Parallel Task for parallelising OO desktop applications

Nasser Giacaman and Oliver Sinnen

PDSEC, IPDPS-10, Atlanta, USA, April 2010
Overview

- Motivation
- Structure of desktop applications
- Parallel Task (ParaTask)
- Implementation
- Performance
- Conclusions
The need for desktop parallelisation

- Desktop systems becoming parallel
- Desktop software MUST be parallel
- Not as easy as “embarrassingly parallel” problems
- Desktop applications: OO & GUI
Graphical User Interfaces (GUI)
Structure of desktop applications
Structure of desktop applications

Event queue
Structure of desktop applications

Event queue
Structure of desktop applications

- Event queue
- Event loop
Structure of desktop applications

Event queue

Event loop

Event handler (mouse click)

Event handler (data received)

Event handler (refresh GUI)
Structure of desktop applications

- Event queue
- Event loop
- Event handler (mouse click)
- Event handler (data received)
- Event handler (refresh GUI)
Structure of desktop applications

- Event queue
- Event loop
- Event handler (mouse click)
- Event handler (data received)
- Event handler (refresh GUI)
Structure of desktop applications

Event queue

Event loop

Event handler (mouse click)

Event handler (data received)

Event handler (refresh GUI)

GUI components
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

Event queue

Event loop

Event handler (mouse click)

Event handler (data received)

Event handler (refresh GUI)

GUI components
Structure of desktop applications

**GUI Thread, Event Dispatch Thread (EDT)**
- Event queue
- Event loop
- Event handler *(mouse click)*
- Event handler *(data received)*
- Event handler *(refresh GUI)*

**Helper Thread**
- Process long task

**GUI components**
- Message box
- Progress bar
- Warning icon
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

Event queue

Event loop

Event handler (mouse click)

Event handler (data received)

Event handler (refresh GUI)

Helper Thread

Process long task

GUI components
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

Event queue

Event loop

Event handler (mouse click)

Event handler (data received)

Event handler (refresh GUI)

GUI components

Helper Thread

Process long task
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

Event queue

Event loop

Event handler (mouse click)

Event handler (data received)

Event handler (refresh GUI)

Helper Thread

Process long task

NOT ALLOWED
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

- Event queue
- Event loop
- Event handler (mouse click)
- Event handler (data received)
- Event handler (refresh GUI)

Helper Thread

- Process long task

GUI components
Background: Task parallelism

class ImageApp {

    ...

    void task1(String f) { ... }
    void task2(String f) { ... }
    void task3(String f) { ... }
    void task4(String f1, f2) { ... }
    void display(String f) { ... }

}
Using threading library

class Task1 : Thread {
    ThreadTask1(String file) {...}
    ...
    void run() {
        // do task1
    }
}

class ImageApp {
    ...
    void task1(String f) {...}
    void task2(String f) { ... }
    void task3(String f) { ... }
    void task4(String f1, f2) { ... }
    void display(String f) { ... }
}
Using threading library

class Task4 : Thread {
    ThreadTask4(String file) {...}
    void run() {
        // do task1
    }
}
class Task3 : Thread {
    ThreadTask3(String file) {...}
    void run() {
        // do task1
    }
}
class Task2 : Thread {
    ThreadTask2(String file) {...}
    void run() {
        // do task1
    }
}
class Task1 : Thread {
    ThreadTask1(String file) {...}
    void run() {
        // do task1
    }
}
Using threading library

class Task4 : Thread {
    Condition waitFor2, waitFor3;
    ThreadTask4(String file) {...}
}

class Task3 : Thread {
    Condition waitFor1, notify4;
    ThreadTask3(String file) {...}
}

class Task2 : Thread {
    Condition waitFor1, notify4;
    ThreadTask2(String file) {...}
}

class Task1 : Thread {
    Condition notify2, notify3;
    ThreadTask1(String file) {...}
    
    void run() {
        // do task1
    }
}
Using threading library

```java
class Task4 : Thread {
    Condition waitFor2, waitFor3;
    ThreadTask4(String file) {...}
}

class Task3 : Thread {
    Condition waitFor1, notify4;
    ThreadTask3(String file) {...}
}

class Task2 : Thread {
    Condition waitFor1, notify4;
    ThreadTask2(String file) {...}
}

class Task1 : Thread {
    Condition notify2, notify3;
    ThreadTask1(String file) {...}
}

void run() {
    // do task1
    notify2.signal();
    notify3.signal();
}
```
Problems with using threading library

- Code restructuring
- Thread management
- Manage dependences
- Coupling between tasks
- Task completion
- Performance hit
ParaTask: Task declaration

```java
public class ImageApp {
    ...
    TASK public void task1(String f) {
        // user code
    }
}
```
ParaTask: Task invocation

List images = ...;
for (int i = 0; i < images.size(); i++)
{
    TaskID id = task1(images.at(i));
    ...
}
Additional features

1) Different task types
2) Task dependences
3) Task completion & return values
   - Blocking (i.e. “Futures”)
   - Non-blocking
4) Exception handling
Different task types

- One-off tasks
  - Task parallelism
- Multi-tasks
  - Data parallelism
- Interactive
  - Latency hiding
Multi-tasks

\[\text{TASK}(*\text{) public int multiTask()} \{\]
  ...
\}

\[\text{TASK}(*) \text{ int multiTask(){myID = 0;}}\]
  ...
\}

\[\text{TASK}(*) \text{ int multiTask(){myID = 1;}}\]
  ...
\}

\[\text{TASK}(*) \text{ int multiTask(){myID = 2;}}\]
  ...
\}

TaskID id = multiTask();
Interactive tasks

Task queue:

TASK void A() {
  ...
}

TASK(*) int B() {
  ...
}

INT_TASK int input() {
  // block
  ...
}

TASK void C() {
  ...
}

(quick)

(quick)

(quick)

- user-interactive
- web access
- blocking

Worker thread 1

Worker thread 2

Interactive thread
Additional features

1) Different task types: TASK(*), INTERACTIVE_TASK
2) Task dependences
3) Task completion & return values
   • Blocking (i.e. “Futures”)
   • Non-blocking
4) Exception handling
Task dependences

List images = ...;
for (int i = 0; i < images.size(); i++)
{
    TaskID id1 = task1(images.at(i));
    TaskID id2 = task2(images.at(i)) dependsOn(id1);
    TaskID id3 = task3(images.at(i)) dependsOn(id1);
    TaskID id4 = task4(images.at(i)) dependsOn(id2,id3);
    ...
}

1 2

3

4 display
Additional features

1) Different task types: TASK(*), INTERACTIVE_TASK
2) Task dependences: dependsOn
3) Task completion & return values
   - Blocking (i.e. “Futures”)
   - Non-blocking
4) Exception handling
1\textsuperscript{st} approach: Blocking (typical "Future" concept)

... 

TaskID id4 = task4("image.jpg");
File result = id4.getResult(); // blocking
display(result);
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

Event queue

Event loop

Event handler (mouse click)

id = task4();

id.getResult();

Event handler (refresh GUI)

GUI components

Helper Thread

task4()
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

Event queue

Event loop

Event handler (mouse click)

id = task4();

id.getResult();

Event handler (refresh GUI)

GUI components

Helper Thread
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

Event queue

Event loop

Event handler (mouse click)

... id = task4(); id.getResult();

Event handler (refresh GUI)

GUI components

Helper Thread

task4()
Additional features

1) Different task types: `TASK(*)`, `INTERACTIVE_TASK`

2) Task dependences: `dependsOn`

3) Task completion & return values
   - Blocking (i.e. “Futures”): `getResult()`
   - Non-blocking

4) Exception handling
Task completion & return values

2\textsuperscript{nd} approach: Non-blocking

... 

TaskID id4 = task4() \texttt{notify(display(TaskID))};
/* ... no blocking, return to Event Loop ... */
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

Event queue

Event loop

Event handler (mouse click)

id = task4()
notify(display());

Event handler (display())

Helper Thread

task4()

GUI components
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

Event queue

Event loop

Event handler (mouse click)

\[ \text{id = task4()} \]

\[ \text{notify(display())}; \]

Event handler (display())

Helper Thread

task4()
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

- Event queue
- Event loop
- Event handler (mouse click)
  \[ \text{id} = \text{task4()} \]
  \[ \text{notify(display())} \]
- Event handler (display())

Helper Thread

- task4()

GUI components
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

Event queue

Event loop

Event handler (mouse click)

\[
\text{id} = \text{task4()}
\]

\[
\text{notify(display());}
\]

Event handler (display())

Helper Thread

task4()
Structure of desktop applications

GUI Thread, Event Dispatch Thread (EDT)

Event queue

Event loop

Event handler

(mouse click)

\[ \text{id} = \text{task4()} \]

notify(display());

Event handler

(display())

GUI components

Helper Thread

task4()
Putting it all together...

```
TaskID id1 = task1(images.at(i));
TaskID id2 = task2(images.at(i)) dependsOn(id1);
TaskID id3 = task3(images.at(i)) dependsOn(id1);
TaskID id4 = task4(images.at(i)) dependsOn(id2, id3)
         notify(display(TaskID));
```
**Additional features**

1) Different task types: `TASK(*), INTERACTIVE_TASK`

2) Task dependences: `dependsOn`

3) Task completion & return values
   - Blocking (i.e. “Futures”): `getResult()`
   - Non-blocking: `notify`

4) Exception handling
Exception handling

```java
TASK int myTask() throws IOException {
    ...
}
```

```java
void myMethod() {
    ...
    TaskID id = myTask();
    ...
}
```
Exception handling

```java
TASK int myTask() throws IOException {
    ...
}
```

```java
void myMethod() {
    ...
    TaskID id = myTask();
    ~~~~~~~~~~~~~~~~~~~~~~
    error Unhandled exception type IOException
}
```
Exception handling

```java
TASK int myTask() throws IOException {
    ...
}

void myMethod() {
    ...
    TaskID id = myTask()
    trycatch(IOException handler());
}
```
Additional features

1) Different task types: \texttt{TASK(*)}, \texttt{INTERACTIVE\_TASK}

2) Task dependences: \texttt{dependsOn}

3) Task completion & return values
   - Blocking (i.e. “Futures”): \texttt{getResult()}
   - Non-blocking: \texttt{notify}

4) Exception handling: \texttt{trycatch}
Related work

- Tasks as **objects**
  - Active objects
  - ThreadWeaver, Intel TBB, SwingWorker
- Tasks as **functions**
  - Cilk++ / JCilk, CC++
  - Visual Studio 2010 TPL, X10, QtConcurrent
  - OpenMP tasks

Over 100 concurrent OO languages surveyed by [Philippsen 2000]
Implementation overview

```java
void method() {
    ...
    myTask("Hello");
    ...
}
```

Main thread
Implementation overview

```c
void method() {
    ...
    myTask("Hello");
    ...
}
```

```c
TaskID enqueue() {
    // analyse dependencies,
    // save arguments,
    // enqueue task,
    // ..., return ID
}n```

Main thread

Taskpool
void method() {
    ...
    myTask("Hello");
    ...
}

TaskID enqueue() {
    // analyse dependencies,
    // save arguments,
    // enqueue task,
    // ..., return ID
    ...
}

Implementation overview
Implementation overview

```java
void method() {
    ...
    myTask("Hello");
    ...
}
```

```java
TaskID enqueue() {
    // analyse dependencies,
    // save arguments,
    // enqueue task,
    // ..., return ID
}
```

Main thread

Taskpool
Implementation overview

```c
void method() {
    ...
    myTask(“Hello”);
    ...
}
```

```c
TaskID enqueue() {
    // analyse dependencies,
    // save arguments,
    // enqueue task,
    // ..., return ID
}
```
void method() {
    ...
    myTask(“Hello”);
    ...
}

TaskID enqueue() {
    // analyse dependencies,
    // save arguments,
    // enqueue task,
    // ..., return ID
}

TASK int myTask(String str) {
    // user code
}

Main thread

Worker thread

Taskpool
Implementation overview

1) Source to source compiler
2) Runtime system
Performance

1) Compute-intensive applications
   - Balanced workload
   - Unbalanced workload

2) Disk-intensive applications

3) Recursive applications
Compute-intensive & balanced workload

Comparison to traditional parallelism approaches (balanced)
Compute-intensive & unbalanced workload

Comparison to traditional parallelism approaches (unbalanced)

- Ideal speedup
- JT – max
- JT – min
- SwingWorker
- PT – Stealing
- PT – Sharing
- PT – Mix
- Jcilk

Speedup vs. Processor count
Disk-intensive workload
Recursive (fine grained)
Conclusions

- Multi-cores are here!
- Parallelisation of desktop applications
- OOP parallelism, familiar to developers
- Encapsulation of scheduling and parallelisation concerns
- ParaTask: different task types, dependences, non-blocking, exception handling

www.ece.auckland.ac.nz/~ngia003
Parsing task declarations

```java
TASK int myTask(String str) {
    /* user code */
}
```
Parsing task declarations

```cpp
TASK int myTask(String str) {
    /* user code */
}

TaskID myTask(String str, TaskInfo t) {
    return Taskpool.enqueue<int>(
        "__p_myTask(String)",
        t, ANY_THREAD, ARG(String, str));
}

int __p_myTask(String str) {
    /* user code */
}
```
Parsing task declarations

```cpp
TASK int myTask(String str) {
    /* user code */
}

TaskID myTask(String str, TaskInfo t) {
    return Taskpool.enqueue<int>(
        "__p_myTask(String)",
        t, ANY_THREAD, ARG(String, str));
}

int __p_myTask(String str) {
    /* user code */
}
```
Parsing task declarations

```c
TASK int myTask(String str) {
    /* user code */
}
```

```c
TaskID myTask(String str, TaskInfo t) {
    return Taskpool.enqueue<int>(
        "__p_myTask(String)",
        t, ANY_THREAD, ARG(String, str));
}
```

```c
int __p_myTask(String str) {
    /* user code */
}
```
TaskID id2 = myTask("Hello") dependsOn(id1)
         notify(slot(), obj::slot2());
TaskID id2 = myTask("Hello") dependsOn(id1) 
notify(slot(), obj::slot2());

TaskInfo __p_id2 = new TaskInfo();
__p_id2.addDependency(id1);
__p_id2.addNotify(this, "slot()");
__p_id2.addNotify(obj, "slot2()");
TaskID id2 = myTask("Hello", __p_id2);
TaskID id2 = myTask("Hello") dependsOn(id1)
notify(slot(), obj::slot2());

TaskInfo __p_id2 = new TaskInfo();
__p_id2.addDependency(id1);
__p_id2.addNotify(this, "slot()");
__p_id2.addNotify(obj, "slot2()");
TaskID id2 = myTask("Hello", __p_id2);
Parsing task invocations

TaskID id2 = myTask(“Hello”) dependsOn(id1)
notify(slot(), obj::slot2());

TaskInfo __p_id2 = new TaskInfo();
__p_id2.addDependency(id1);
__p_id2.addNotify(this, “slot()”);
__p_id2.addNotify(obj, “slot2()”);
TaskID id2 = myTask(“Hello”, __p_id2);
TaskID id2 = myTask("Hello") dependsOn(id1)
notify(slot(), obj::slot2());

TaskInfo __p_id2 = new TaskInfo();
__p_id2.addDependency(id1);
__p_id2.addNotify(this, "slot()");
__p_id2.addNotify(obj, "slot2()");
TaskID id2 = myTask("Hello", __p_id2);
Parsing task invocations

```java
TaskID id2 = myTask("Hello") dependsOn(id1)
    notify(slot(), obj::slot2());

TaskInfo __p_id2 = new TaskInfo();
__p_id2.addDependency(id1);
__p_id2.addNotify(this, "slot()");
__p_id2.addNotify(obj, "slot2()");
TaskID id2 = myTask("Hello", __p_id2);
```
TaskInfo __p_id = new TaskInfo();
Method _h = [Java reflection get "handler()"]
__p_id.addExcHandler(IOException.class, _h);
TaskID id = null;
try {
    id = myTask("Hello", __p_id);
} catch(IOException e){}
Parsing task invocations

```java
TaskID id = myTask("Hello") trycatch(
    IOException handler());

TaskInfo __p_id = new TaskInfo();
Method _h = [Java reflection get "handler()"
__p_id.addExcHandler(IOException.class, _h);
TaskID id = null;
try {
    id = myTask("Hello", __p_id);
} catch(IOException e){}
```
TaskID id = myTask("Hello") trycatch(IOException handler());

TaskInfo __p_id = new TaskInfo();
Method _h = [Java reflection get "handler()"
__p_id.addExcHandler(IOException.class, _h);
TaskID id = null;
try {
    id = myTask("Hello", __p_id);
} catch(IOException e){}
TaskID id = myTask("Hello") trycatch(IOException handler());

TaskInfo ___p_id = new TaskInfo();
Method _h = [Java reflection get "handler()"
___p_id.addExcHandler(IOException.class, _h);
TaskID id = null;
try {
    id = myTask("Hello", ___p_id);
} catch(IOException e){}
Runtime system

Waiting tasks

Ready tasks
- Wrk-Sh / Wrk-St / Mixed

Private ready queue

Interactive thread

Worker thread 1

Worker thread 2
Runtime system

Waiting tasks

Ready tasks
• Wrk-Sh / Wrk-St / Mixed

Private ready queue

Interactive thread

Worker thread 1

Worker thread 2
Runtime system

- Private ready queue
- Worker thread 1
- Worker thread 2
- Ready tasks
  - \textit{Wrk-Sh} / \textit{Wrk-St} / \textit{Mixed}
- Waiting tasks
- Interactive thread

Wrk-Sh / Wrk-St / Mixed
Runtime system

- Waiting tasks
- Ready tasks
  - Wrk-Sh / Wrk-St / Mixed
- Private ready queue
- Interactive thread
- Worker thread 1
- Worker thread 2
Runtime system

Waiting tasks

Ready tasks
• Wrk-Sh / Wrk-St / Mixed

Private ready queue

Interactive thread

Worker thread 1

Worker thread 2
Runtime system

- Waiting tasks
- Ready tasks
  - *Wrk-Sh / Wrk-St / Mixed*
- Private ready queue

Interactive thread
- Worker thread 1
- Worker thread 2
Runtime system

Waiting tasks

Ready tasks
- Wrk-Sh / Wrk-St / Mixed

Private ready queue

Interactive thread

Worker thread 1

Worker thread 2
Runtime system

- Private ready queue
- Worker thread 1
- Worker thread 2
- Waiting tasks
  - Interactive thread
- Ready tasks
  - *Wrk-Sh / Wrk-St / Mixed*
- Private ready queue