University of Science & Technology of Hanoi

Multiprocessing: Problem & Resolution Median Filter Using Java Thread

ADVISOR

NGUYEN DUC MINH

GROUP MEMBERS

CAO ANH QUAN

DO SON TUNG

NGUYEN GIA KHANG

CONTENT

1. OVERVIEW OF MULTIPROCESSOR
   - ✓ Taxonomy of parallelism architecture
   - ✓ Challenges of multiprocessing
   - ✓ Types of multiprocessors
   - ✓ Problems with multiprocessors
   - ✓ Resolutions
2. IMPLEMENT MEDIAN FILTER USING JAVA THREAD

OVERVIEW OF MULTIPROCESSORS

TAXONOMY OF PARALLEL ARCHITECTURES

- By Flynn -> 4 categories:
  - SISD: single instruction stream, single data stream: uniprocessor
  - SIMD: single instruction stream, multiple data stream: exploit data-level parallelism
  - MISD: multiple instruction stream, single data stream: no commercial implementation
  - MIMD: multiple instruction stream, multiple data stream: exploit thread-level parallelism -> *chosen for general-purpose multiprocessors*

CHALLENGES OF PARALLEL PROCESSING

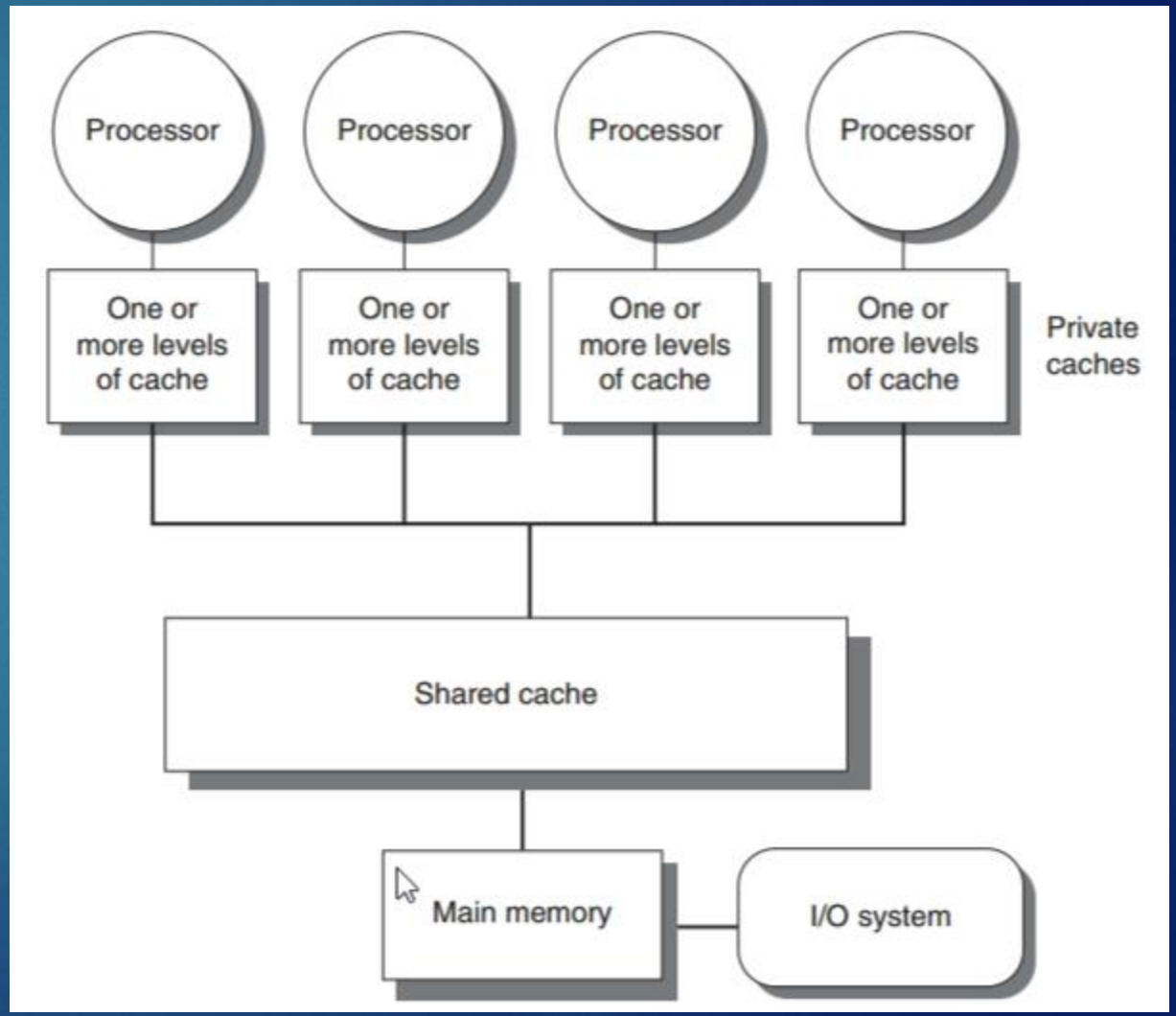
- Running parallel programs where threads must communicate to complete the task
- The large latency of remote access in a parallel processor

TYPES OF MULTIPROCESSORS

- 2 classes, depending on number of processors involved
  - 1st: **centralized shared – memory architecture**: few dozens processors
    - Also called symmetric multiprocessors (SMPs) & uniform memory access (UMA).
  - 2nd: **distributed shared memory** (DSM)
    - Also called non-uniform memory access (NUMA)

CENTRALIZED SHARED-MEMORY ARCHITECTURE (SMPs)

- Supports both shared & private data
- Private data:
  - Used by single processor
  - When cached, location migrates to cache -> reduce access time & memory bandwidth
- Shared data:
  - Used by multiprocessor
  - When cached, values are replicated into multiple caches -> reduce access latency, mem. bandwidth & conflicts in reading btw multiple processors
  - *-> new problem: cache coherence*



PERFORMANCE OF SMPs

- Coherence misses: arising from inter-processor communication, from 2 sources
  - True sharing misses: arising from communication of data through cache coherence mechanism
    - A write to shared cache block -> cause invalidation to create ownership
    - Read a modified word in that block -> occur miss, block is transferred
  - False sharing misses: arising from use of an invalidation-based coherence algorithm with a single valid bit per cache block.

PERFORMANCE OF SMPs

- False sharing misses:
  - Occur when a block is invalidated: some word in block is written into, other than being read.
  - If word being written ≠ word read & invalidation causes cache miss -> false sharing miss.
  - Block is shared, no word in cache is shared.
  - Not occur if block size is a single word

LIMITATION IN SMPs

- Number of processor / memory demands of each processor grow -> centralized resource become bottleneck
- Using the higher bandwidth connection available on-chip and a shared L3 cache

  ..not work once multiple multicores are combined.
- Use multiple buses & interconnection networks (crossbars or small point-to-point)

  ➡ increase communication bandwidth btw processors & memory
  - Memory system configured into multiple banks => boost mem bandwidth & retain uniform access time

LIMITATION IN SMPs

- Not scale well to a multichip multiprocessor using multicore building blocks since the memory is already attached to the individual multicore chips, rather than centralized

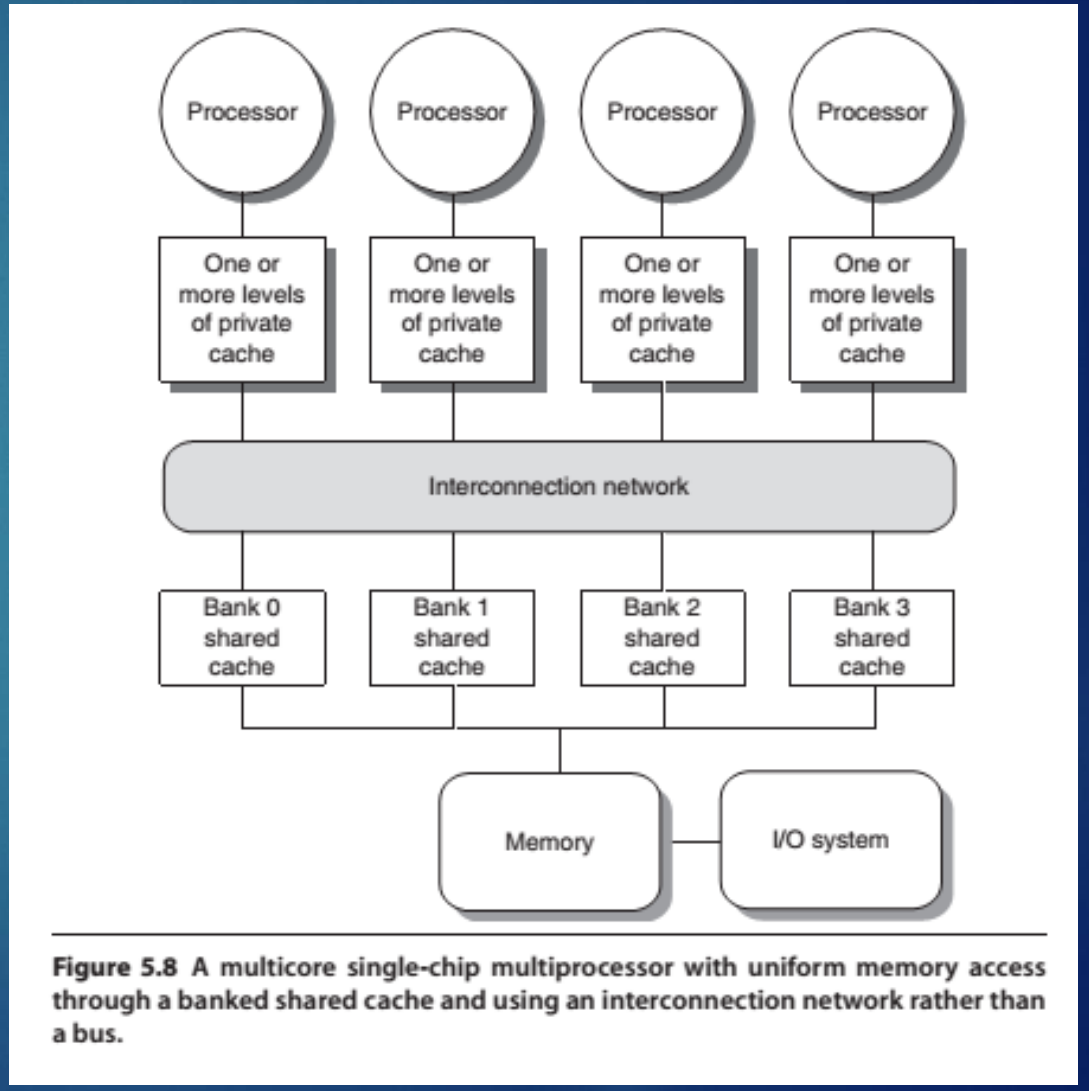


Figure 5.8 A multicore single-chip multiprocessor with uniform memory access through a banked shared cache and using an interconnection network rather than a bus.

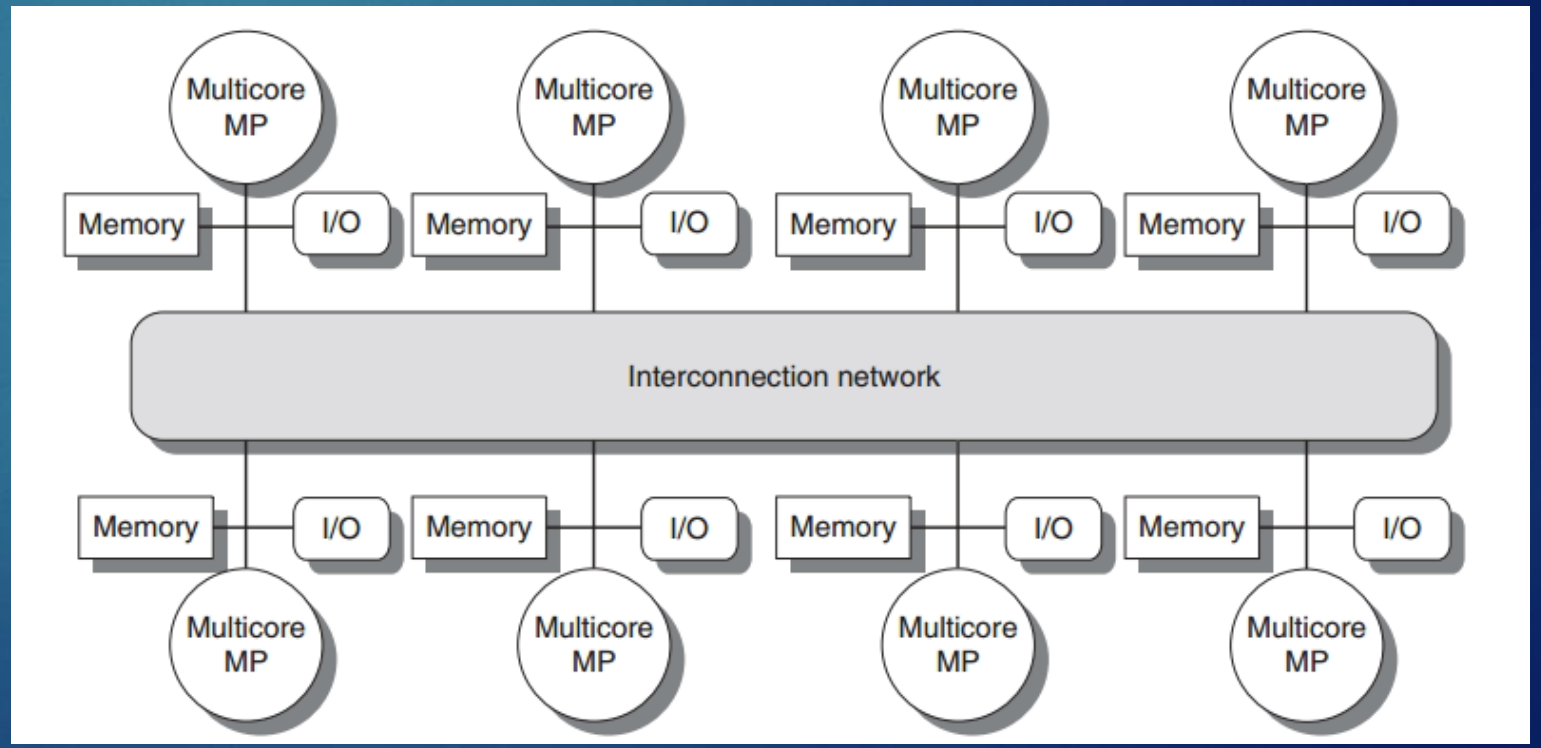
LIMITATION IN SMPs

- AMD Opteron: intermediate point btw snooping + directory protocol
- Memory directly connected to each multicore chip, up to 4
- System: NUMA <- local mem is faster
- Using point – to – point links ➡ broadcast up to 3 other chips
- Interprocessor links are not shared ➡ know when invalid operation completes by explicit acknowledgement
- Coherence protocol uses a broadcast to find potentially shared copies; uses the acknowledgments to order operations

DISTRIBUTED SHARED MEMORY

- Support larger processor counts
- Basic architecture: individual nodes contain a processor + memory + I/O + interface to an interconnection network that connects all the nodes

- 2 main benefits:
  - cost-effective way to scale the memory bandwidth if most of the accesses are to the local memory in the node
  - reduces the latency for accesses to the local memory

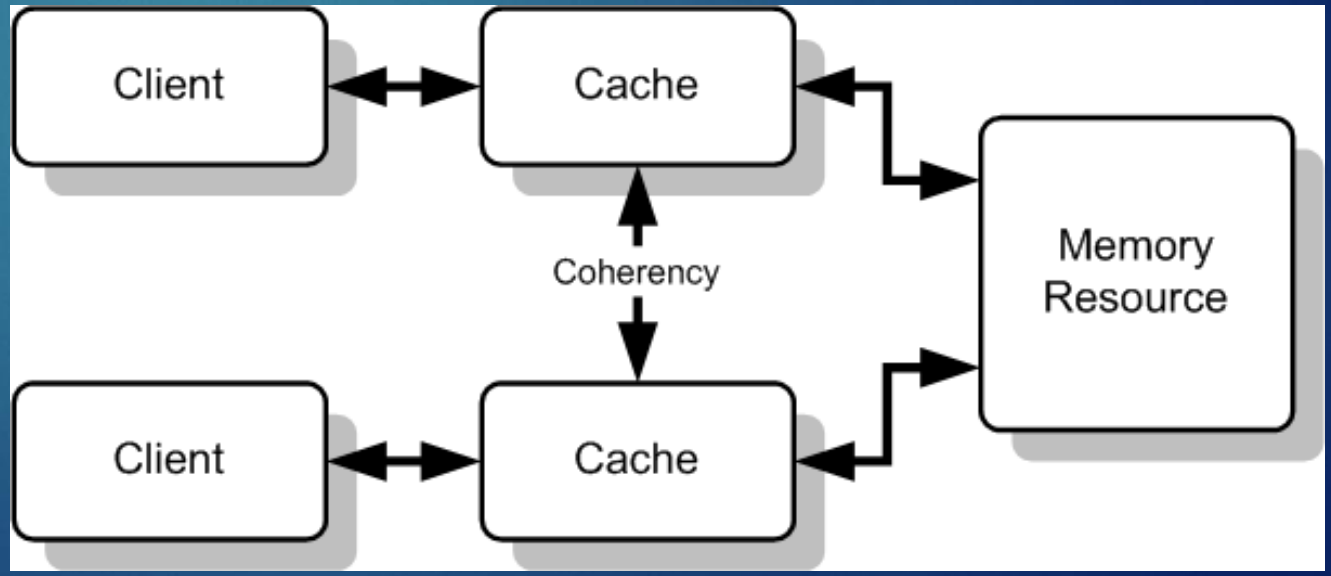


DISTRIBUTED SHARED MEMORY

- Main drawback: communicating btw processor is complex -> require more effort in software to take advantage of it
- Increase memory & interconnection bandwidth by distributing the memory
  - Separate local from remote memory traffic -> reduce bandwidth on mem system & interconnection networks
- Directory protocol: keep the state of block which may be cached

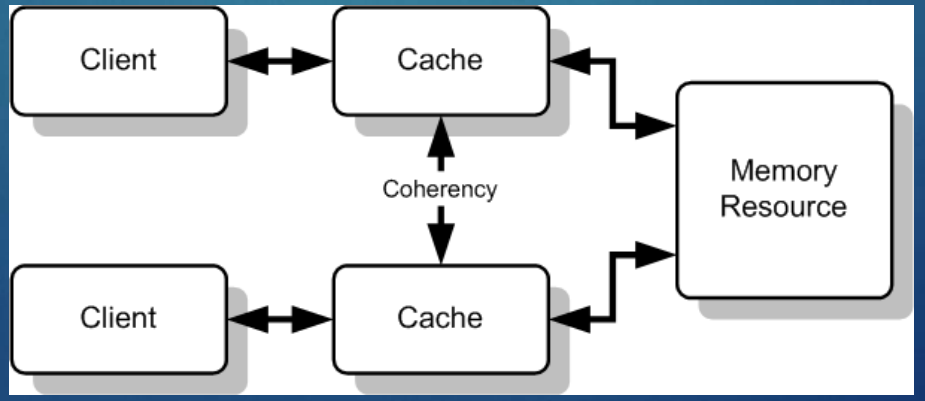
MULTIPROCESSOR CACHE COHERENCE

- The consistency of shared resource data that ends up stored in multiple local caches.
- Helps to manage the conflict and maintain the consistency between memory and cache



MULTIPROCESSOR CACHE COHERENCE

- 2 different processors have 2 different values for the same location
- Reason: existing 2 states: global (defined by main memory) & local (defined by individual caches)
- Coherence: define what values can be returned by a read
- Consistency: determine when a written value will be returned by a read



MULTIPROCESSOR CACHE COHERENCE

For P1, P2 are processors, X is memory location

- Coherence if:
  1. A read by P1 following a write by P1, no interrupts by another processor
  2. A read by P1 following a write by P2 returns the written value if P1 and P2 are sufficiently separated in time & no other writes interrupt btw 2 accesses
  3. Writes to the same X are serialized: the order of writing = the order of reading

     -> write serialization

MULTIPROCESSOR CACHE COHERENCE

- When a written value will be seen?

  Write on X by P1 precedes read on X by P2 by a very small time

  Written data may not have left P1 at the time P2 reads.

  -> read wrong value

  $\Rightarrow$ *memory consistent model*

- Coherence defines behaviors of reads & write to same memory location
- Consistency defines behaviors with respect to accesses to other memory location

BASIC SCHEMES FOR ENFORCING COHERENCE

- In coherent multiprocessor, caches provide both migration & replication
  - Migration: move data items to local caches -> reduce accessing latency & demanded bandwidth
  - Replication: make a copy of data item in local cache -> reduce accessing latency & contention for reading shared data
- Cache coherence protocols: to maintain coherence for multiple processors

  Idea: tracking the state of any sharing of a data block

SNOOPING PROTOCOL

- Invalidate all the bus about the shared block
- Update the value to all of the processor that have the copy of the block

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
				0
Processor A reads X	Cache miss for X	0		0
Processor B reads X	Cache miss for X	0	0	0
Processor A writes a 1 to X	Invalidation for X	1		0
Processor B reads X	Cache miss for X	1	1	1

SNOOPING PROTOCOL

- Write Invalidate Protocol:
  - Multiple readers, single writer
  - Write to shared data: an invalidate is sent to all caches which snoop and invalidate any copies
  - Read Miss:
    - » Write-through: memory is always up-to-date
    - » Write-back: snoop in caches to find most recent copy

SNOOPING PROTOCOL

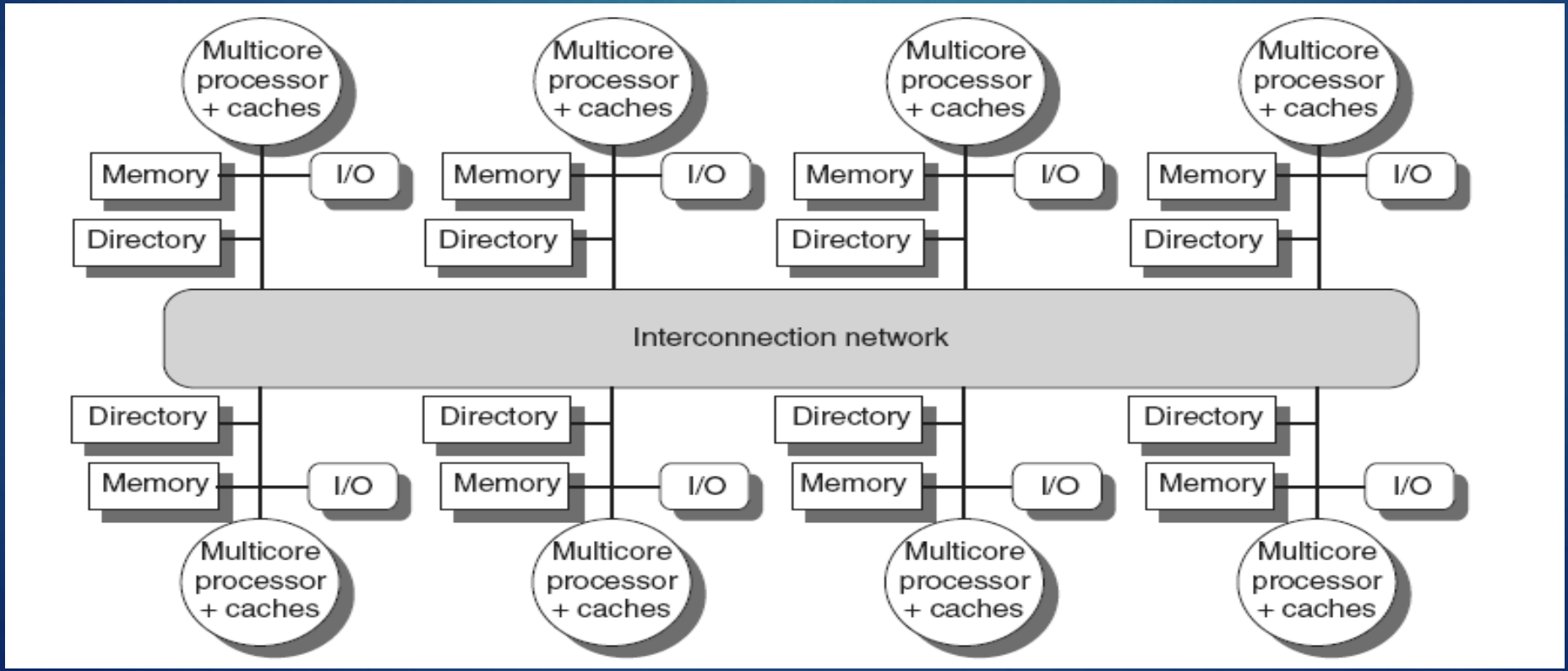
- Write Broadcast Protocol:
  - Write to shared data: broadcast on bus, processors snoop, and update copies
  - Read miss: memory is always up-to-date
- Write serialization: bus serializes requests
  - Bus is single point of arbitration

SNOOPING PROTOCOL

- Advantage
  - Fast
  - All request/response can seen by all processors
- Disadvantage
  - Broadcast the value to all over the bus
  - Not scalable

DIRECTORY PROTOCOL

- Add the directory into each memory node
- Use the vector bit to keep track of
- Sharing status of each block is kept in directory



DIRECTORY PROTOCOL

- Shared: One or more processors have the block cached, and the value in memory is up to date
- Uncached: No processor has a copy of the cache block.
- Modified: Exactly one processor has a copy of the cache block, and it has written the block, so the memory copy is out of date. The processor is called the owner of the block.

DIRECTORY PROTOCOL -TERMS

- Local node: the node where a request originates
- Home node: the node where the memory location of an address resides
- Remote node: the node that has a copy of a cache block, whether exclusive or shared

SPIN LOCK

- Locks that a processor continuously tries to acquire, spinning around a loop until it succeeds
- Used when programmers expect the lock to be held for a very short amount of time and when they want the process of locking to be low latency when the lock is available

SPIN LOCK - NO COHERENCE

- Store the lock variable in memory

```
        DADDUI   R2, R0. #1
lockit  EXCH     R2,0(R1)    atomic exchange
        BNEZ     R2,lockit   already lock?
```

SPIN LOCK – SUPPORT COHERENCE


```
lockit   LD      R2,0(R1)    ;load of lock
         BNEZ    R2,lockit   ;not-available spin
         DADDUI  R2,R0,#1    ;load locked value
         EXCH    R2,0(R1)    ;swap
         BNEZ    R2, lockit  ;branch if lock
                             wasn't 0
```

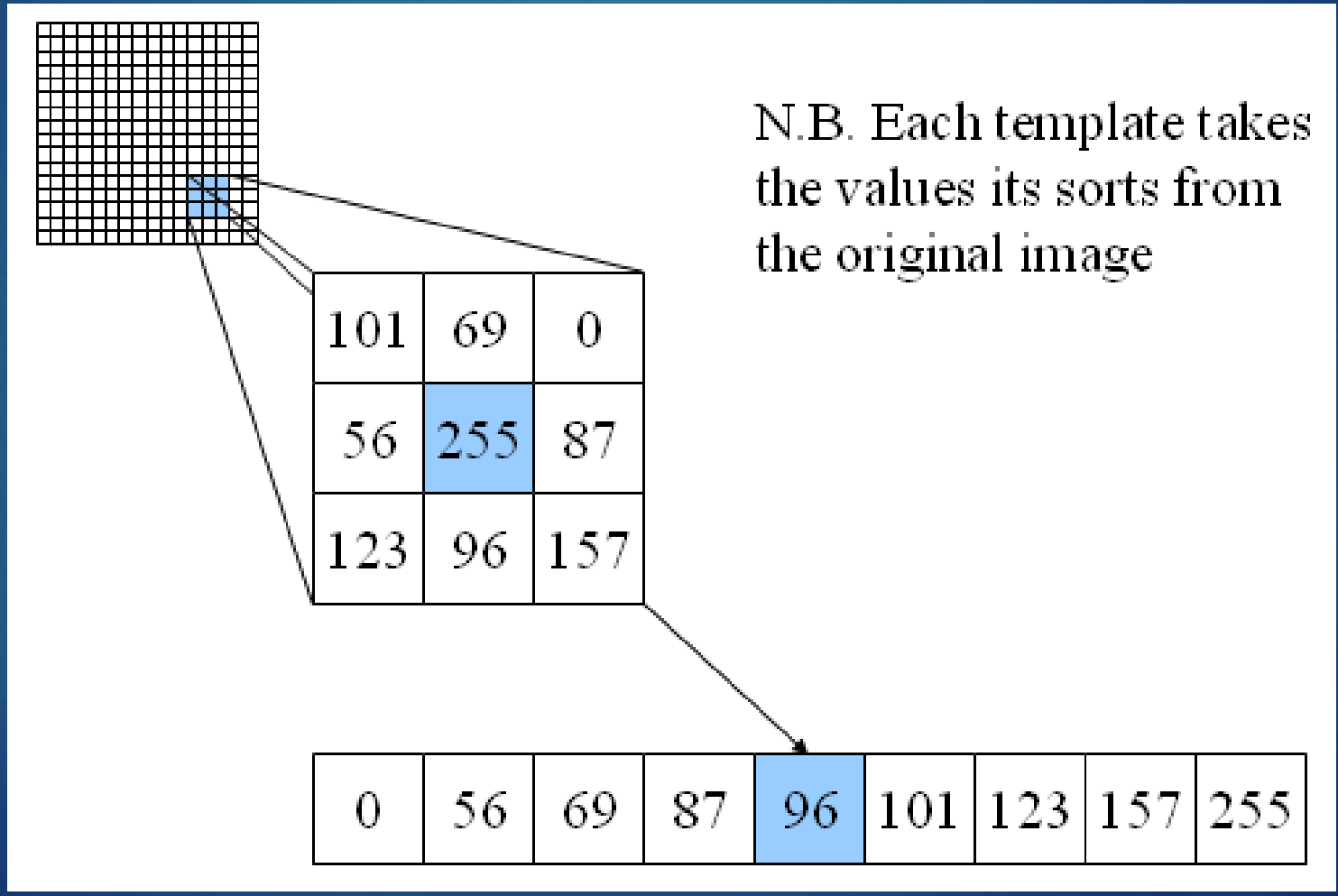
IMPLEMENT MEDIAN FILTER USING JAVA THREAD

MEDIAN FILTER

- What ?
  - Nonlinear process useful in reducing impulsive or salt-and-pepper noise
- How ?
  - Run through the signal, entry by entry, replacing each entry with the median of neighboring entries.
  - The pattern of neighbors is called the "window" ➡ slides, entry by entry, over the entire signal.
  - For 1D signals, the most obvious window is just the first few preceding and following entries.
  - For 2D (or higher-dimensional) signals such as images, more complex window patterns are possible (such as "box" or "cross" patterns).

MEDIAN FILTER

- How ? (cont)



MEDIAN FILTER: EXPERIMENTS

- Tool:
  - Profiling tools: YourKit Java profiler 2015 build 15050
  - IDE: Eclipse 4.4.2 (Luna)
  - Language: Java – Multitasking (JDK 1.8.0_40)

MEDIAN FILTER: EXPERIMENTS ON PC

- Testing system

Device name	Dell Vostro 3560
Operating system	Windows 8 Enterprise
Memory	8 GB
Processor	Intel® Core™ i5-3230M CPU @ 2.60GHz x 4
Physical cores	2
Logical cores	4 (hyperthreading technology)
Graphic	Intel® Ivybridge Mobile
Image	800 * 600

MEDIAN FILTER: EXPERIMENTS ON PC

Testing cases

Median Filter's Runtime for number of runs is constant and is 1

NO.	Threads	Runs	Window Size	Runtime
1	1	1	3	406
2	2	1	3	220
3	3	1	3	183
4	4	1	3	151
5	5	1	3	184
6	6	1	3	172
7	7	1	3	156

Median Filter's Runtime for number of threads is constant and is 1

NO.	Threads	Runs	Window Size	Runtime
1	1	6	3	1720
2	1	6	5	5610
3	1	6	7	13338
4	1	6	9	24255
5	1	6	11	41029
6	1	6	13	61542
7	1	6	15	89480

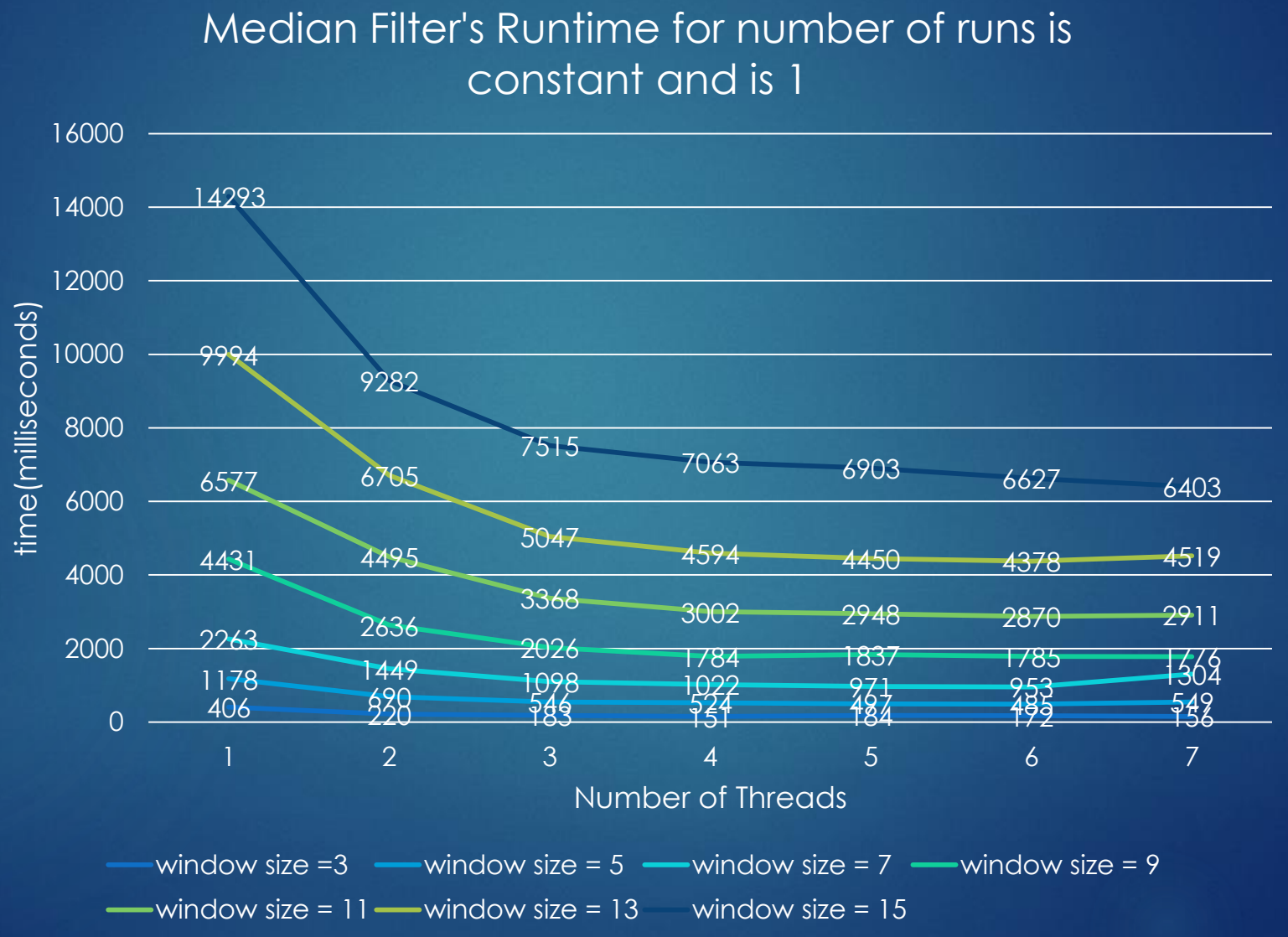
MEDIAN FILTER: EXPERIMENTS ON PC

Testing cases

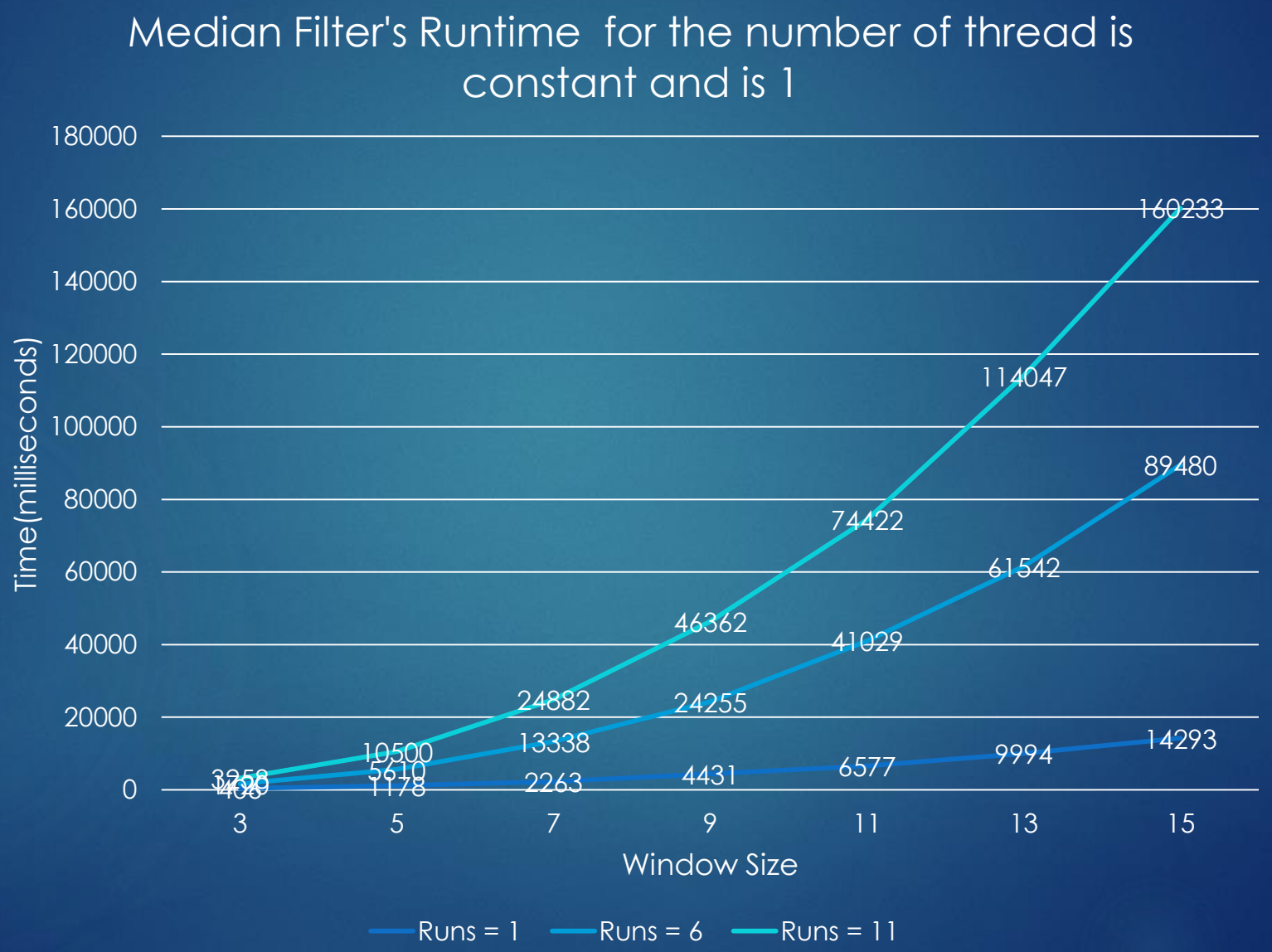
Median filter's runtime for the window size is constant and is 5

NO.	Threads	Runs	Window Size	Runtime
1	1	1	5	1178
2	2	1	5	690
3	3	1	5	546
4	4	1	5	524
5	5	1	5	497
6	6	1	5	485
7	7	1	5	549

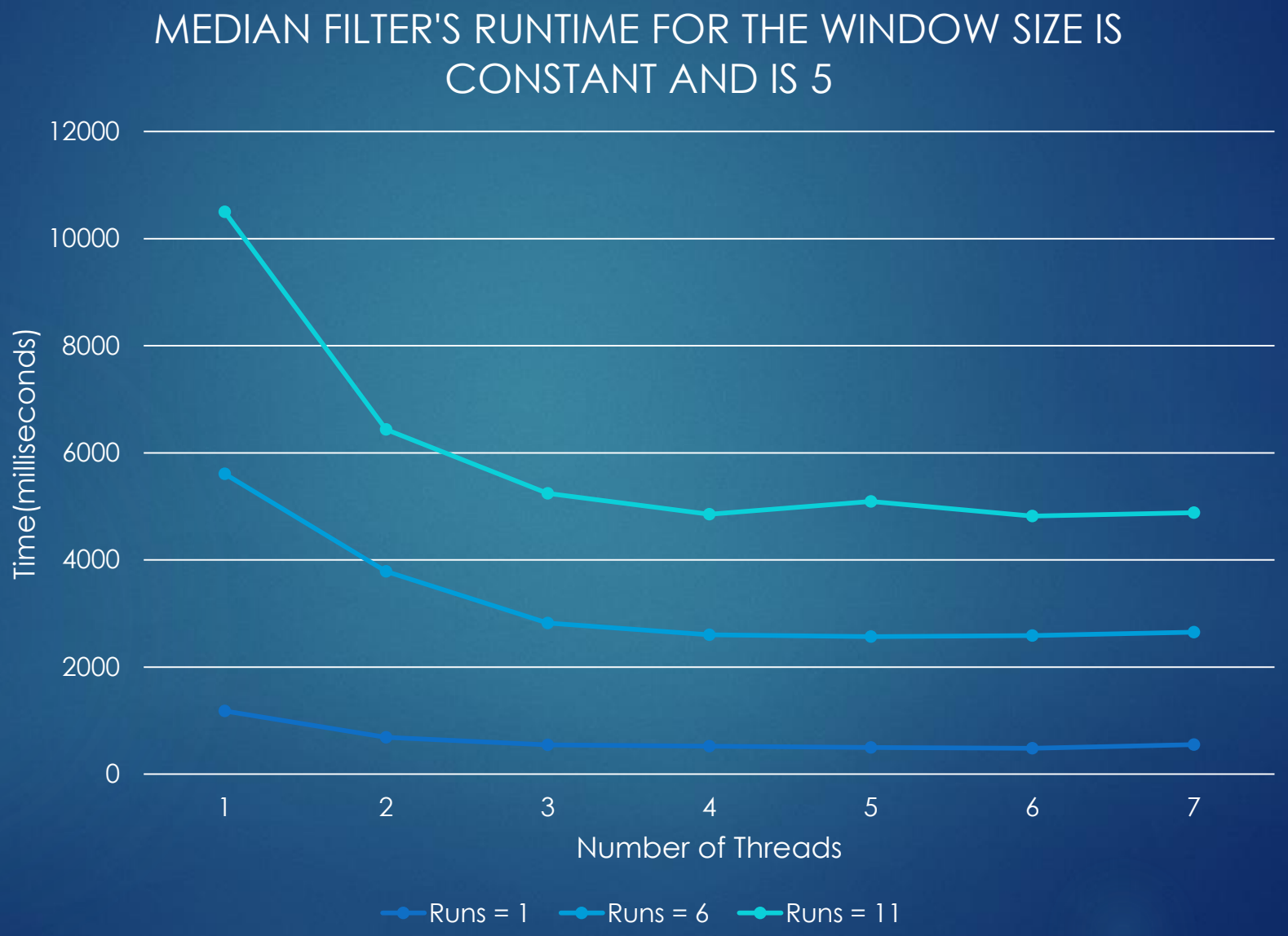
MEDIAN FILTER: EXPERIMENTS ON PC



MEDIAN FILTER: EXPERIMENTS ON PC



MEDIAN FILTER: EXPERIMENTS ON PC



MEDIAN FILTER: EXPERIMENTS ON ANDROID DEVICE

- Testing system

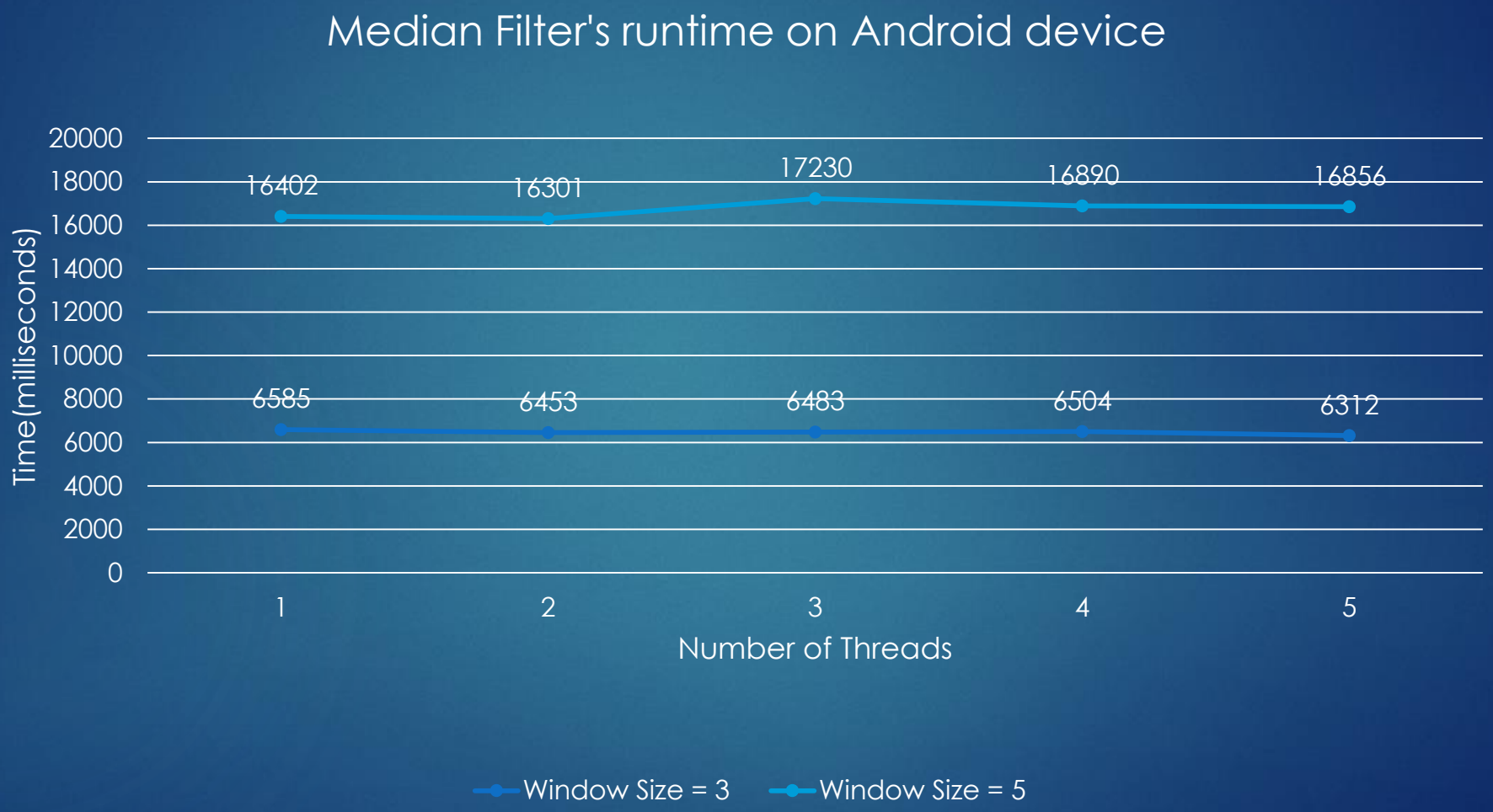
Device name	**LG - F160L (d1lu)**
Android version	4.1.2
API Level	16
Java VM	Dalvik 1.6.0
RAM	1809
SOC	Qualcomm Snapdragon S4 1.51 GHz
Model	LGE MSM8960 D1LKR
Architecture	Krait
Core	2
Process	28 nm
Image	800 * 600

MEDIAN FILTER: EXPERIMENTS ON ANDROID DEVICE

- Testing cases

NO.	Threads	Runs	Window Size	Runtime
1	1	1	3	6585
2	2	1	3	6453
3	3	1	3	6483
4	4	1	3	6504
5	5	1	3	6312
6	1	1	5	16402
7	2	1	5	16301
8	3	1	5	17230
9	4	1	5	16890
10	5	1	5	16856

MEDIAN FILTER: EXPERIMENTS ON ANDROID DEVICE



DISCUSSION ON THE RESULT

- Expectation
  - The program running on Android device will be slower than on PC
  - The runtime decreases respectively for the program with no. of threads that are smaller than no. of processors of system.

DISCUSSION ON THE RESULT

- Result on Android device
  - The runtime Is the same with different no. of thread
  - Much higher than PC
  - Exist bugs
- Result on PC
  - The runtime of 4 threads is lowest and of 1 thread is highest.
  - 1 thread ➡ 2 threads: decreases sharply
  - 2 threads ➡ 4 threads: decreases slightly
  - Rises slowly when runs at 5 threads.
  - Then, it fluctuates for the rest.
  - The runtime increases linearly with the number of runs.
  - The Runtime increases as the window size increases

DISCUSSION ON THE RESULT

Original image

DISCUSSION ON THE RESULT

9x9

DISCUSSION ON THE RESULT

15x15

DISCUSSION ON THE RESULT

- The program runs fastest with 4 threads because the System has the CPU with 4 cores, so there are maximum 4 threads can run simultaneously. The program run with the higher number of threads is slower because with the number of threads higher than 4, the program has to spend time to create threads, manage and switch between them while with 4 threads, the program does not need to switch between threads. In addition, the program with many threads will consume much amount of memory

Thank you for your listening!

Question & Answer

References

Computer Architecture - A Quantitative Approach - 4th & 5th editions

(John L.Hennessy & David A.Patterson)