McC++/Java: Enabling Multi-core Based Monitoring and Fault Tolerance in C++/Java

Lu YANG, Liqian YU, Jianwen TANG, Linzhang WANG, Jianhua ZHAO, Xuandong LI

State Key Laboratory of Novel Software Technology &
Department of Computer Science and Technology
Nanjing University

Agenda

- Motivation
- Preliminary Principles of Multi-core Based Monitoring and Fault Tolerance
- Approach for Multi-core Based Monitoring and Fault Tolerance in C++/Java
- Case Study
- Discussion
- Conclusion

Agenda

- **Motivation**
- Preliminary Principles of Multi-core Based Monitoring and Fault Tolerance
- Approach for Multi-core Based Monitoring and Fault Tolerance in C++/Java
- Case Study
- Discussion
- Conclusion

Motivation

- Reliable software system design approaches, e.g. monitoring and fault tolerance, may become common design choices.
- The multi-core architecture is a suitable platform to support reliable software system design: the advantage of the parallel performance and prevalence.
- For allowing software developers to handle programming tasks on multi-core platforms more efficiently, we propose an approach for enabling monitoring and fault tolerance in C++/Java programs on multi-core platforms.

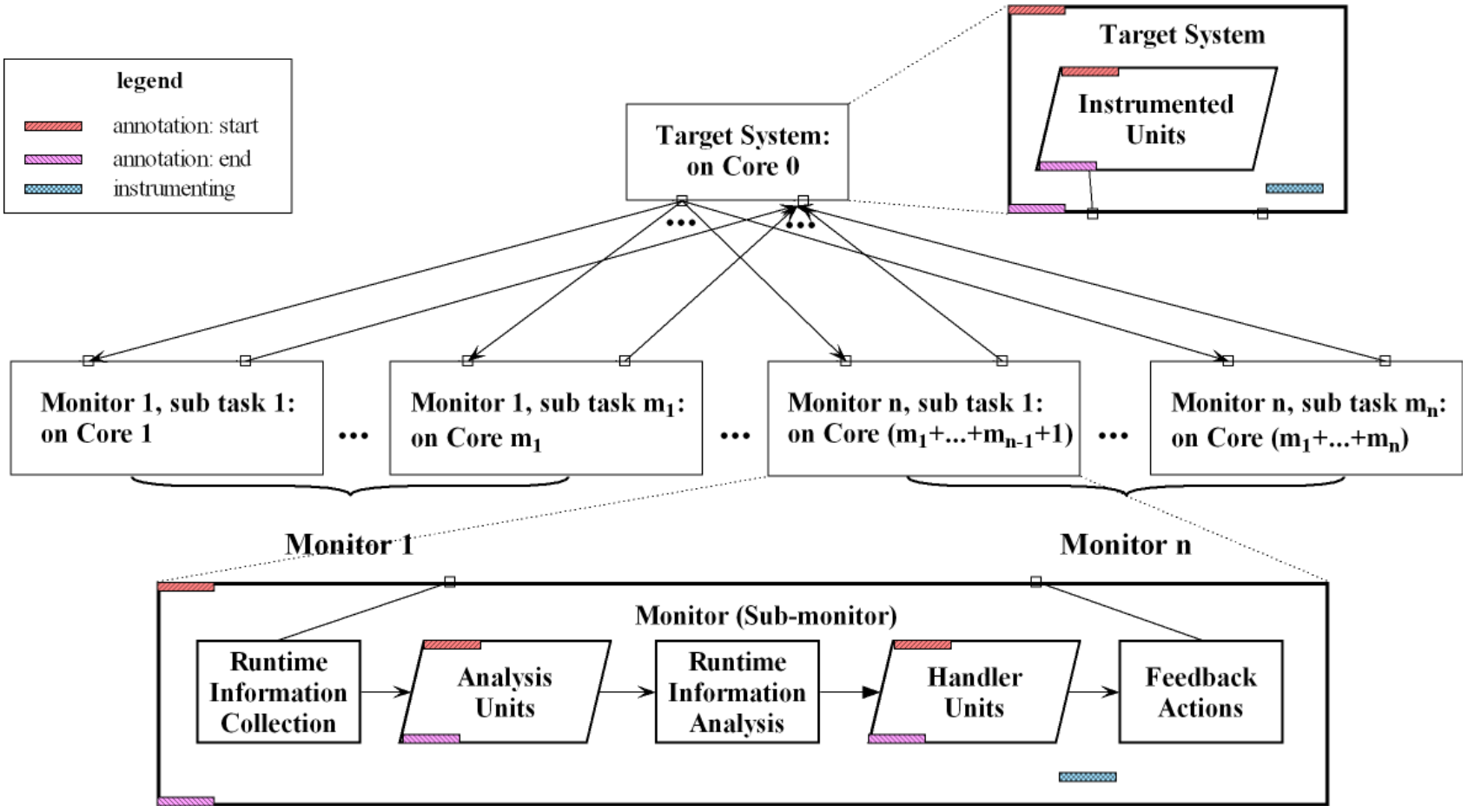
Agenda

- Motivation
- **Preliminary Principles of Multi-core Based Monitoring and Fault Tolerance**
- Approach for Multi-core Based Monitoring and Fault Tolerance in C++/Java
- Case Study
- Discussion
- Conclusion

Principles of Multi-core Based Monitoring

- Software monitoring uses monitors to check the behavior of a target system, and influences the behavior of the system when monitors confirm that the given properties are either satisfied or falsified.
- In general, the monitoring design includes three steps: instrumenting, monitoring and handling.
- optimize the monitoring design on multi-core platforms:
  - executing monitors for different properties in parallel on different cores
  - decomposing a monitor task into several sub-tasks running in parallel on different cores

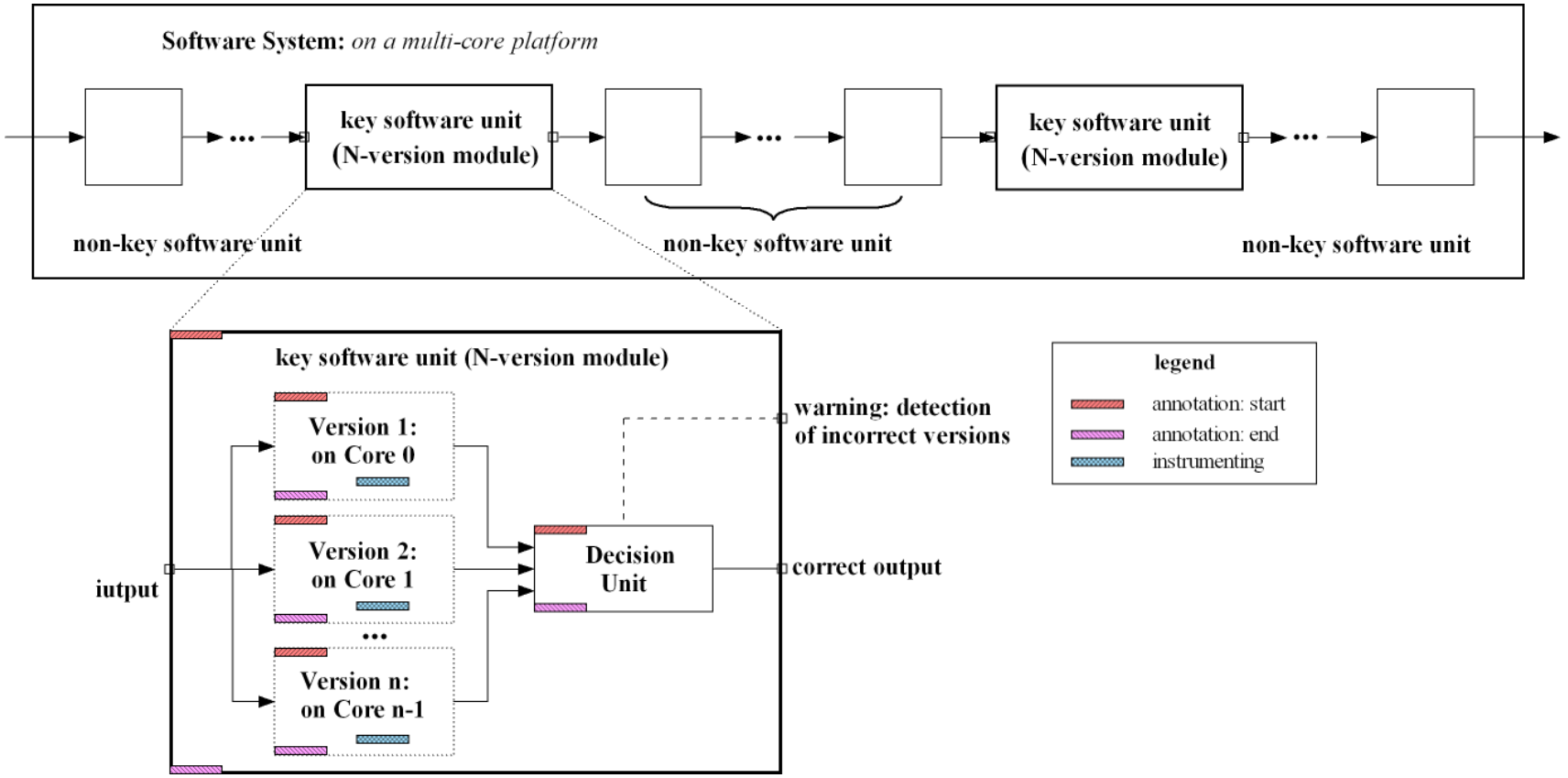
Principles of Multi-core Based Monitoring



Principles of Multi-core Based Fault Tolerance

- Software fault tolerance is the ability for the software to detect and recover from failures of the system in order to ensure that the system performs as specified.
- N-version programming: develop N separate versions with equivalent functionalities only for some key software units of the system instead of the whole system. Each version is developed independently by an isolated group to prevent identical faults among versions.
- With the multi-core architecture, redundant versions of a key software unit can run in parallel on different cores to improve the performance.

Principles of Multi-core Based Fault Tolerance



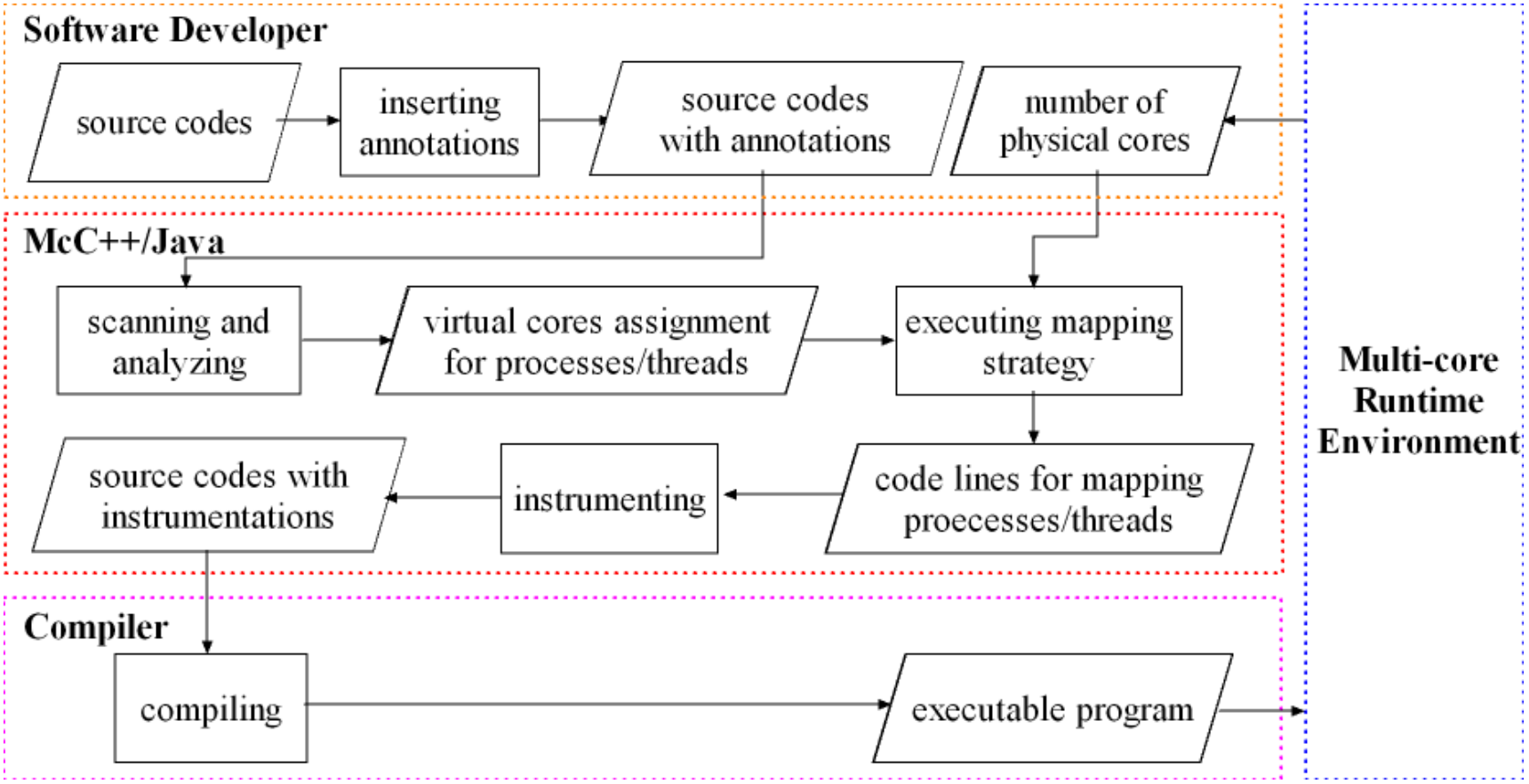
Agenda

- Motivation
- Preliminary Principles of Multi-core Based Monitoring and Fault Tolerance
- **Approach for Multi-core Based Monitoring and Fault Tolerance in C++/Java**
- Case Study
- Discussion
- Conclusion

Enabling Multi-core Based Monitoring and Fault Tolerance in C++/Java

- A monitoring or fault tolerance task can be designed and implemented into several monitoring or fault tolerance modules. All these modules are implemented as separate processes/threads.
- We introduce a group of special annotations for software developers to specify a simple and virtual multi-core based design in a high abstraction level.
- According to these annotations, an automatic and convenient mapping to a given multi-core platform is established via a prototype tool McC++/Java.

Workflow of McC++/Java



Annotation Based Virtual Design

- provide a simple and virtual multi-core based design by using annotations to be instrumented into source codes in a high abstraction level, which can:
  - help software developers understand and complete the structure of monitoring and fault tolerance programs
  - let software developers determine the assignment of required virtual cores for all monitoring and fault tolerance modules
  - help McC++/Java find the locations to instrument code lines for mapping the processes/threads for these modules to suitable physical cores

Annotation Based Virtual Design

```
1 /*@ start target system which needs v virtual cores @*/
2   ...
3   /*@ start instrumented unit k used by monitor i @*/
4     ...
5   /*@ end instrumented unit k used by monitor i @*/
6   ...
7 /*@ end target system @*/
8 ...
9 /*@ start creating process for target system @*/ (/*@ start creating thread
        for target system @*/)
10  ...
11 /*@ end creating process for target system @*/ (/*@ end creating thread for
        target system @*/)
12 ...
13
14 /*@ start monitor i which needs v virtual cores @*/
15  ...
16  /*@ start analysis unit of monitor i @*/
17    ...
18  /*@ end analysis unit of monitor i @*/
19  ...
20  /*@ start handler unit of monitor i @*/
21    /*@ start success condition @*/
22      ...
23    /*@ end success condition @*/
24    /*@ start validation handler @*/
25      ...
26    /*@ end validation handler @*/
27    /*@ start failure condition @*/
28      ...
29    /*@ end failure condition @*/
30    /*@ start violation handler @*/
31      ...
32    /*@ end violation handler @*/
33  /*@ end handler unit of monitor i @*/
34  ...
35 /*@ end monitor i @*/
36 ...
37 /*@ start creating process for monitor i @*/ (/*@ start creating thread for
        monitor i @*/)
38 ...
39 /*@ end creating process for monitor i @*/ (/*@ end creating thread for
        monitor i @*/)
```

Annotations for Monitoring (C++)

```
1 /*@ start key unit i @*/
2   ...
3   /*@ start version j of key unit i which needs v virtual cores @*/
4     ...
5   /*@ end version j of key unit i @*/
6   ...
7   /*@ start creating thread for version j of key unit i with dll name
        dllName @*/
8     ...
9   /*@ end creating thread for version j of key unit i @*/
10  ...
11  /*@ start decision unit of key unit i @*/
12    ...
13  /*@ end decision unit of key unit i @*/
14  ...
15 /*@ end key unit i @*/
```

Annotations for Fault Tolerance (Java)

Automatic Mapping to Multi-core Platforms

- McC++/Java can help software developers assign a group of physical cores to all monitoring and fault tolerance modules.
- The physical cores assignment is in proportion to the virtual cores assignment.
- three steps:
  - scanning source codes with annotations, and getting the virtual cores assignment information
  - executing mapping strategy
  - instrumenting and mapping to multi-core runtime platforms
    - instrument rules for C++
    - instrument rules for Java

Agenda

- Motivation
- Preliminary Principles of Multi-core Based Monitoring and Fault Tolerance
- Approach for Multi-core Based Monitoring and Fault Tolerance in C++/Java
- **Case Study**
- Discussion
- Conclusion

Case Study

- case study: monitoring error pattern Array Index Out of Bound in C++ programs (VideoNet) on a multi-core platform
- Multi-Process Based Monitoring Design
  - the monitoring task is decomposed into 3 sub-monitors: implemented as separate processes in parallel
- Annotation Based Virtual Design
  - a target system with 92 instrumented units and 3 sub-monitors with analysis units and handler units
  - The annotations for monitoring tasks in C++ programs are inserted into the source codes.

Case Study

- Source Codes Mapping to Runtime Platforms
  - use McC++/Java to transform the source codes with annotations to the source codes with instrumentations

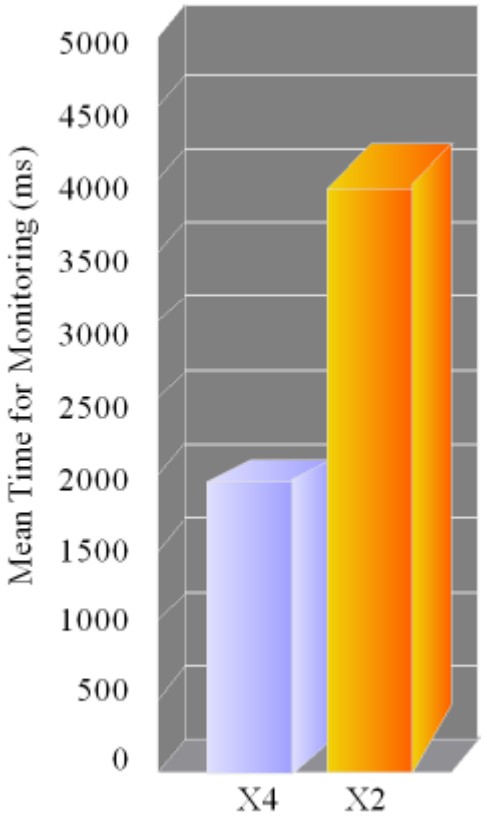
```
Scanning ....
Target System   Need Cores: 1  Affinity Mask: 1  index: 0
Monitor 1       Need Cores: 1  Affinity Mask: 2  index: 1
Monitor 2       Need Cores: 1  Affinity Mask: 4  index: 2
Monitor 3       Need Cores: 1  Affinity Mask: 8  index: 3

Converting ....
Convert Successfully
```

```
/*@ start mapping for target system @*/
   SetProcessAffinityMask(hProcess, 1);
/*@ end mapping for target system @*/
```

Case Study

- Experiment Design and Results Analysis
  - experiment platform: quad-core
  - experiment design: maps the target system and sub-monitors to 2/4 cores respectively to show the improvement of the performance
  - results analysis
    - 4 cores: 1947.2 ms
    - 2 cores: 4089.9 ms



Agenda

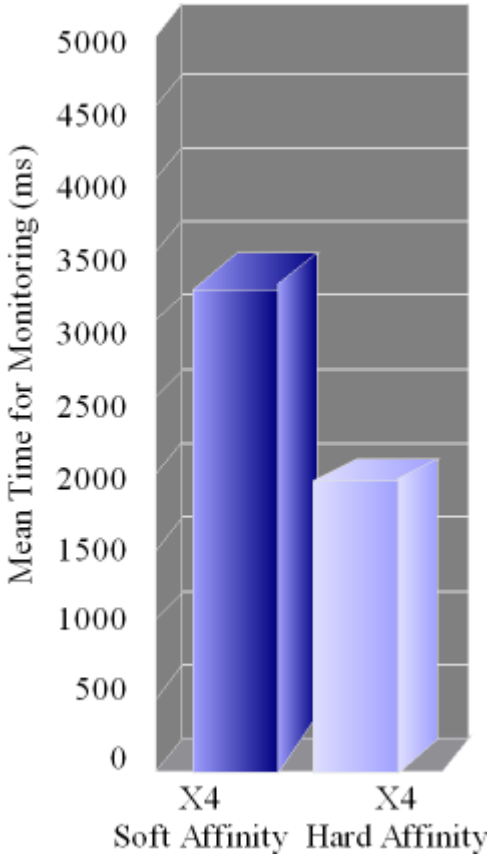
- Motivation
- Preliminary Principles of Multi-core Based Monitoring and Fault Tolerance
- Approach for Multi-core Based Monitoring and Fault Tolerance in C++/Java
- Case Study
- **Discussion**
- Conclusion

Discussion: soft affinity or hard affinity?

- Affinity
  - soft affinity: let the operating system schedule processes/threads
  - hard affinity: software developers explicitly specify a core (or a group of cores) for a process/thread to run on
  - hard affinity or soft affinity: an application specific problem
- scenarios suitable for hard affinity
  - long-running time-sensitive applications
  - applications in scientific and academic computing area

Discussion: soft affinity or hard affinity?

- Experiment Design and Results Analysis
  - experiment design:
    soft affinity vs hard affinity on 4 cores, to compare the efficiency of soft affinity and hard affinity
  - results analysis: mean time
    - hard affinity: 1947.2 ms
    - soft affinity: 3261.9 ms



Agenda

- Motivation
- Preliminary Principles of Multi-core Based Monitoring and Fault Tolerance
- Approach for Multi-core Based Monitoring and Fault Tolerance in C++/Java
- Case Study
- Discussion
- **Conclusion**

Conclusion

- an approach for enabling multi-core based monitoring and fault tolerance in C++/Java
- a tool McC++/Java
- two case studies on multi-core platforms

- future work
  - more platforms
  - more programming languages

Thanks! & Questions?