort minimizes disk I/O cost:
    - Pass 0: Produces sorted runs of size *B* (# buffer pages). Later passes: merge runs.
    - # of runs merged at a time depends on *B*, and block size.
    - Larger block size means less I/O cost per page.
    - Larger block size means smaller # runs merged.
    - In practice, # of runs rarely more than 2 or 3.
- Choice of internal sort algorithm may matter:
    - Quicksort: Quick!
    - Replacement sort: slower (2x), but with longer runs

Summary (2/2)

- The best sorts are wildly fast:
  - Despite 40+ years of research, we’re still improving!
- Clustered B+ tree is good for sorting; unclustered tree is usually very bad.
- Extracurricular reading materia:
  - Timsort
    - It was implemented by Tim Peters in 2002 for use in the Python programming language.
    - Techniques from McIlroy, Peter (January 1993). "Optimistic Sorting and Information Theoretic Complexity". Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 467–474. ISBN 0-89871-313-7.

Step-by-step tutorial

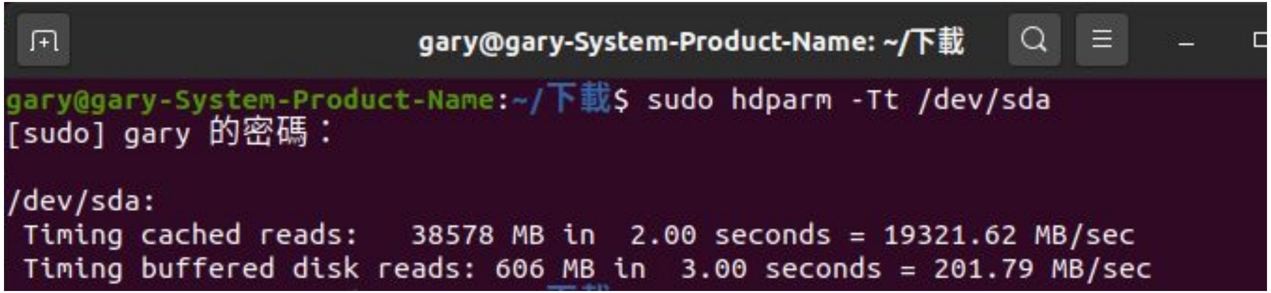
- Know the speed limit of your black box.
- Linux Performance Event (perf)
- Computer multitasking
- 4K對齊

Computer Architecture

- Computing unit
  - Central processing unit(CPU): a few instructions committed per nanoseconds (10^-9)
  - Graphics processing unit (GPU): up-to hundreds evens thousands per ns
- Storage Unit
  - Cache: per cache block per ns
  - Random access memory(RAM): per memory block per 10x ns
  - Disk drive : mesured in bandwidth, roughly ~100MBytes/sec
  - Solid-state drive (SSD): 200~500 MBytes/sec
- Communication Unit
  - Network interface card (NIC) : ~100MBytes/sec for 1GBbits/sec Ethernet.
  - Switch: can upto 10Gbits/sec in parallel for connections

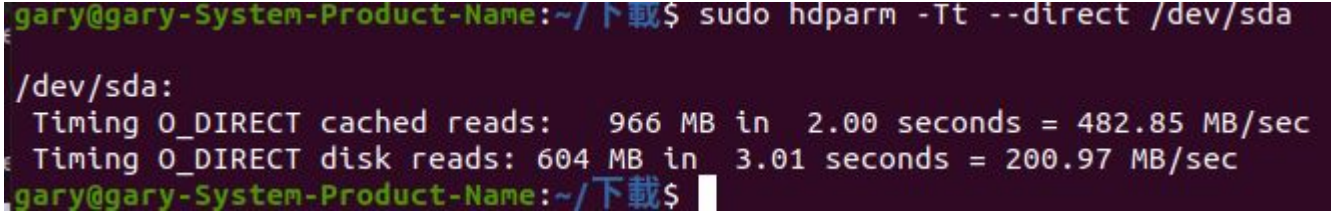
Know the speed limit of your black box. (1/3)

```
$ sudo hdparm -t /dev/sda  #每秒可以從Disk Read多少資料
```



```
gary@gary-System-Product-Name:~/下載$ sudo hdparm -Tt /dev/sda
[sudo] gary 的密碼：

/dev/sda:
 Timing cached reads:   38578 MB in  2.00 seconds = 19321.62 MB/sec
 Timing buffered disk reads: 606 MB in  3.00 seconds = 201.79 MB/sec
```

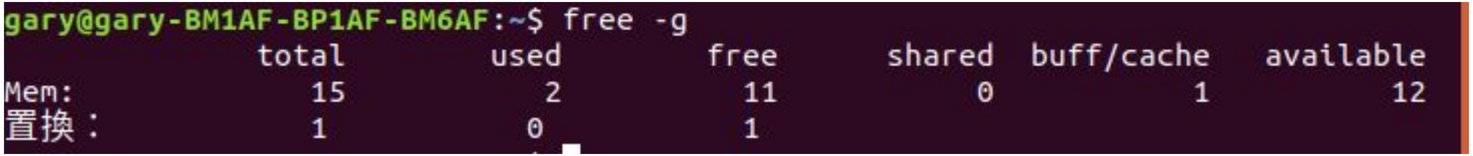
```
# hdparm -t --direct /dev/sda
```



```
gary@gary-System-Product-Name:~/下載$ sudo hdparm -Tt --direct /dev/sda

/dev/sda:
 Timing O_DIRECT cached reads:   966 MB in  2.00 seconds = 482.85 MB/sec
 Timing O_DIRECT disk reads: 604 MB in  3.01 seconds = 200.97 MB/sec
gary@gary-System-Product-Name:~/下載$
```

Know the speed limit of your black box. (2/3)

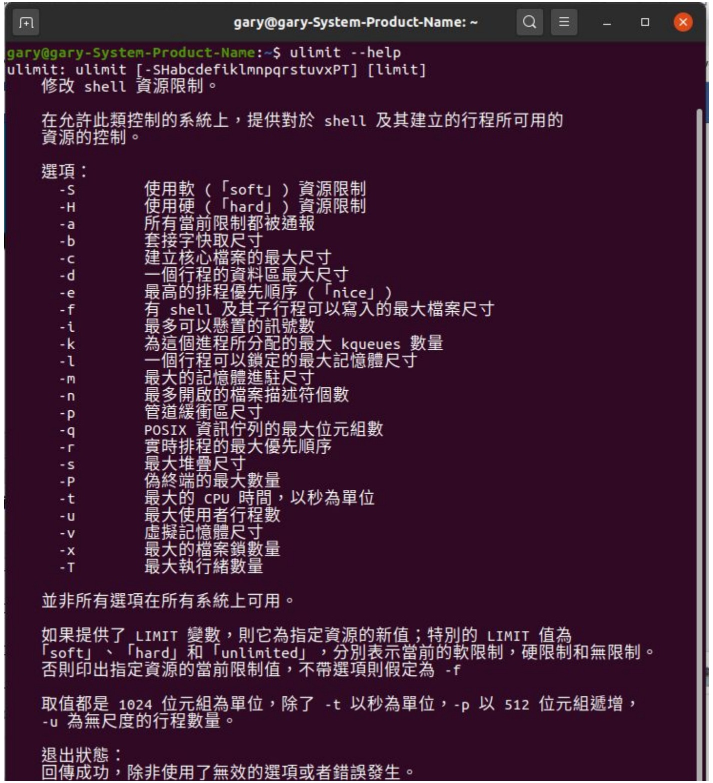
$ free -g #以 GB作為單位顯示記憶體使用狀況 (-m 是MB)



```
gary@gary-BM1AF-BP1AF-BM6AF:~$ free -g
              total        used        free      shared  buff/cache   available
Mem:             15           2          11           0           1          12
置換：            1           0           1
```

Know the speed limit of your black box. (3/3)

$ ulimit --help



```
gary@gary-System-Product-Name:~$ ulimit --help
ulimit: ulimit [-SHabcdefiklmnpqrstuvxPT] [limit]
    修改 shell 資源限制。

    在允許此類控制的系統上，提供對於 shell 及其建立的行程所可用的
    資源的控制。

    選項：
      -S        使用軟（「soft」）資源限制
      -H        使用硬（「hard」）資源限制
      -a        所有當前限制都被通報
      -b        套接字快取尺寸
      -c        建立核心檔案的最大尺寸
      -d        一個行程的資料區最大尺寸
      -e        最高的排程優先順序（「nice」）
      -f        有 shell 及其子行程可以寫入的最大檔案尺寸
      -i        最多可以懸置的訊號數
      -k        為這個進程所分配的最大 kqueues 數量
      -l        一個行程可以鎖定的最大記憶體尺寸
      -m        最大的記憶體進駐尺寸
      -n        最多開啟的檔案描述符個數
      -p        管道緩衝區尺寸
      -q        POSIX 資訊佇列的最大位元組數
      -r        實時排程的最大優先順序
      -s        最大堆疊尺寸
      -P        偽終端的最大數量
      -t        最大的 CPU 時間，以秒為單位
      -u        最大使用者行程數
      -v        虛擬記憶體尺寸
      -x        最大的檔案鎖數量
      -T        最大執行緒數量

    並非所有選項在所有系統上可用。

    如果提供了 LIMIT 變數，則它為指定資源的新值；特別的 LIMIT 值為
    「soft」、「hard」和「unlimited」，分別表示當前的軟限制，硬限制和無限制。
    否則印出指定資源的當前限制值，不帶選項則假定為 -f

    取值都是 1024 位元組為單位，除了 -t 以秒為單位，-p 以 512 位元組遞增，
    -u 為無尺度的行程數量。

    退出狀態：
    回傳成功，除非使用了無效的選項或者錯誤發生。
```

Process State

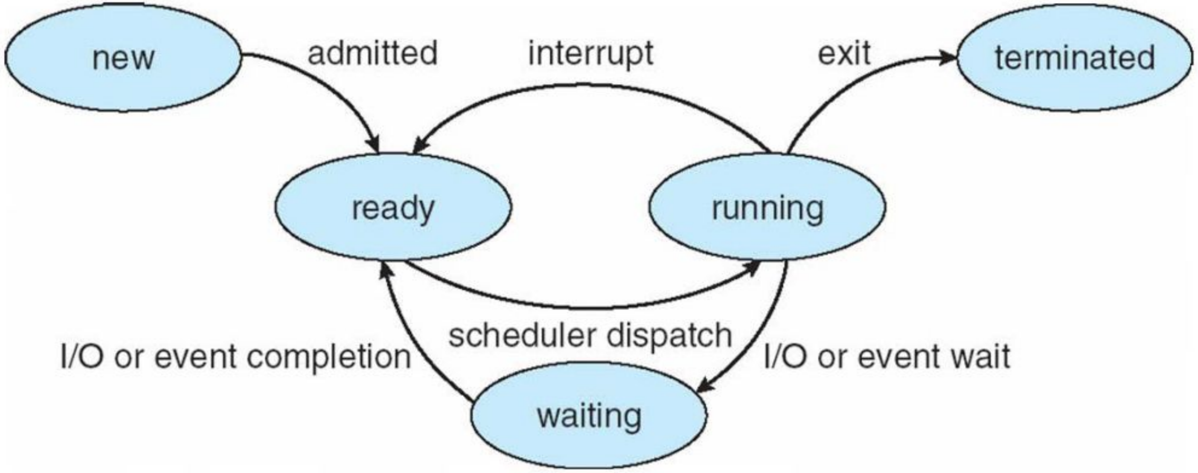
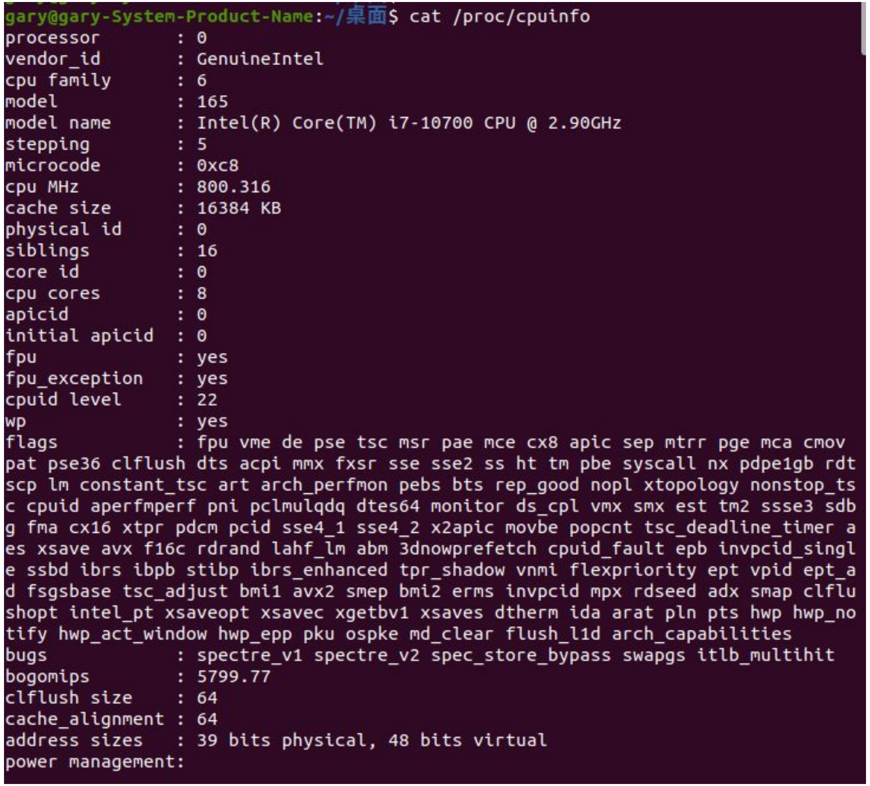


Figure. Diagram of Process State

CPU 資訊

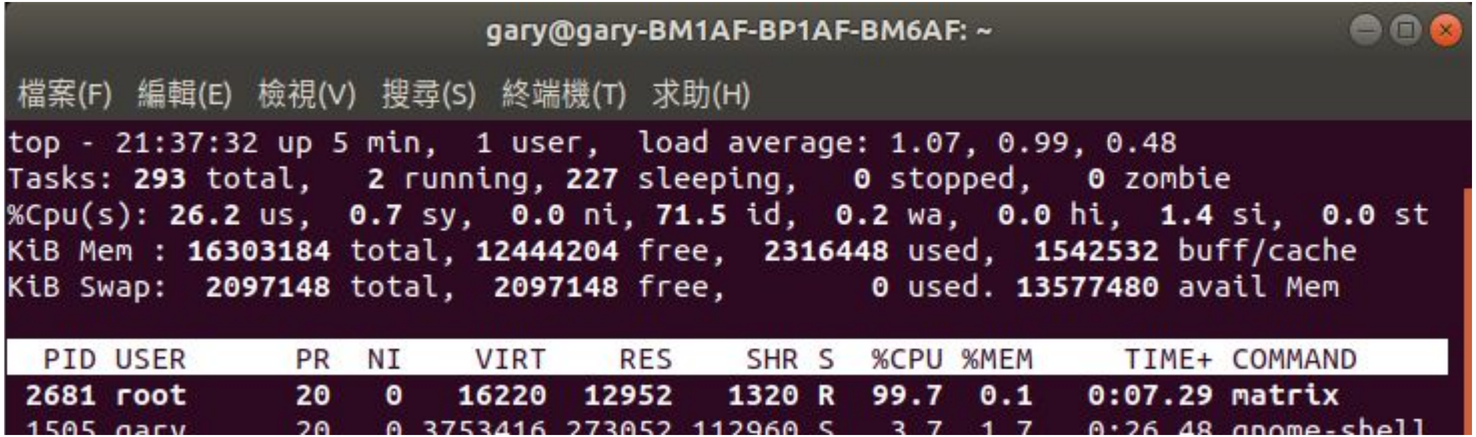
$ cat /proc/cpuinfo



```
gary@gary-System-Product-Name:~/桌面$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz
stepping        : 5
microcode       : 0xc8
cpu MHz         : 800.316
cache size      : 16384 KB
physical id     : 0
siblings        : 16
core id         : 0
cpu cores       : 8
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdt
scp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_ts
c cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdb
g fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer a
es xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb invpcid_singl
e ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid ept_a
d fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflu
shopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_no
tify hwp_act_window hwp_epp pku ospke md_clear flush_l1d arch_capabilities
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit
bogomips        : 5799.77
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

Get the utilization of a process on CPU or Memory.

$ top #查看記憶體&CPU使用量


gary@gary-BM1AF-BP1AF-BM6AF: ~
檔案(F) 編輯(E) 檢視(V) 搜尋(S) 終端機(T) 求助(H)

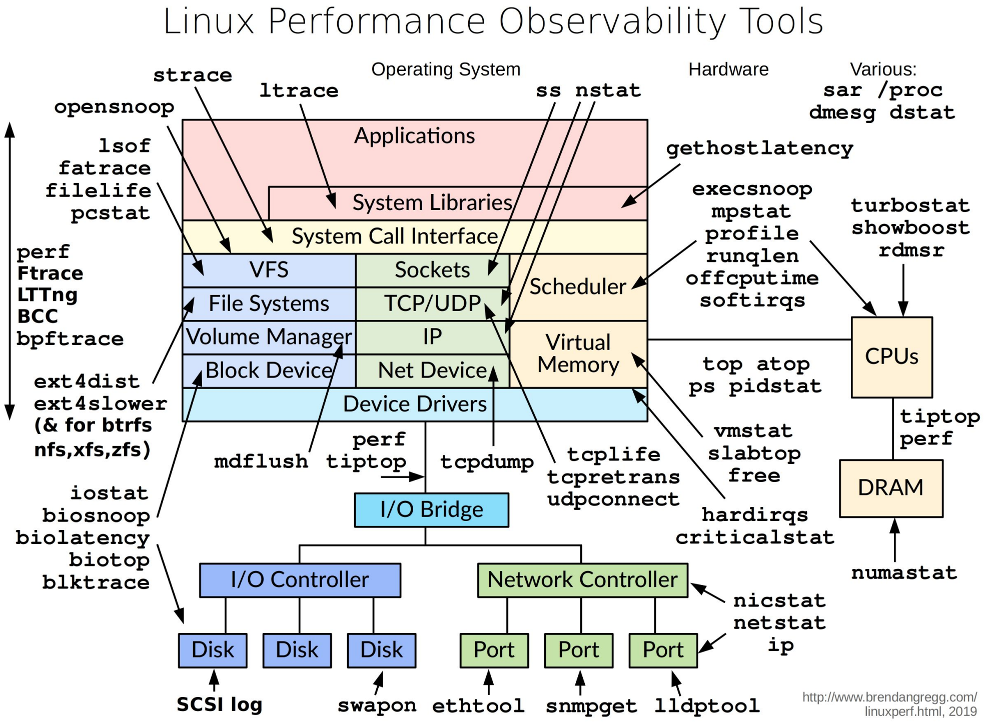
```
top - 21:37:32 up 5 min,  1 user,  load average: 1.07, 0.99, 0.48
Tasks: 293 total,   2 running, 227 sleeping,   0 stopped,   0 zombie
%Cpu(s): 26.2 us,  0.7 sy,  0.0 ni, 71.5 id,  0.2 wa,  0.0 hi,  1.4 si,  0.0 st
KiB Mem : 16303184 total, 12444204 free,  2316448 used,  1542532 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used. 13577480 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2681 root      20   0   16220  12952   1320 R  99.7  0.1   0:07.29 matrix
 1505 gary      20   0 3753416 273052 112960 S   3.7  1.7   0:26.48 gnome-shell
```

Performance Event (perf)

- Linux 2.6.31 以後內建的系統效能分析工具
- 分析 Hardware event
  - cpu-cycles, instructions, cache-misses, branch-misses ......
- 分析 Software event
  - page-faults, context-switches ......
- Tracepoint event

Performance Event

- " perf " tool.



source code: https://github.com/torvalds/linux/tree/master/tools/perf

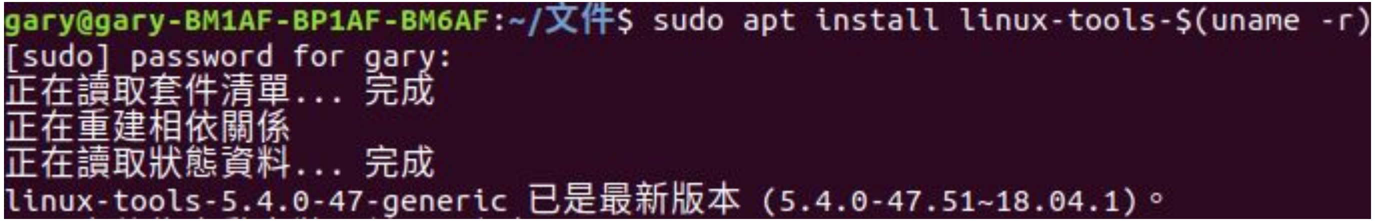
Performance profiler

- Check environment

```
$ cat "/boot/config-`uname -r`" | grep "PERF_EVENT"
```

- Install linux tools

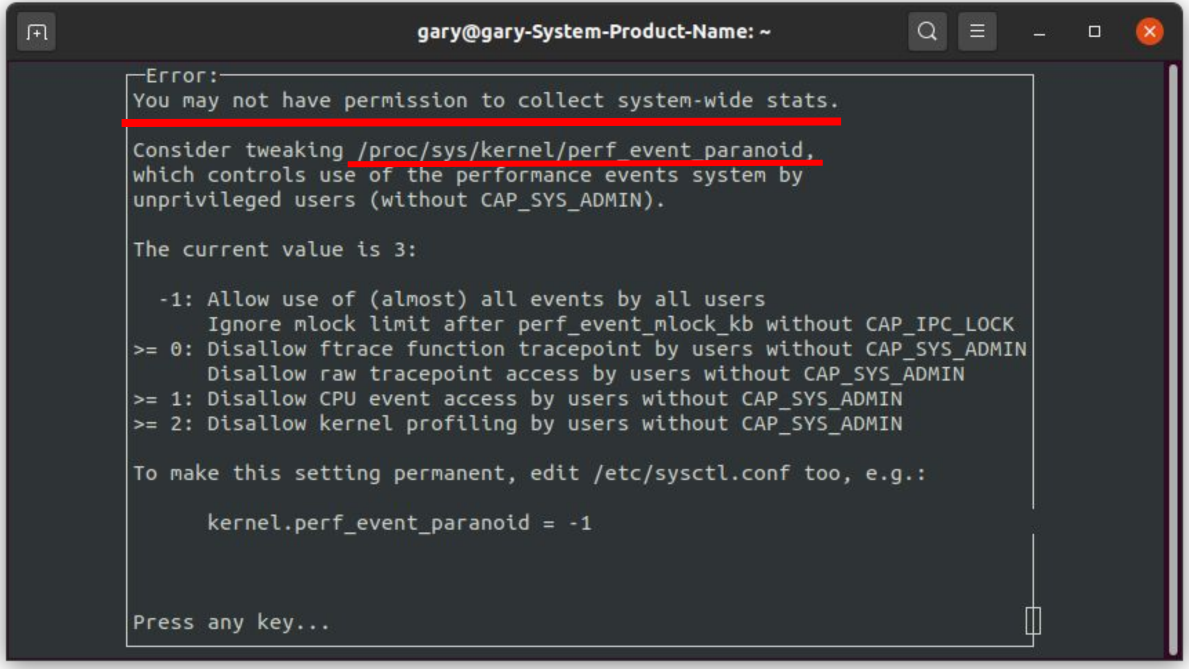
```
$ sudo apt install linux-tools-$(uname -r)  #Kernel 版本
```



```
gary@gary-BM1AF-BP1AF-BM6AF:~/文件$ sudo apt install linux-tools-$(uname -r)
[sudo] password for gary:
正在讀取套件清單... 完成
正在重建相依關係
正在讀取狀態資料... 完成
linux-tools-5.4.0-47-generic 已是最新版本 (5.4.0-47.51~18.04.1)。
```

How to use perf (Performance Event)

$ perf top



```
Error:
You may not have permission to collect system-wide stats.

Consider tweaking /proc/sys/kernel/perf_event_paranoid,
which controls use of the performance events system by
unprivileged users (without CAP_SYS_ADMIN).

The current value is 3:

  -1: Allow use of (almost) all events by all users
      Ignore mlock limit after perf_event_mlock_kb without CAP_IPC_LOCK
>= 0: Disallow ftrace function tracepoint by users without CAP_SYS_ADMIN
      Disallow raw tracepoint access by users without CAP_SYS_ADMIN
>= 1: Disallow CPU event access by users without CAP_SYS_ADMIN
>= 2: Disallow kernel profiling by users without CAP_SYS_ADMIN

To make this setting permanent, edit /etc/sysctl.conf too, e.g.:

      kernel.perf_event_paranoid = -1



Press any key...
```

切換至管理員

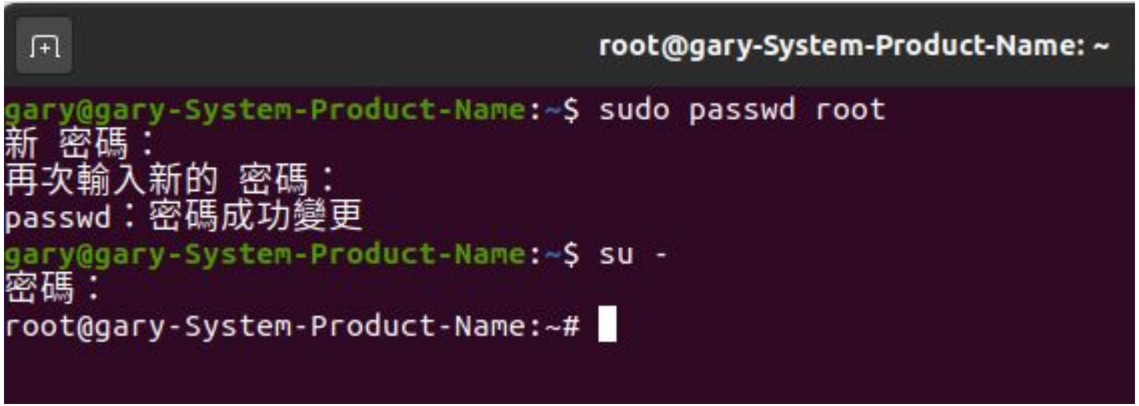
$ sudo passwd root

輸入密碼

$ su -

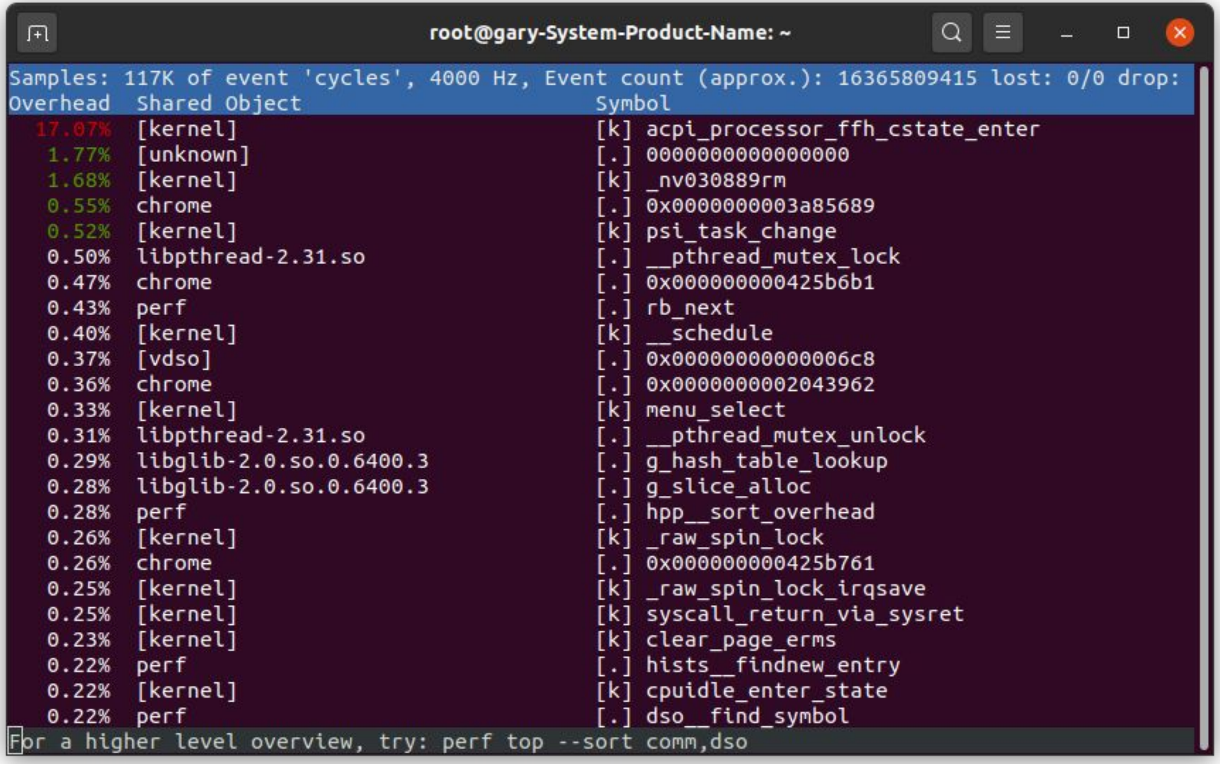
輸入密碼

# 切到root模式



```
gary@gary-System-Product-Name:~$ sudo passwd root
新 密碼：
再次輸入新的 密碼：
passwd：密碼成功變更
gary@gary-System-Product-Name:~$ su -
密碼：
root@gary-System-Product-Name:~#
```

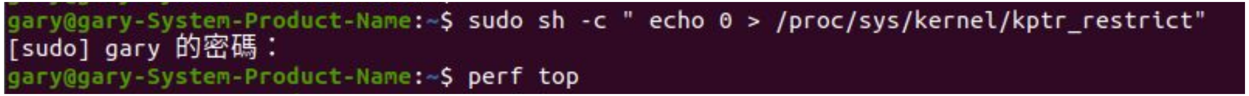
perf top



```
Samples: 117K of event 'cycles', 4000 Hz, Event count (approx.): 16365809415 lost: 0/0 drop:
Overhead  Shared Object                                    Symbol
  17.07%  [kernel]                                         [k] acpi_processor_ffh_cstate_enter
   1.77%  [unknown]                                        [.] 0000000000000000
   1.68%  [kernel]                                         [k] _nv030889rm
   0.55%  chrome                                           [.] 0x0000000003a85689
   0.52%  [kernel]                                         [k] psi_task_change
   0.50%  libpthread-2.31.so                               [.] __pthread_mutex_lock
   0.47%  chrome                                           [.] 0x000000000425b6b1
   0.43%  perf                                             [.] rb_next
   0.40%  [kernel]                                         [k] __schedule
   0.37%  [vdso]                                           [.] 0x00000000000006c8
   0.36%  chrome                                           [.] 0x0000000002043962
   0.33%  [kernel]                                         [k] menu_select
   0.31%  libpthread-2.31.so                               [.] __pthread_mutex_unlock
   0.29%  libglib-2.0.so.0.6400.3                          [.] g_hash_table_lookup
   0.28%  libglib-2.0.so.0.6400.3                          [.] g_slice_alloc
   0.28%  perf                                             [.] hpp__sort_overhead
   0.26%  [kernel]                                         [k] _raw_spin_lock
   0.26%  chrome                                           [.] 0x000000000425b761
   0.25%  [kernel]                                         [k] _raw_spin_lock_irqsave
   0.25%  [kernel]                                         [k] syscall_return_via_sysret
   0.23%  [kernel]                                         [k] clear_page_erms
   0.22%  perf                                             [.] hists__findnew_entry
   0.22%  [kernel]                                         [k] cpuidle_enter_state
   0.22%  perf                                             [.] dso__find_symbol
For a higher level overview, try: perf top --sort comm,dso
```

最後如果要檢測 cache miss event，需要先取消 kernel pointer 的禁用。

```
#   echo 0 > /proc/sys/kernel/kptr_restrict
```



```
gary@gary-System-Product-Name:~$ sudo sh -c " echo 0 > /proc/sys/kernel/kptr_restrict"
[sudo] gary 的密碼：
gary@gary-System-Product-Name:~$ perf top
```

RAM access rule of Thumb.

- 避免瑣碎性(Granularity):
  - 程式要循序存取的資料，盡量避免一次只存取太少的資料 (例如:單一次只存取一個 byte)。
- 善用局地性(Locality):
  - 程式要循序存取的資料，盡量考量資料間位址的，而避免資料間的位址差距太遠。
- 善用快取記憶體(Cache)
  - L1, L2 cache:
    - 最接近CPU，單一執行緒會反覆存取 (re-access)的小量資料(通常為16KB-256KB)，會被CPU的L1, L2 cache所存放，因而提高此執行緒之存取效能。
  - L3 cache:
    - 次於L1, L2 cache，所有執行緒會反覆存取 (re-access)的資料(通常為8MB-20MB)，會被CPU的L3 cache所存放，因而提高所有核心的存取效能。

Matrix Multiplication Experiment

- Implement the Matrix multiplication using C language.
- Compare three way methods for n×n matrix. ($n \in N$)
- Compare three strategy.

Matrix Multiplication (1/3)

Algorithm I

input : A,B are both n×n matrices. (the index of *n* from 0 ~ *n*-1)

output: C is a n×n matrix .

```
C ← $O$ (C is a zero matrix)
for each $i$ from 0 to n
    for each $j$ from 0 to n
        for each $k$ from 0 to n
            $C[i, j] = C[i, j]+A[i, k] \times B [k, j]$
```

Matrix Multiplication (2/3)

Algorithm II

input : A,B are both n×n matrices. (the index of *n* from 0 ~ *n*-1)

output: C is a n×n matrix .

```
C ← $O$ (C is a zero matrix)
for each $i$ from 0 to n
    for each $k$ from 0 to n
        for each $j$ from 0 to n
            $C[i, j] = C[i, j]+A[i, k] \times B [k, j]$
```

Matrix Multiplication (3/3)

Algorithm III

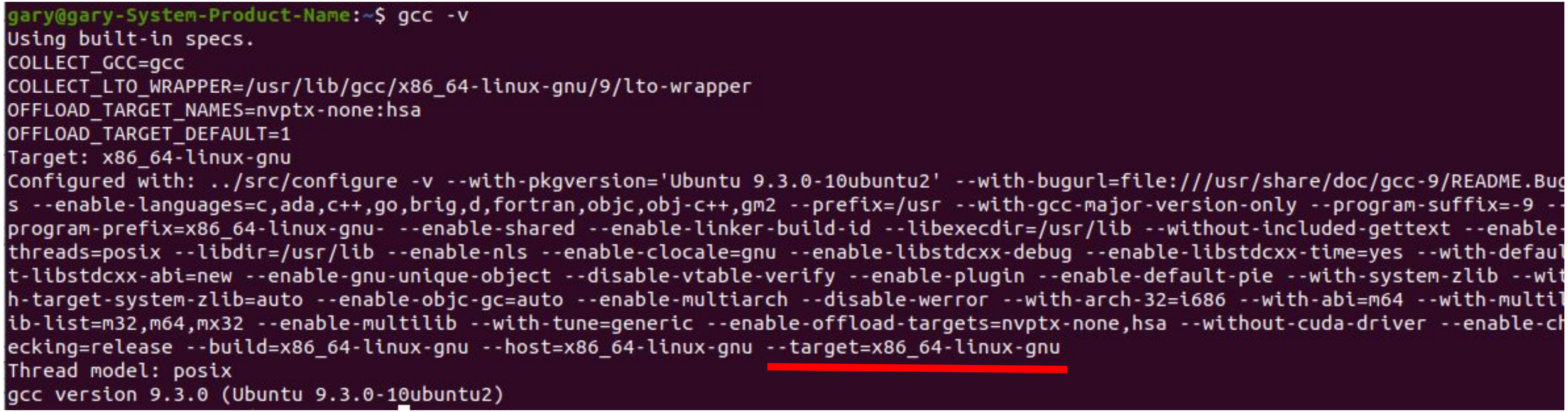
input : A,B are both n×n matrices. (the index of *n* from 0 ~ *n*-1)

output: C is a n×n matrix .

```
C ← $O$ (C is a zero matrix)
for each $j$ from 0 to n
    for each $k$ from 0 to n
        for each $i$ from 0 to n
            $C[i, j] = C[i, j]+A[i, k] \times B[k, j]$
```

GCC-complier

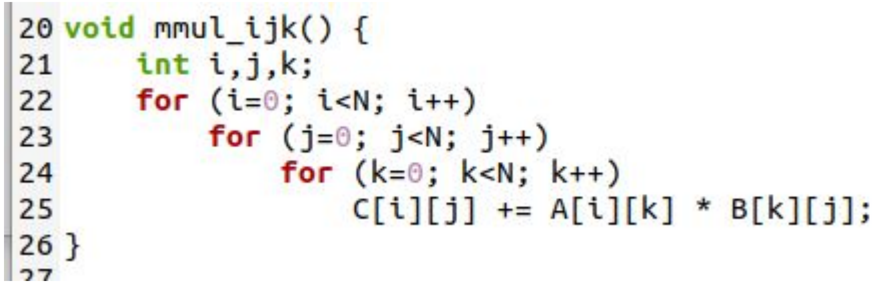
- 64位元



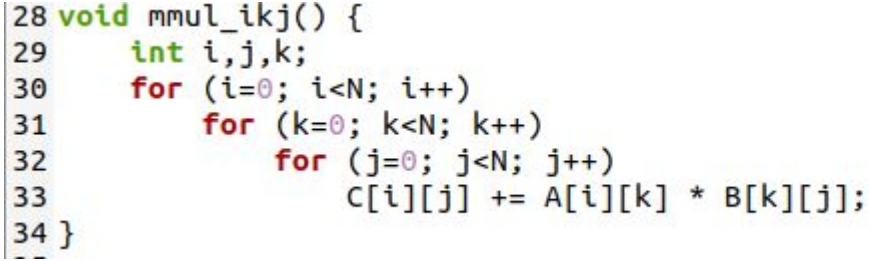
```
gary@gary-System-Product-Name:~$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/9/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none:hsa
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 9.3.0-10ubuntu2' --with-bugurl=file:///usr/share/doc/gcc-9/README.Bug
s --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++,gm2 --prefix=/usr --with-gcc-major-version-only --program-suffix=-9 --
program-prefix=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-
threads=posix --libdir=/usr/lib --enable-nls --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-defaul
t-libstdcxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-plugin --enable-default-pie --with-system-zlib --wit
h-target-system-zlib=auto --enable-objc-gc=auto --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multil
ib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-offload-targets=nvptx-none,hsa --without-cuda-driver --enable-ch
ecking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 9.3.0 (Ubuntu 9.3.0-10ubuntu2)
```

Matrix Multiplication

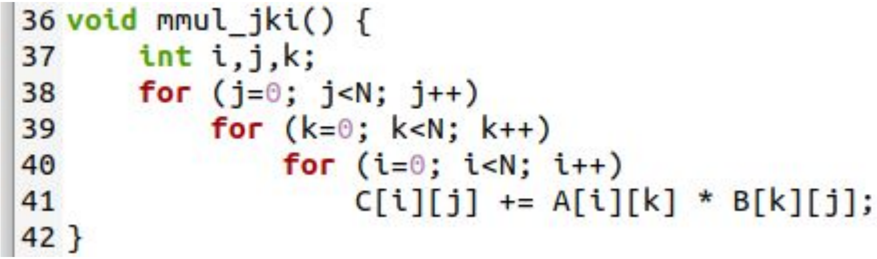
- 機器學習裡常用矩陣相乘
- A and B are 1000×1000 matrices.
- C = A B
- Compare 3 way in C language.
- 使用 gcc -O0 (沒有最佳化模式)



```c
void mmul_ijk() {
    int i,j,k;
    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
            for (k=0; k<N; k++)
                C[i][j] += A[i][k] * B[k][j];
}
```



```c
void mmul_ikj() {
    int i,j,k;
    for (i=0; i<N; i++)
        for (k=0; k<N; k++)
            for (j=0; j<N; j++)
                C[i][j] += A[i][k] * B[k][j];
}
```



```c
void mmul_jki() {
    int i,j,k;
    for (j=0; j<N; j++)
        for (k=0; k<N; k++)
            for (i=0; i<N; i++)
                C[i][j] += A[i][k] * B[k][j];
}
```

Locality示意圖

- i, j, k: C[i][j] += A[i][k] * B[k][j]

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

- i, k, j: C[i][j] += A[i][k] * B[k][j]

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

- j, k, i: C[i][j] += A[i][k] * B[k][j]

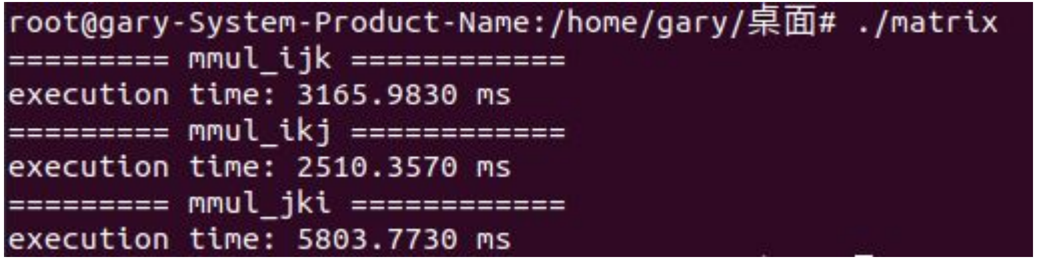
0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

Result (1/2)

- Programs use timer API

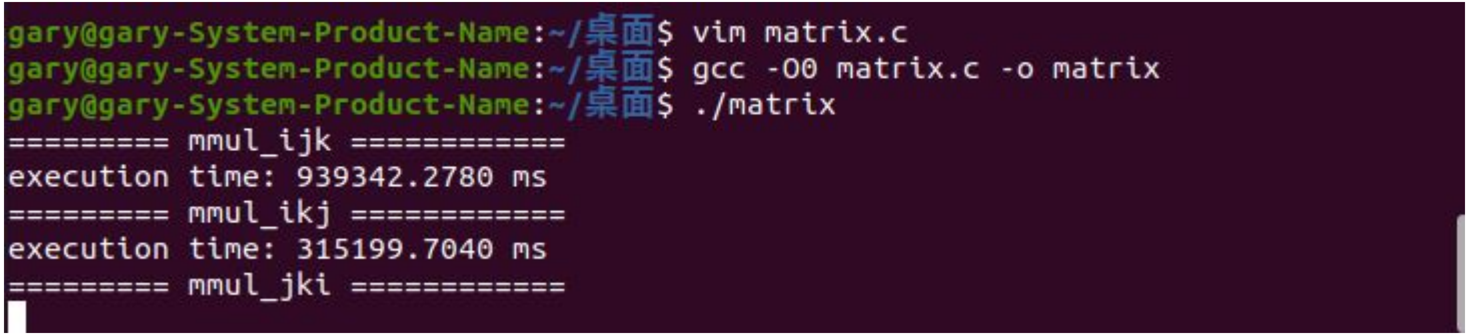


```
root@gary-System-Product-Name:/home/gary/桌面# ./matrix
========= mmul_ijk ============
execution time: 3165.9830 ms
========= mmul_ikj ============
execution time: 2510.3570 ms
========= mmul_jki ============
execution time: 5803.7730 ms
```

- mmul_ijk() 執行時間: 3.1s
- mmul_ikj() 執行時間: 2.5s
- mmul_jki() 執行時間: 5.8s
- 結論 mmul_ikj()只花2秒，比mmul_jki()的5秒快上了約2倍 ！

Result (2/2)

- 測試5000×5000的矩陣相乘

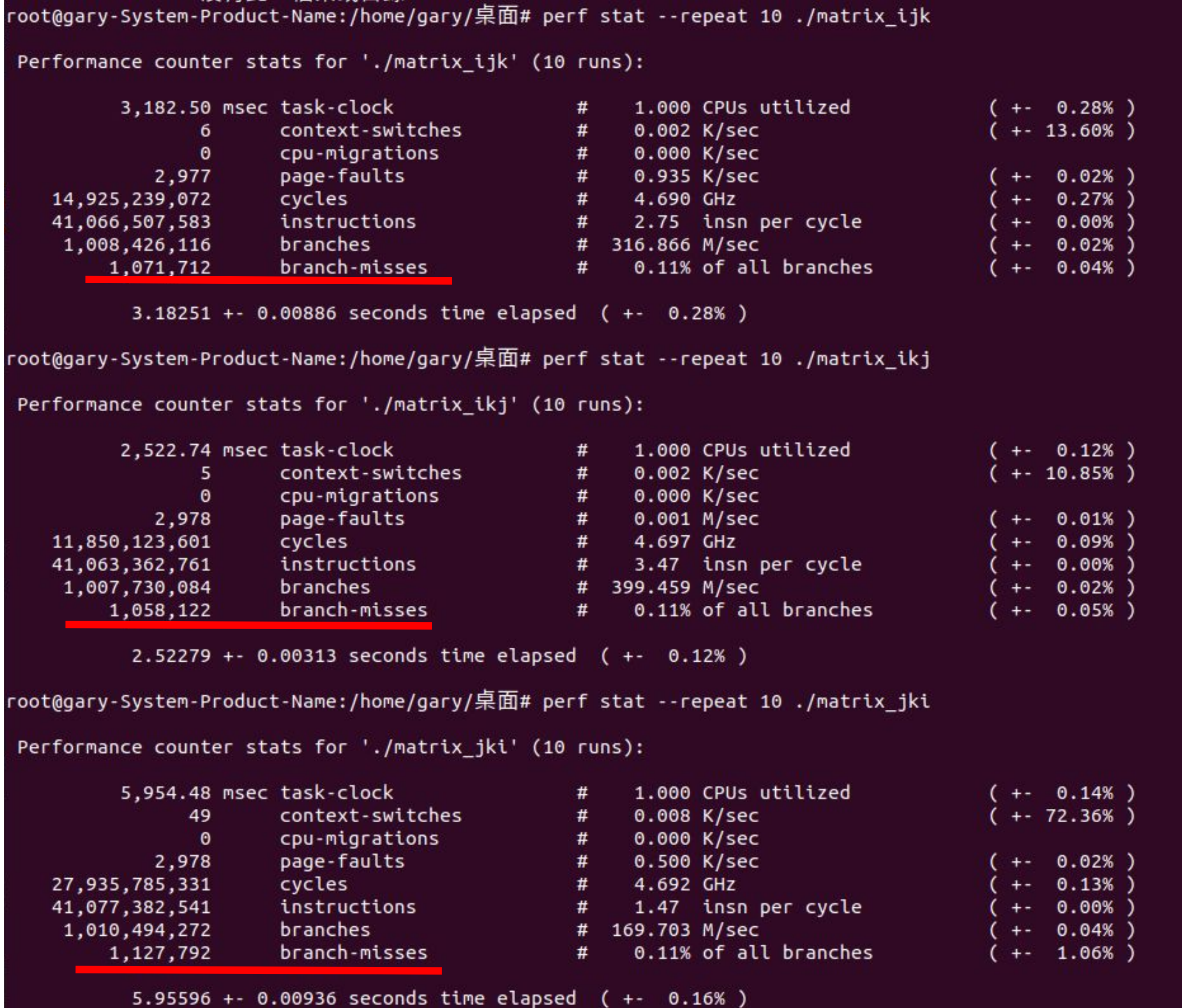


```
gary@gary-System-Product-Name:~/桌面$ vim matrix.c
gary@gary-System-Product-Name:~/桌面$ gcc -O0 matrix.c -o matrix
gary@gary-System-Product-Name:~/桌面$ ./matrix
========= mmul_ijk ============
execution time: 939342.2780 ms
========= mmul_ikj ============
execution time: 315199.7040 ms
========= mmul_jki ============
```

- mmul_ikj()比mmul_ijk()快將近三倍！

perf status

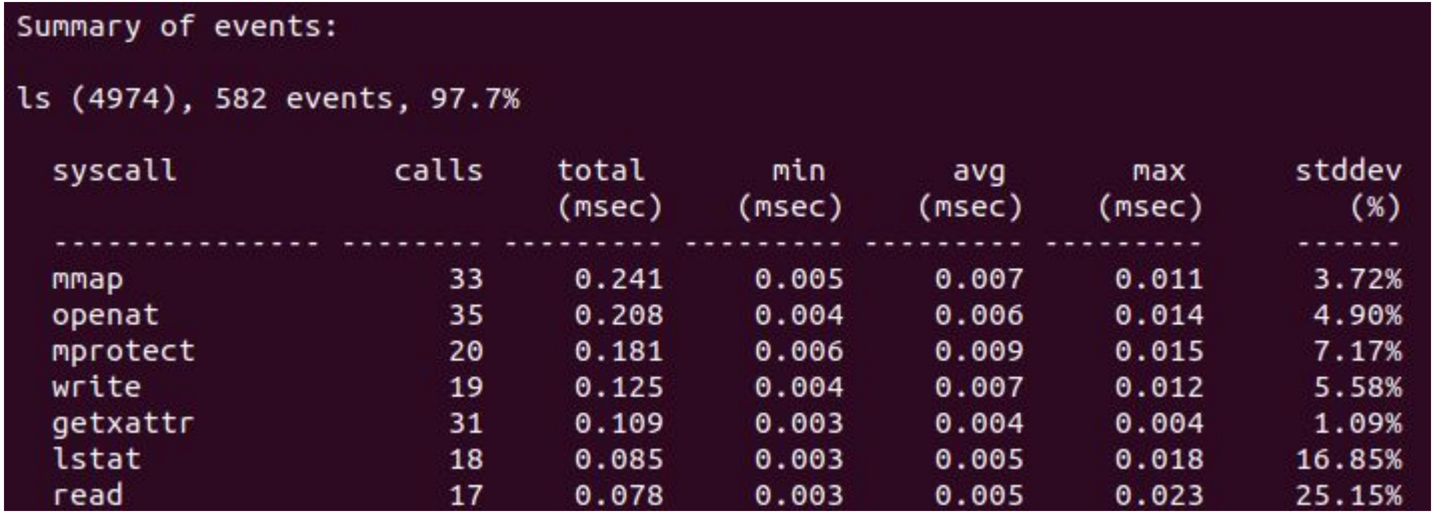
- 使用 perf stat 往往是你已經有個要優化的目標，對這個目標進行特定或一系列的 event 檢查，進而了解該程序的效能概況。
- 採用perf status 查看執行結果
  - cache-misses,cache-references,instructions,cycles



```
root@gary-System-Product-Name:/home/gary/桌面# perf stat --repeat 10 ./matrix_ijk

 Performance counter stats for './matrix_ijk' (10 runs):

          3,182.50 msec task-clock                #    1.000 CPUs utilized            ( +-  0.28% )
                 6      context-switches          #    0.002 K/sec                    ( +- 13.60% )
                 0      cpu-migrations            #    0.000 K/sec
             2,977      page-faults               #    0.935 K/sec                    ( +-  0.02% )
    14,925,239,072      cycles                    #    4.690 GHz                      ( +-  0.27% )
    41,066,507,583      instructions              #    2.75  insn per cycle           ( +-  0.00% )
     1,008,426,116      branches                  #  316.866 M/sec                    ( +-  0.02% )
         1,071,712      branch-misses             #    0.11% of all branches          ( +-  0.04% )

           3.18251 +- 0.00886 seconds time elapsed  ( +-  0.28% )

root@gary-System-Product-Name:/home/gary/桌面# perf stat --repeat 10 ./matrix_ikj

 Performance counter stats for './matrix_ikj' (10 runs):

          2,522.74 msec task-clock                #    1.000 CPUs utilized            ( +-  0.12% )
                 5      context-switches          #    0.002 K/sec                    ( +- 10.85% )
                 0      cpu-migrations            #    0.000 K/sec
             2,978      page-faults               #    0.001 M/sec                    ( +-  0.01% )
    11,850,123,601      cycles                    #    4.697 GHz                      ( +-  0.09% )
    41,063,362,761      instructions              #    3.47  insn per cycle           ( +-  0.00% )
     1,007,730,084      branches                  #  399.459 M/sec                    ( +-  0.02% )
         1,058,122      branch-misses             #    0.11% of all branches          ( +-  0.05% )

           2.52279 +- 0.00313 seconds time elapsed  ( +-  0.12% )

root@gary-System-Product-Name:/home/gary/桌面# perf stat --repeat 10 ./matrix_jki

 Performance counter stats for './matrix_jki' (10 runs):

          5,954.48 msec task-clock                #    1.000 CPUs utilized            ( +-  0.14% )
                49      context-switches          #    0.008 K/sec                    ( +- 72.36% )
                 0      cpu-migrations            #    0.000 K/sec
             2,978      page-faults               #    0.500 K/sec                    ( +-  0.02% )
    27,935,785,331      cycles                    #    4.692 GHz                      ( +-  0.13% )
    41,077,382,541      instructions              #    1.47  insn per cycle           ( +-  0.00% )
     1,010,494,272      branches                  #  169.703 M/sec                    ( +-  0.04% )
         1,127,792      branch-misses             #    0.11% of all branches          ( +-  1.06% )

           5.95596 +- 0.00936 seconds time elapsed  ( +-  0.16% )
```

Observe I/O

# perf trace -s ls -al



```
Summary of events:

ls (4974), 582 events, 97.7%

   syscall            calls    total       min       avg       max      stddev
                               (msec)    (msec)    (msec)    (msec)        (%)
   --------------- -------- --------- --------- --------- ---------     ------
   mmap                  33     0.241     0.005     0.007     0.011      3.72%
   openat                35     0.208     0.004     0.006     0.014      4.90%
   mprotect              20     0.181     0.006     0.009     0.015      7.17%
   write                 19     0.125     0.004     0.007     0.012      5.58%
   getxattr              31     0.109     0.003     0.004     0.004      1.09%
   lstat                 18     0.085     0.003     0.005     0.018     16.85%
   read                  17     0.078     0.003     0.005     0.023     25.15%
```

Computer multitasking

- 硬體差異(主要)

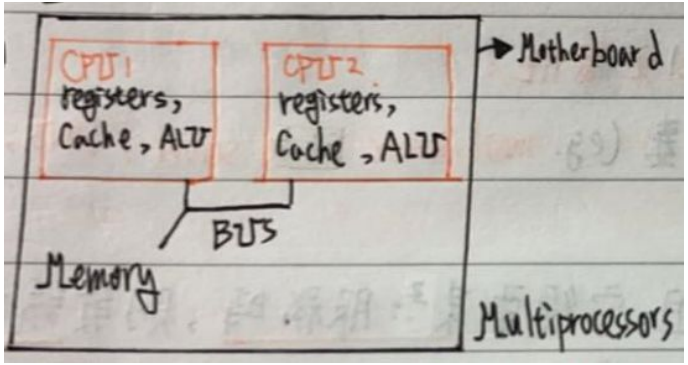


Figure1. Multiprocessors

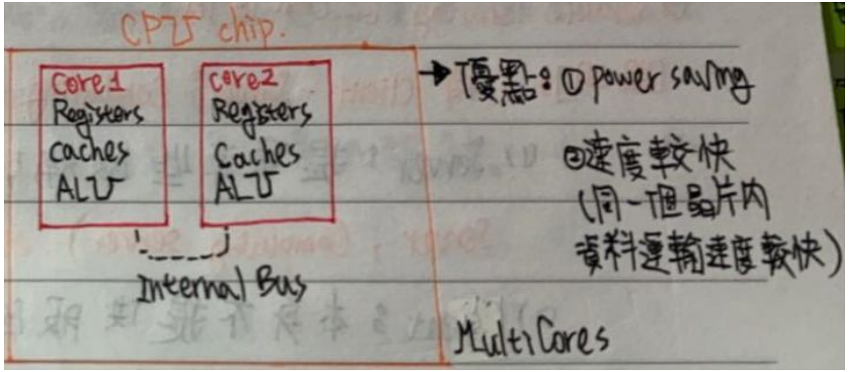
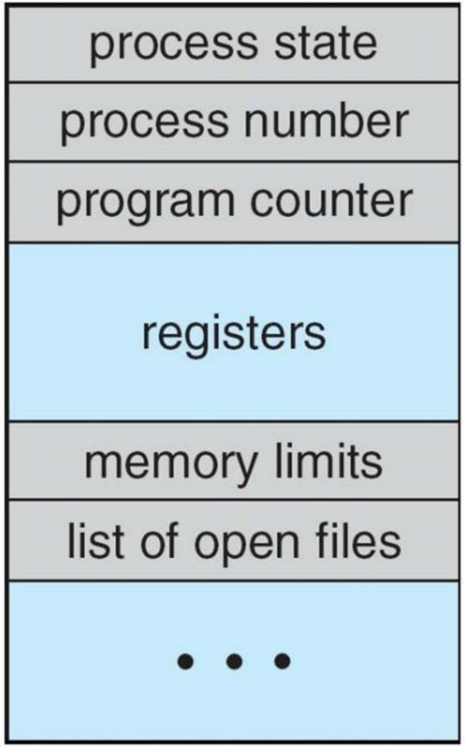


Figure2.　MultiCores (多核)

- 以OS的角度，1 core = 1 logical CPU

Process Control Block (PCB)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all processcentric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



process state

process number

program counter

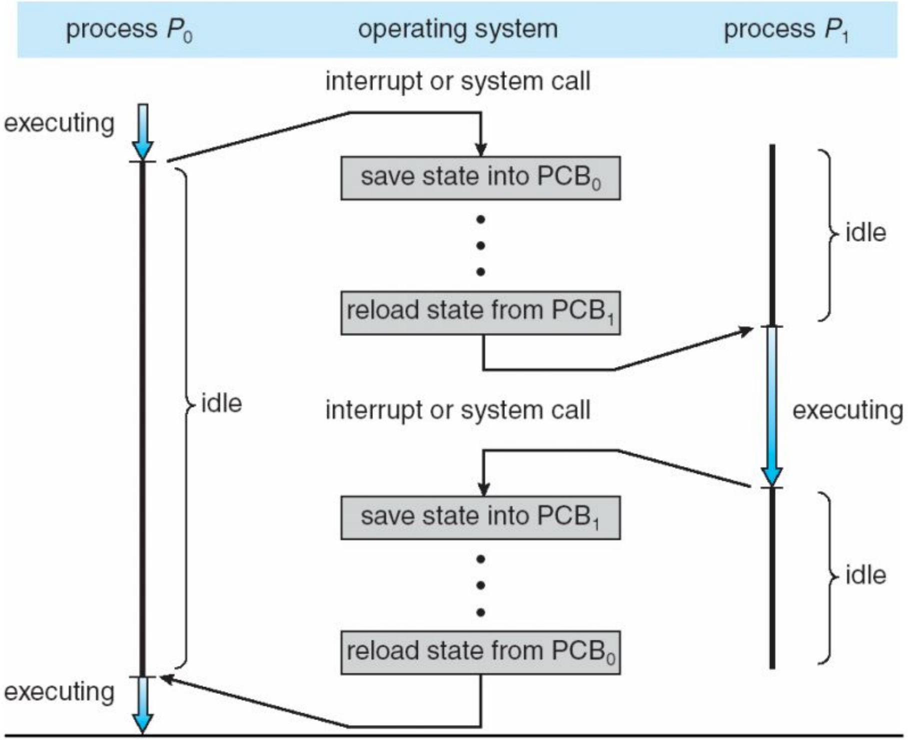
registers

memory limits

list of open files

• • •

CPU Switch From Process to Process



Schedulers

- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)
- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow)
  - The long-term scheduler controls the degree of multiprogramming
- Processes can be described as either:
  - I/O-bound process – spends more time doing I/O than computations, many short CPU bursts
  - CPU-bound process – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good ***process mix***

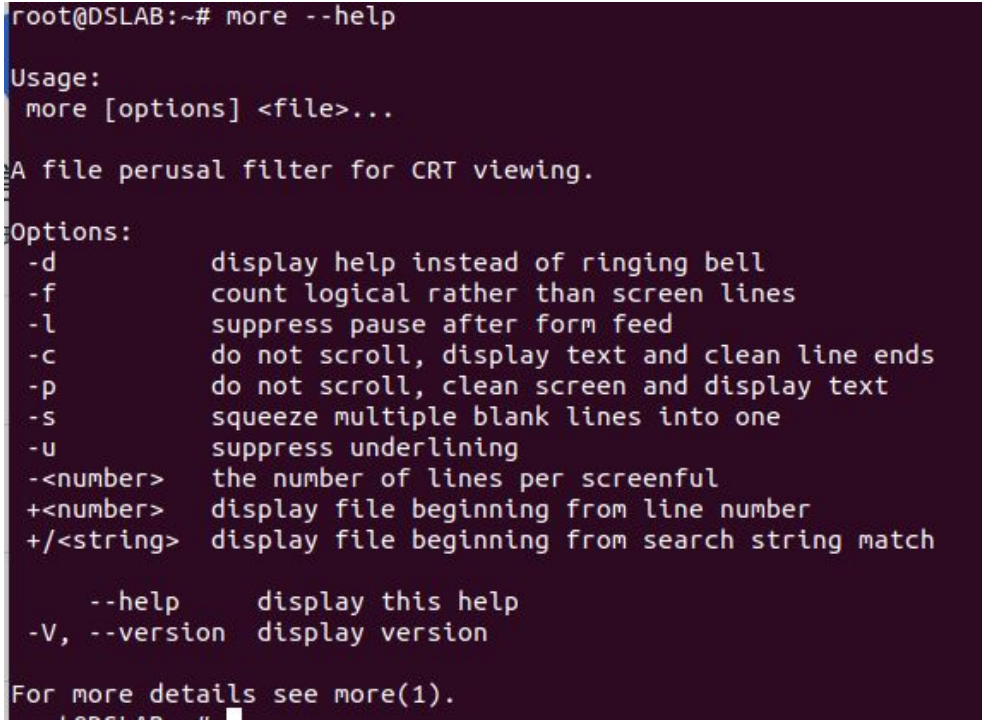
將perf結果寫到檔案裡

- 可以用來找出最佔用CPU 的process。
- 下面的指令對系統 CPU 事件做取樣，取樣時間為60 秒，每秒取樣99 個事件。

```
# perf record -F 99 -a -g -- sleep 60
```

- 執行這個指令會產生一個 perf.data 檔案：
  - 執行 `sudo perf report -n` 可以預覽報告。
  - 執行 `sudo perf report -n --stdio` 可以產生一個詳細的报告。
  - 執行 `sudo perf script` 可以 dump 出 perf.data 的内容。
- 也可以紀錄某一個process的事件，eg.紀錄process id 為 1641 的process:
  - `$ sudo perf record -F 99 -p 1641 -g -- sleep 60`
  - `$ sudo perf script > out.perf    # 将 perf.data 的内容 dump 到 out.perf`

利用more指令列出大檔案



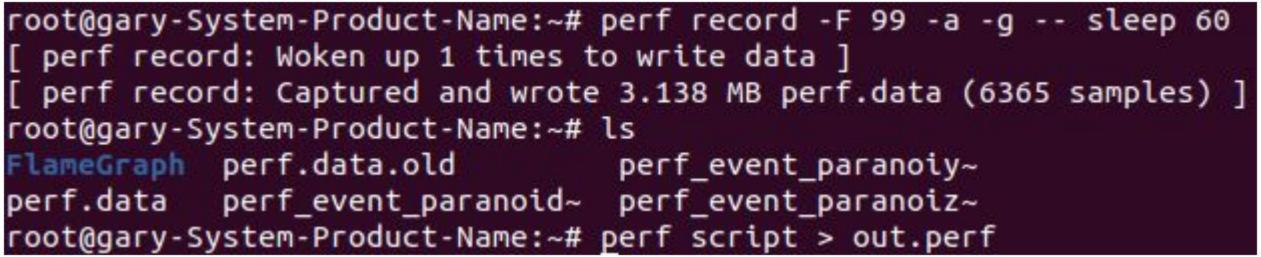
```
root@DSLAB:~# more --help

Usage:
 more [options] <file>...

A file perusal filter for CRT viewing.

Options:
 -d          display help instead of ringing bell
 -f          count logical rather than screen lines
 -l          suppress pause after form feed
 -c          do not scroll, display text and clean line ends
 -p          do not scroll, clean screen and display text
 -s          squeeze multiple blank lines into one
 -u          suppress underlining
 -<number>   the number of lines per screenful
 +<number>   display file beginning from line number
 +/<string>  display file beginning from search string match

     --help     display this help
 -V, --version  display version

For more details see more(1).
```

Flame Graphs visualize (1/3)

- 安裝火焰圖程式

```
# apt install git
```

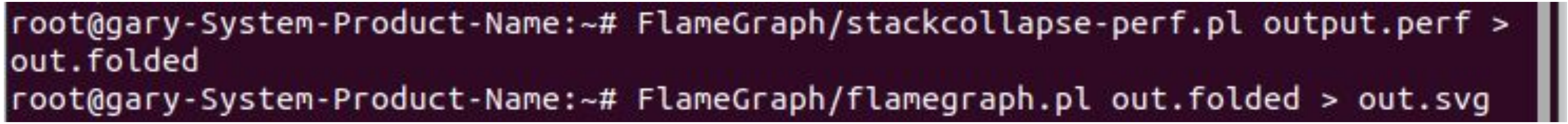
```
# git clone --depth 1 https://github.com/brendangregg/FlameGraph.git
```



```
root@gary-System-Product-Name:~# perf record -F 99 -a -g -- sleep 60
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 3.138 MB perf.data (6365 samples) ]
root@gary-System-Product-Name:~# ls
FlameGraph  perf.data.old          perf_event_paranoiy~
perf.data   perf_event_paranoid~   perf_event_paranoiz~
root@gary-System-Product-Name:~# perf script > out.perf
```

Flame Graphs visualize (2/3)

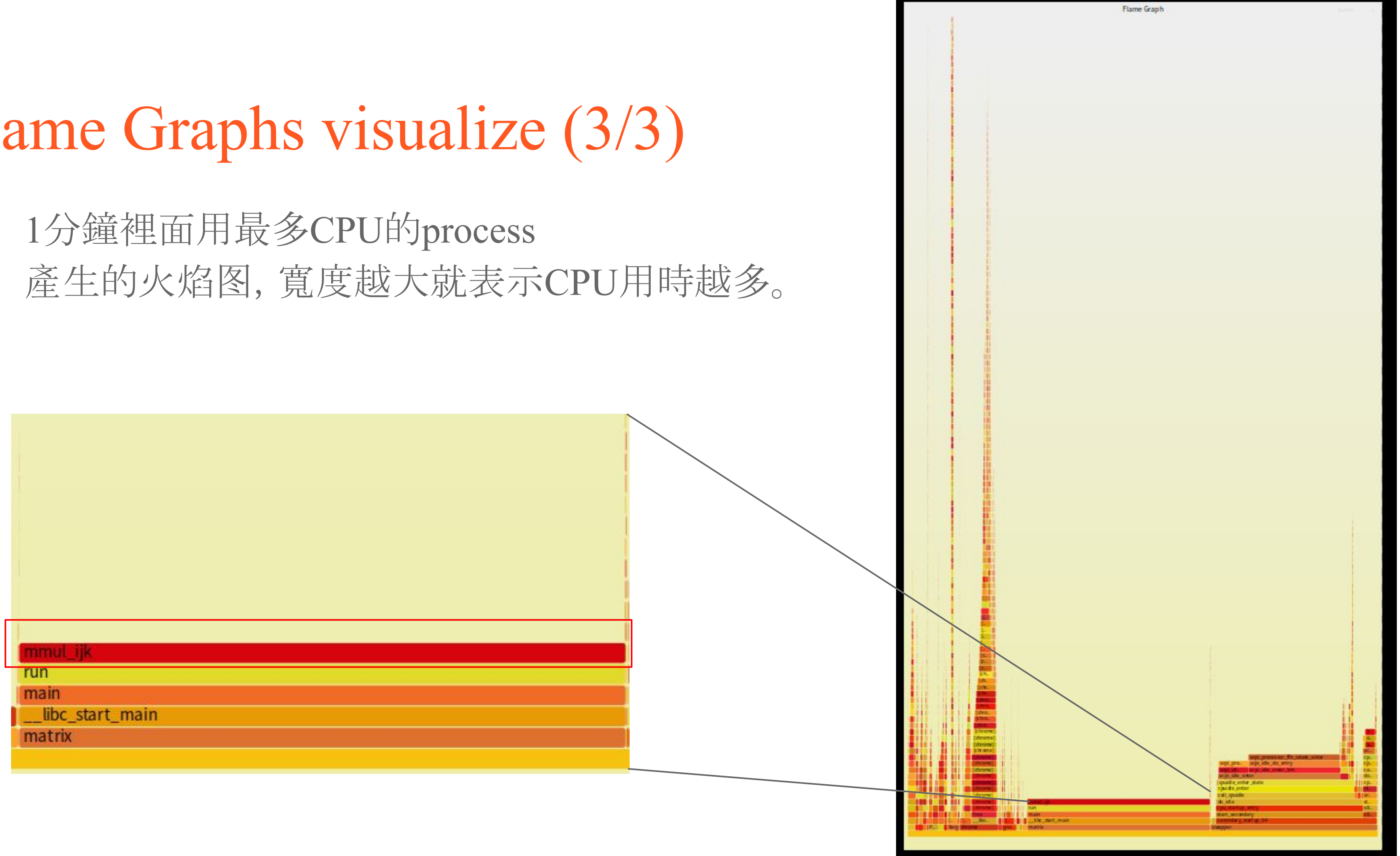
```
# 折疊調用堆疊

$ FlameGraph/stackcollapse-perf.pl out.perf > out.folded

# 產生火焰圖

$ FlameGraph/flamegraph.pl out.folded > out.svg
```



```
root@gary-System-Product-Name:~# FlameGraph/stackcollapse-perf.pl output.perf >
out.folded
root@gary-System-Product-Name:~# FlameGraph/flamegraph.pl out.folded > out.svg
```

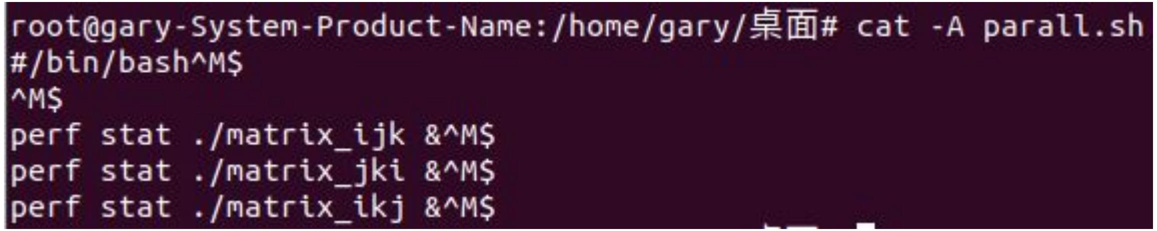
Flame Graphs visualize (3/3)

- 1分鐘裡面用最多CPU的process
- 產生的火焰图，寬度越大就表示CPU用時越多。



Windows文件換行轉linux換行(1/2)

- 能看到所有的換行
  - # cat -A [Filename]

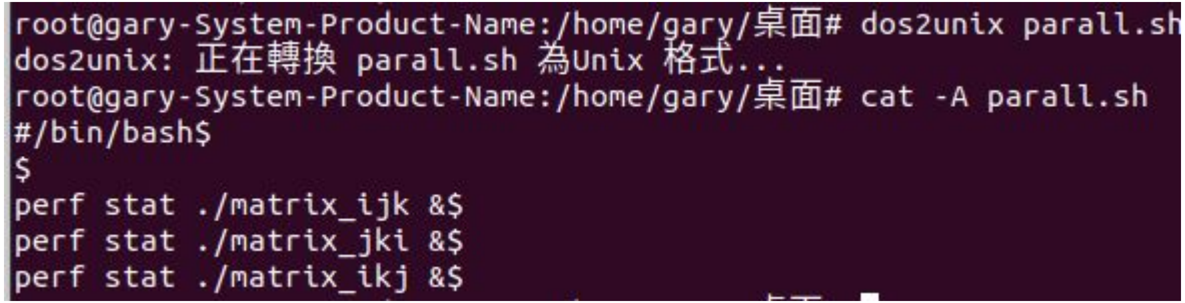


```
root@gary-System-Product-Name:/home/gary/桌面# cat -A parall.sh
#/bin/bash^M$
^M$
perf stat ./matrix_ijk &^M$
perf stat ./matrix_jki &^M$
perf stat ./matrix_ikj &^M$
```

- 看到的是一個Windows形式的換行符號, \r對應符號^M, \n對應的是$

Windows文件換行轉linux換行(2/2)

- 安裝dos2unix
  - # apt-get install dos2unix



```
root@gary-System-Product-Name:/home/gary/桌面# dos2unix parall.sh
dos2unix: 正在轉換 parall.sh 為Unix 格式...
root@gary-System-Product-Name:/home/gary/桌面# cat -A parall.sh
#/bin/bash$
$
perf stat ./matrix_ijk &$
perf stat ./matrix_jki &$
perf stat ./matrix_ikj &$
```

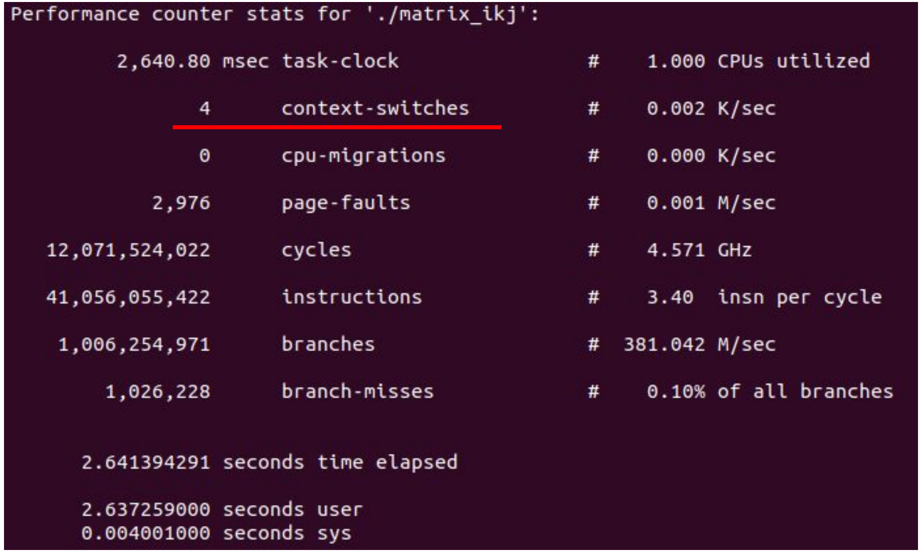
計算機多工測試 (1/3)

- parall.sh (bash檔案)

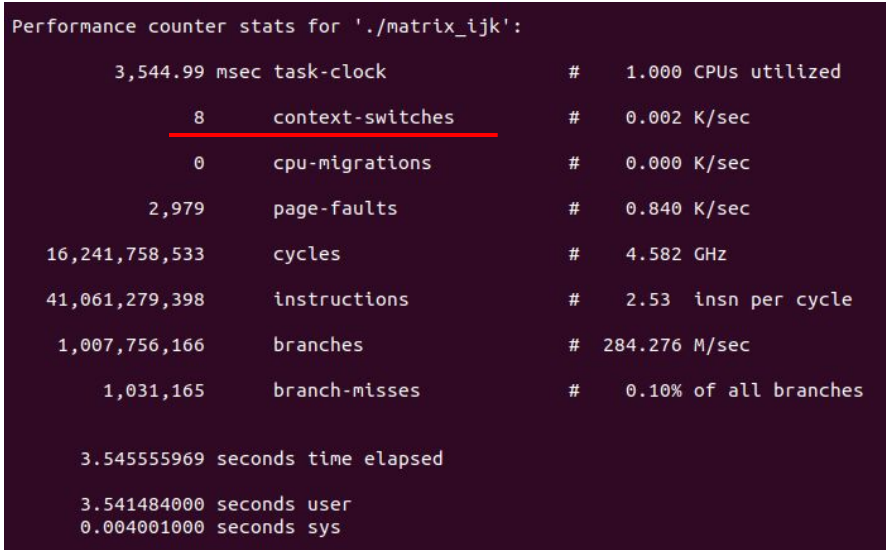
  ```
  #/bin/bash
  perf stat command &
  perf stat command &
  perf stat command &
  ```

- # bash parall.sh

計算機多工測試 (2/3)



```
Performance counter stats for './matrix_ikj':

         2,640.80 msec task-clock                #    1.000 CPUs utilized
                4      context-switches          #    0.002 K/sec
                0      cpu-migrations            #    0.000 K/sec
            2,976      page-faults               #    0.001 M/sec
   12,071,524,022      cycles                    #    4.571 GHz
   41,056,055,422      instructions              #    3.40  insn per cycle
    1,006,254,971      branches                  #  381.042 M/sec
        1,026,228      branch-misses             #    0.10% of all branches

      2.641394291 seconds time elapsed

      2.637259000 seconds user
      0.004001000 seconds sys
```

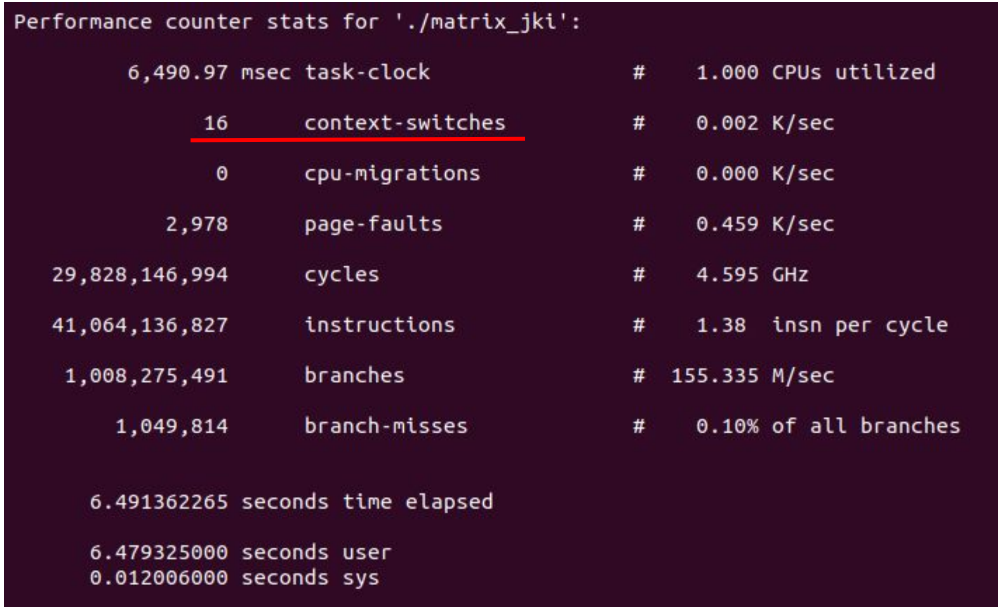


```
Performance counter stats for './matrix_ijk':

         3,544.99 msec task-clock                #    1.000 CPUs utilized
                8      context-switches          #    0.002 K/sec
                0      cpu-migrations            #    0.000 K/sec
            2,979      page-faults               #    0.840 K/sec
   16,241,758,533      cycles                    #    4.582 GHz
   41,061,279,398      instructions              #    2.53  insn per cycle
    1,007,756,166      branches                  #  284.276 M/sec
        1,031,165      branch-misses             #    0.10% of all branches

      3.545555969 seconds time elapsed

      3.541484000 seconds user
      0.004001000 seconds sys
```

計算機多工測試 (3/3)

Compare to context-switches

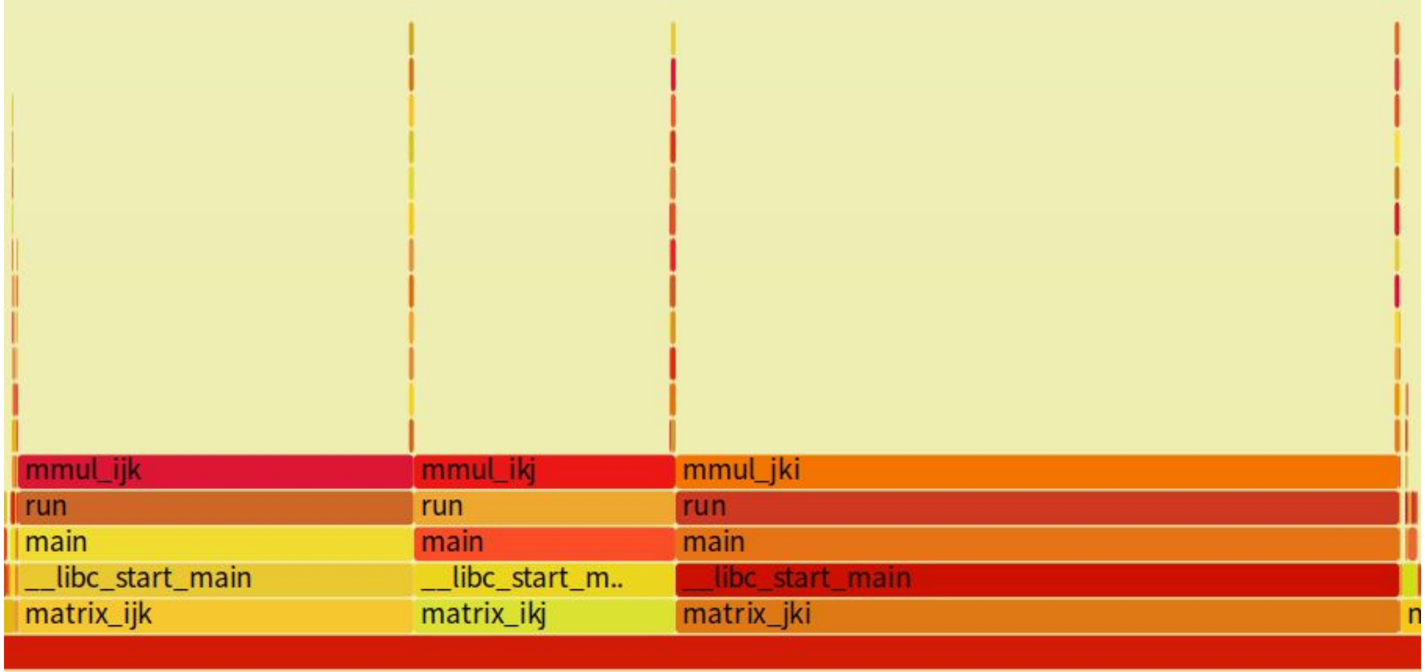
Function	number of context-switches
mmul_ijk()	8
mmul_ikj()	4
mmul_jki()	16



```
Performance counter stats for './matrix_jki':

        6,490.97 msec task-clock                #    1.000 CPUs utilized
              16      context-switches          #    0.002 K/sec
               0      cpu-migrations            #    0.000 K/sec
           2,978      page-faults               #    0.459 K/sec
  29,828,146,994      cycles                    #    4.595 GHz
  41,064,136,827      instructions              #    1.38  insn per cycle
   1,008,275,491      branches                  #  155.335 M/sec
       1,049,814      branch-misses             #    0.10% of all branches

     6.491362265 seconds time elapsed

     6.479325000 seconds user
     0.012006000 seconds sys
```

Flame Graphs

X軸為時間, Y軸為堆疊深度



4K對齊

- 使用SSD在切割分割區
  - SSD內部最小的寫入單位是4KB(1 page)
  - Linux的磁碟分割軟體，在預設情況下切割出來的分割區就已經有4K對齊了。
  - # fdisk -lu

Reference (1/5)

- Ellis Horowitz, Sartaj Sahni,and Susan Anderson-Freed. *Fundamentals of Data Structure in C.* p.342
- Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems, 3rd Edition.* p.426
- 外排序法External sorting
  - http://www.csbio.unc.edu/mcmillan/Media/Comp521F10Lecture17.pdf
    - http://courses.cs.vt.edu/~cs2604/fall04/docs/C8.pdf
  - http://www.ittc.ku.edu/~jsv/Papers/Vit.IO_book.pdf?fbclid=IwAR1NgR5jYEydpUVJJZIi1yuIALwo5pHDlDLRoUYE2U55GpZWjQfi86WFGS4

Reference (2/5)

- Windows文件換行轉linux換行
  - https://blog.csdn.net/CJF_iceKing/article/details/47836201
- Timsort
  - https://hackernoon.com/timsort-the-fastest-sorting-algorithm-youve-never-heard-of-36b28417f399?fbclid=IwAR3n5VB-Ttd9c9jJCdNuCUZr8hWWLJ_VsX1AT2w_oYO4HOM83fPUESHI-W8
  - https://svn.python.org/projects/python/trunk/Objects/listsort.txt
  - https://www.itread01.com/content/1545794102.html

Reference (3/5)

- Ulimit
  - https://www.ibm.com/support/knowledgecenter/zh-tw/SS5RWK_3.5.0/com.ibm.discovery.es.in.doc/iiysiulimits.htm
- 程式人月刊 — 2018年7月號
  - https://medium.com/%E7%A8%8B%E5%BC%8F%E4%BA%BA%E6%9C%88%E5%88%8A/%E7%A8%8B%E5%BC%8F%E4%BA%BA%E6%9C%88%E5%88%8A-2018%E5%B9%B47%E6%9C%88%E8%99%9F-e0a59c1b2031
  - https://medium.com/%E7%A8%8B%E5%BC%8F%E4%BA%BA%E6%9C%88%E5%88%8A/gcc-%E5%B7%A5%E5%85%B7%E7%9A%84%E4%BD%BF%E7%94%A8-cc7775c84964
- 生成火焰圖
  - http://www.brendangregg.com/flamegraphs.html
  - http://senlinzhan.github.io/2018/03/18/perf/
  - https://medium.com/statementdog-engineering/using-framegraph-to-find-out-application-bottleneck-ac5596b01736

Reference (4/5)

- Linux commands to check your disk performance
  - https://www.zylk.net/en/web-2-0/blog/-/blogs/linux-commands-to-check-your-disk-performance
- 硬碟讀寫測試
  https://webcache.googleusercontent.com/search?q=cache:gy6DM6ZNn8IJ:https://www.itread01.com/content/1550200162.html+&cd=3&hl=zh-TW&ct=clnk&gl=tw
- top觀察使用量

  https://www.arthurtoday.com/2015/02/sort-cpu-and-memory-usage-with-top-command-in-ubuntu.html

Reference (5/5)

- 利用perf 指令查看效能
  - http://www.brendangregg.com/perf.html

    https://tigercosmos.xyz/post/2020/08/system/perf-basic/

    http://wiki.csie.ncku.edu.tw/embedded/perf-tutorial

    https://blog.csdn.net/zzhongcy/article/details/105512565
- 矩陣相乘

  http://programmermagazine.github.io/201402/htm/article2.html
- 如何優化SSD固態硬碟

  https://magiclen.org/linux-ssd-optimization/