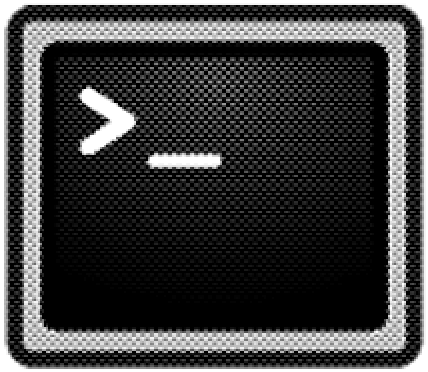
Exception Handling with OpenMP in Object-Oriented Languages

Xing Fan, Mostafa Mehrabi, Oliver Sinnen, Nasser Giacaman
Department of Electrical and Computer Engineering
The University of Auckland
Auckland, New Zealand

Background

- Using OpenMP in high-performance multi-core servers.

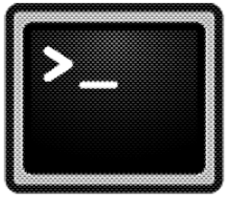
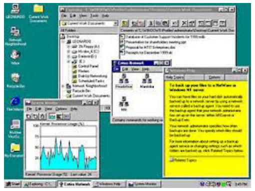


Motivation

- Using OpenMP in a wider range of multi-core devices.




Time


Scientific
batch-like

Visible
GUI

Responsive
Smooth

User-friendly
Interactive

Smart
Everywhere

Motivation

- Using OpenMP in high-level languages.

Procedural languages	**Object-oriented languages**
C, Fortran, Pascal	C++, C#, Java
Low-level semantic abstraction • Primitive operations • Function/procedure • No special error recovery support • Integer based for-loop	**High-level semantic abstraction** • Polymorphism • Operator overloading • Exception handling • For-each iteration
Low-level data abstraction • Primitive data types • Structures/Unions	**High-level data abstraction** • User-defined data type/class • Inheritance

Motivation

- Using OpenMP in high-level languages.

Procedural languages	**Object-oriented languages**
C, Fortran, Pascal	C++, C#, Java
Low-level semantic abstraction • Primitive operations • Function/procedure • No special error recovery support • Integer based for-loop	**High-level semantic abstraction** • Polymorphism • Operator overloading • Exception handling • For-each iteration
Low-level data abstraction • Primitive data types • Structures/Unions	**High-level data abstraction** • User-defined data type/class • Inheritance
Standard OpenMP	Extended OpenMP (Pyjama)

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

Pyjama

- An OpenMP implementation for Java.
- Aim for an easier parallelisation for Java programs, especially for Java interactive applications.
- Can be used for Android apps development.
- Concerns for software developing principles: programming productivity, usability, robustness, etc.

Java
OpenMP

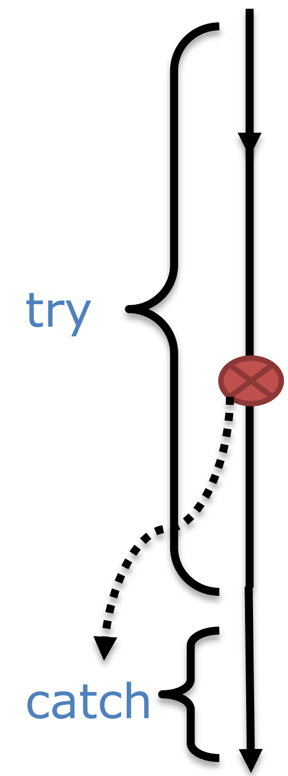
Why exception handling is important in OO?

- Language-level semantic support for error recovery, providing clean and self-evident control flow.
- A high level abstraction of errors. An exception object is able to contain rich information about an error.
- Conform with software engineering principles- Being friendly to encapsulation, inheritance, polymorphism, etc.

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

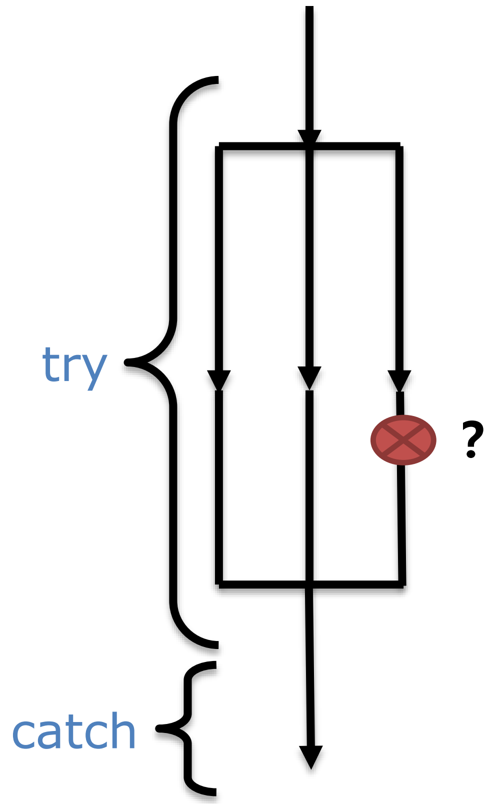
Sequential exception handling

```
try {

    for (int i=0; i<fileNames.size(); i++) {
        Image img = load(fileNamses[i]);
        Set<KeyPoint> kp = extract(img);
        kps.union(kp);
    }
} catch(Exception e){
    //handle other exception
}
```



Parallel exception handling

```
try {
    #pragma omp parallel for
    for (int i=0; i<fileNames.size(); i++) {
        Image img = load(fileNamses[i]);
        Set<KeyPoint> kp = extract(img);
        kps.union(kp);
    }
} catch(Exception e){
    //handle other exception
}
```



Parallel exception handling

```
try {
    #pragma omp parallel
    {
        may_cause_exception();
    }
} catch(Exception e){
    //handling exception
}
```

Global exception handling- we can't

```
fxin927@UOA323534:~/temp$ g++ -fopenmp -o openmp_exception_test ./openmp_exception.cpp
fxin927@UOA323534:~/temp$ ./openmp_exception_test
iterate 0 in thread 0
iterate 1 in thread 0
iterate 2 in thread 0
iterate 6 in thread 2
iterate 7 in thread 2
terminate called after throwing an instance of 'int'
iterate 3 in thread 1
iterate 4 in thread 1
iterate 5 in thread 1
Aborted (core dumped)
fxin927@UOA323534:~/temp$
```

Parallel exception handling

```
#pragma omp parallel
{
   try {
      phase1_may_cause_exception();

      phase2();
   } catch(Exception e) {
    //handling exception
   }
}
```

Local exception handling- Yes we can

Parallel exception handling

```
#pragma omp parallel
{
   try {
      phase1_may_cause_exception();
     #pragma omp barrier
      phase2();
   } catch(Exception e) {
    //handling exception
   }
}
```

Local exception handling- Yes we can

Wait- May cause dead lock!

Parallel exception handling -Problems

```
try {
    #pragma omp parallel for
    for (int i=0; i<4; i++){
        may_cause_exception();
    }
} catch(Exception e){
    //handling exception
}
```

```
#pragma omp parallel
{
    try {
        phase1_may_cause_exception();
       #pragma omp barrier
        phase2();
    } catch(Exception e) {
     //handling exception
    }
}
```

Try-catch mechanism that does not **syntactically** and **semantically** conform with the OpenMP specification

Syntactically conforms with OpenMP specification, but **semantically** it has a defect

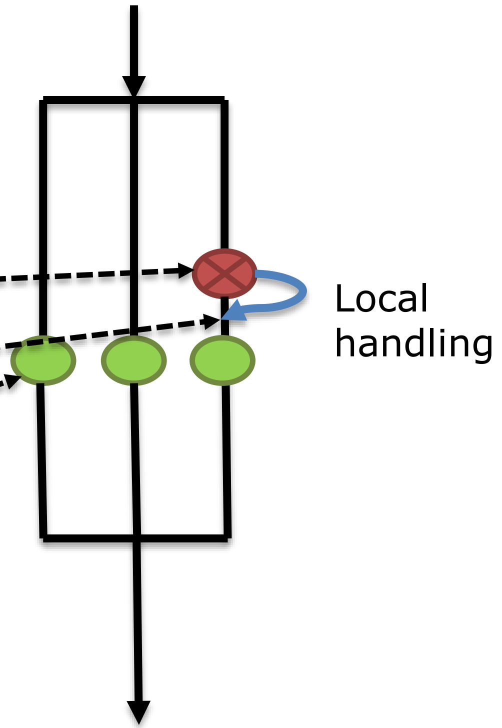
Definitions

- Local exception handling: An exception happened inside the parallel region, then it is handled by the same thread which threw the exception within the parallel thread group.

- Global exception handling: An uncaught exception escapes from the parallel region, which could influence the entire parallel processing. Handling this type of exception is called global exception handling.

Stronger exception handling support

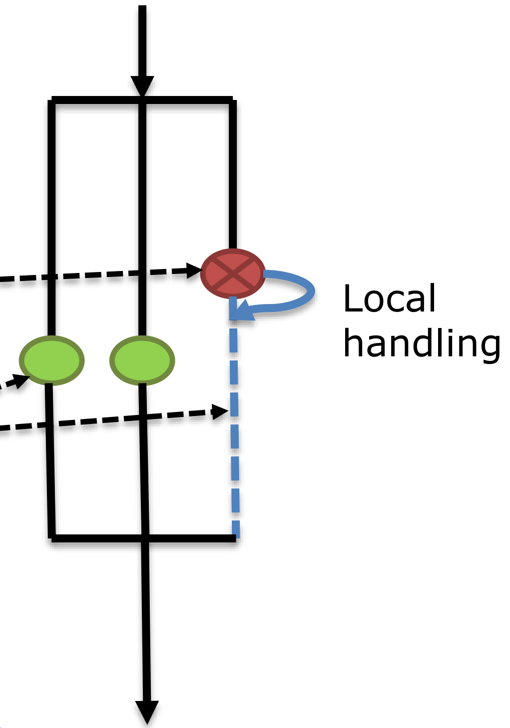
```
#pragma omp parallel
{
    try {
        exception_happens();
    } catch(Exception e){
        //handling local exception

    }
    #pragma omp barrier
    stage2();
}
```



Conform with current standard.(Global exception handling will never happen)

Stronger exception handling support

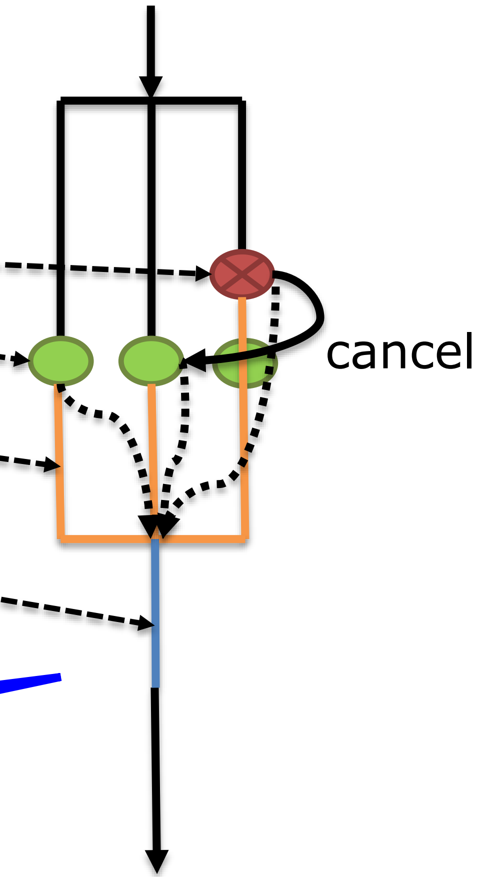
```
try {
    #pragma omp parallel
    {
        try {
            exception_happens();
        } catch(Exception e){
            //handling local exception
#pragma omp cancel parallel local
        }
        #pragma omp barrier
        stage2();
    }
} catch(Exception e){
    //handling global exception
}
```



Boosted thread control:
Enable one thread cancel locally without entire parallel processing stopping.

Stronger exception handling support

```
try {
    #pragma omp parallel
    {
        exception_happens();
        #pragma omp barrier
        stage2();
    }
} catch(Exception e){
    //handling exception
}
```



Boosted semantic: Global handling.
Boosted runtime: Uncaught exception inside parallel region stops the entire parallel processing.

Stronger exception handling support

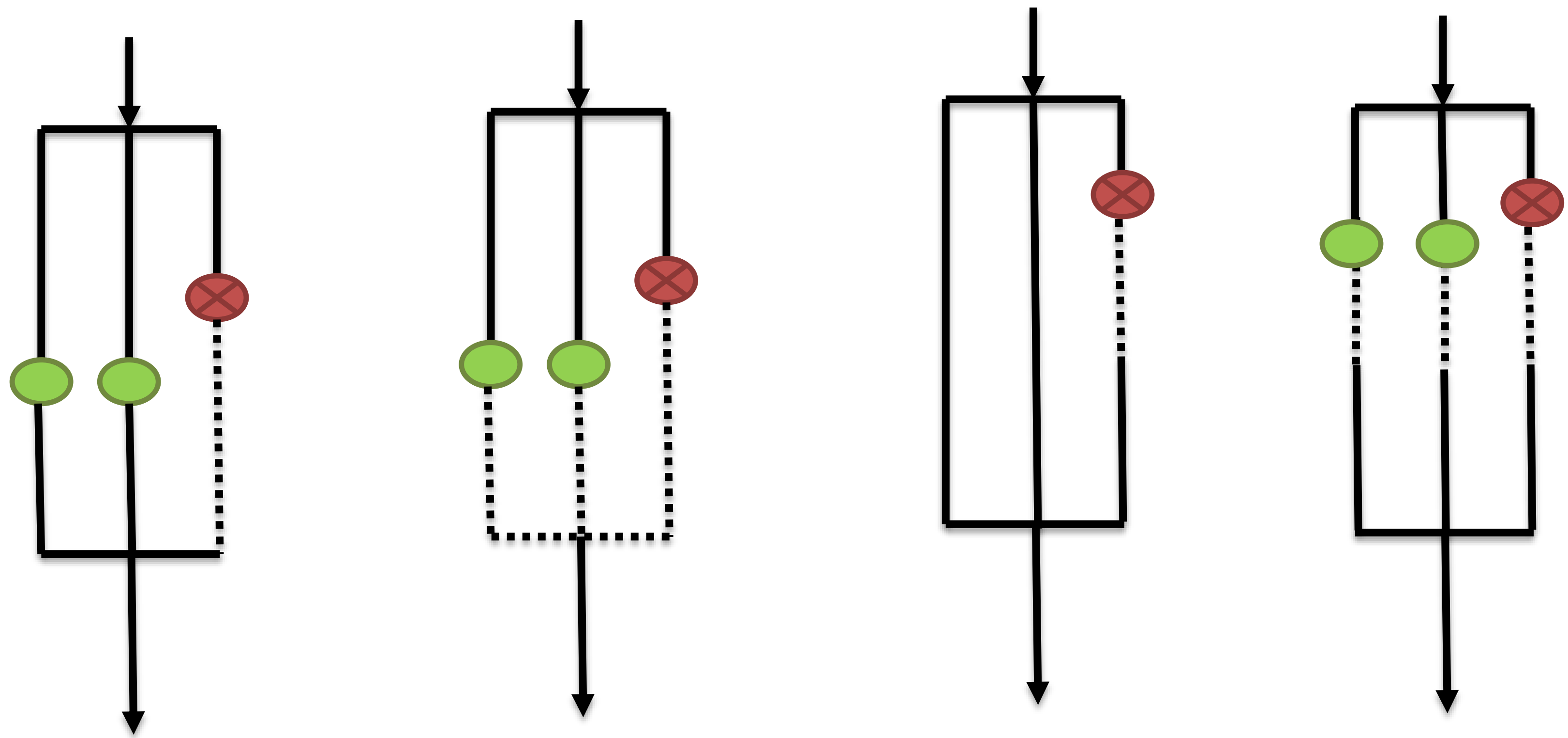
```
try {
    #pragma omp parallel
    {
        try {
            exception_happens();
        ⚠ #pragma omp barrier
            stage2();
        } catch(Exception e){
            //handling local exception
    }
} catch(Exception e){
    //handling global exception
}
```

Boosted compilation checking: The code which contains potential defect will trigger the compiler's warning.

Extended cancellation directive

#pragma omp cancel
construct-type-clause thread-affiliate-clause [if-clause]

Where *construct-type-clause* is one of the following:
parallel, sections, for, taskgroup
and *thread-affiliate-clause* is one of the following:
global, local
and *if-clause is*: if(scalar-expression)



parallel local

parallel global

for/section local

for/section global

Cancellation triggering point

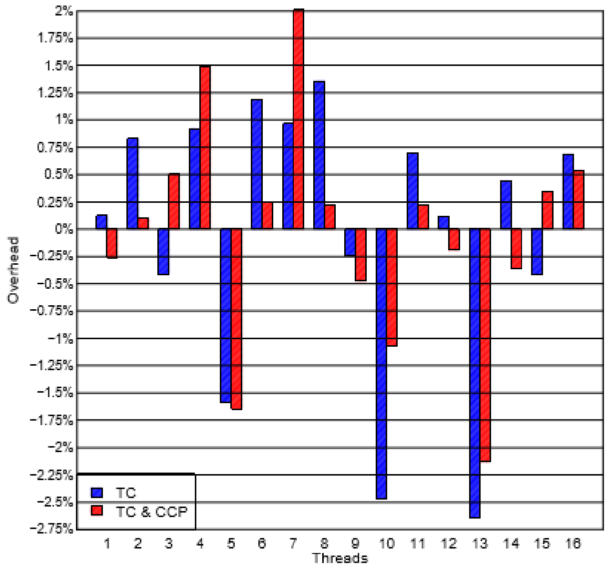
Cancellation checking point

What is boosted?

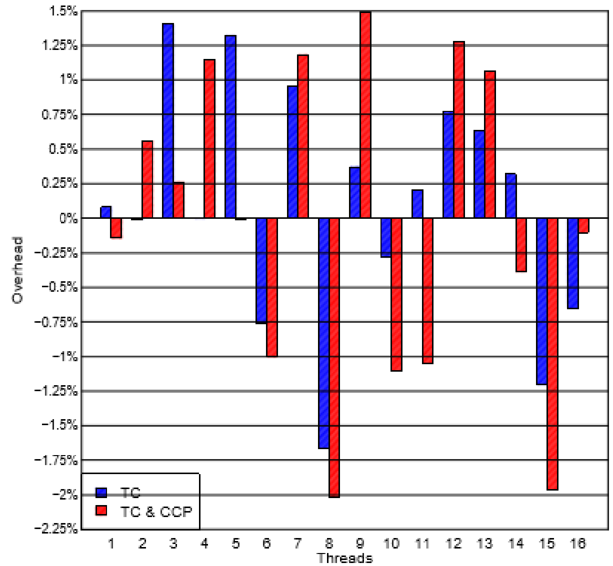
- A compilation stage semantic checking, warning programmers if a local exception handling could cause extra synchronization problems.
- Stop the parallel processing when an uncaught local exception is escaped from the parallel region, in default.
- Extended directives for flexible thread stopping/resuming, for various purposes of programming logic.

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

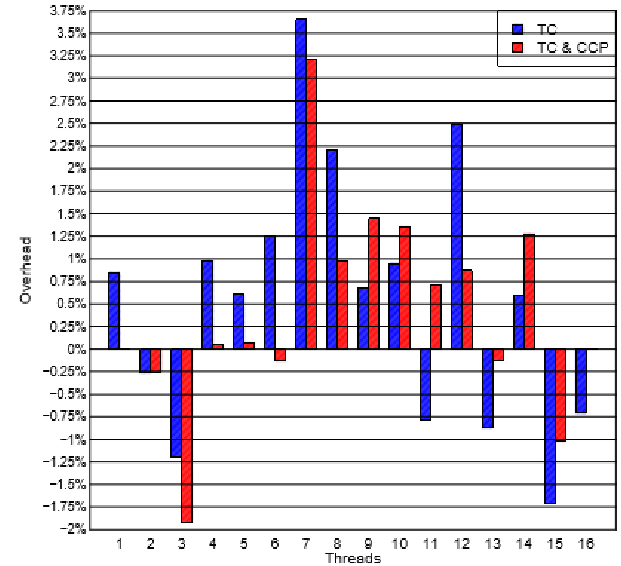
Overhead is negligible



omp for



omp parallel



omp barrier

Using original runtime as the baseline, we compare the overhead of exception handling boosted runtime, and find the boosted OpenMP runtime does not show a noticeable overhead compared with non-modified one.

Concluding remarks

- OpenMP will embrace a wider range of parallel applications, running on various multi-core devices.
- A coexist of OpenMP semantics and other high-level language abstraction concepts requires further explorations.
- From the software engineering point of view, robustness, usability, maintainability etc. could be more important than the executing performance of some programs.
- We are eager for a better speedup, but it is not always the whole story.

THANK YOU

Q & A