Motivations for & Benefits of Concurrency in Android (Part 3)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

• Understand the motivations for developing concurrent software on Android
• Recognize how concurrency can improve performance on Android
• Recognize how concurrency can improve responsiveness on Android
Impediments to Leveraging Hardware Parallelism
Impediments to Leveraging Hardware Parallelism

• Despite all the benefits of concurrency to improve performance, there are limits in practice to leveraging hardware parallelism
Impediments to Leveraging Hardware Parallelism

- Not every computing platform supports the latest hardware advances
Impediments to Leveraging Hardware Parallelism

- Not every computing platform supports the latest hardware advances
  - e.g., older computing devices just have a single core, which limits available parallelism
Impediments to Leveraging Hardware Parallelism

- It’s also hard to fully leverage parallelism due to various impediments, e.g.
Impediments to Leveraging Hardware Parallelism

• It’s also hard to fully leverage parallelism due to various impediments, e.g.
  • Legacy code that’s not thread safe

See en.wikipedia.org/wiki/Legacy_system

See www.wesleysteiner.com/professional/MakingLegacyCodeSafe.html
Impediments to Leveraging Hardware Parallelism

- It’s also hard to fully leverage parallelism due to various impediments, e.g.
  - Legacy code that’s not thread safe
  - e.g., accessing globally visible objects not protected by locks can cause “race conditions”

Race conditions occur when a program depends on the sequence or timing of threads for it to operate properly.

See [en.wikipedia.org/wiki/Race_condition#Software](en.wikipedia.org/wiki/Race_condition#Software)
It’s also hard to fully leverage parallelism due to various impediments, e.g.

- Legacy code that’s not thread safe
- GUI toolkits aren’t thread-safe by design
Impediments to Leveraging Hardware Parallelism

• It’s also hard to fully leverage parallelism due to various impediments, e.g.
  • Legacy code that’s not thread safe
  • GUI toolkits aren’t thread-safe by design, e.g.
  • Eliminate the need for internal locking
Impediments to Leveraging Hardware Parallelism

- It’s also hard to fully leverage parallelism due to various impediments, e.g.
  - Legacy code that’s not thread safe
  - GUI toolkits aren’t thread-safe by design, e.g.
    - Eliminate the need for internal locking
  - Minimize the need for app developers to understand concurrency
Impediments to Leveraging Hardware Parallelism

• It’s also hard to fully leverage parallelism due to various impediments, e.g.
  • Legacy code that’s not thread safe
  • GUI toolkits aren’t thread-safe by design, e.g.
    • Eliminate the need for internal locking
    • Minimize the need for app developers to understand concurrency
  • Android only allows the UI thread to access GUI components

See developer.android.com/training/multiple-threads/communicate-ui.html
Impediments to Leveraging Hardware Parallelism

• It’s also hard to fully leverage parallelism due to various impediments, e.g.
  • Legacy code that’s not thread safe
  • GUI toolkits aren’t thread-safe by design
  • The impact of Amdahl’s Law

“The speedup of a program using multiple processors is limited by the sequential portion of the program that can’t run in parallel”

See en.wikipedia.org/wiki/Amdahl's_law
Using Concurrency to Improve Responsiveness
Using Concurrency to Improve Responsiveness

- Concurrency can often be used to improve *perceived* response time

See [en.wikipedia.org/wiki/Responsiveness](en.wikipedia.org/wiki/Responsiveness)
Using Concurrency to Improve Responsiveness

• Concurrency can often be used to improve *perceived* response time, e.g.
• Don’t ignore user input while long-duration computations or communications are occurring

Using Concurrency to Improve Responsiveness

- Concurrency can often be used to improve *perceived* response time, e.g.
- Don’t ignore user input while long-duration computations or communications are occurring
  - e.g., allow worker threads to perform other processing in the background, while another thread handles user input
Using Concurrency to Improve Responsiveness

- Concurrency can often be used to improve *perceived* response time, e.g.
  - Don’t ignore user input while long-duration computations or communications are occurring
  - As long as the software infrastructure supports preemptive multi-threading even single core hardware can be more responsive

See [en.wikipedia.org/wiki/Preemption_(computing)](en.wikipedia.org/wiki/Preemption_(computing))
Using Concurrency to Improve Responsiveness

- Android concurrency frameworks define features & idioms that can help ensure & improve responsiveness

See developer.android.com/training/articles/perf-anr.html
Using Concurrency to Improve Responsiveness

- Android concurrency frameworks define features & idioms that can help ensure & improve responsiveness

Android UI thread can interact responsively with a user

The UI thread never blocks for more than a few seconds
Using Concurrency to Improve Responsiveness

- Android concurrency frameworks define features & idioms that can help ensure & improve responsiveness

Worker threads run long-duration computations in the background

Worker threads can block waiting for I/O or computations to complete
End of Motivations for & Benefits of Concurrency in Android (Part 3)
Motivations for & Benefits of Concurrency in Android (Part 4)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

• Understand the motivations for developing concurrent software on Android
• Recognize how concurrency can improve performance on Android
• Recognize how concurrency can improve responsiveness on Android
• Recognize how concurrency can improve program structure on Android
Overview of Event-Driven Programs
Overview of Event-Driven Programs

- Many software programs have historically been structured via a purely event-driven programming model.
Overview of Event-Driven Programs

- Many software programs have historically been structured via a purely event-driven programming model.
- The program flow is guided by events.

Overview of Event-Driven Programs

- Many software programs have historically been structured via a purely event-driven programming model
  - The program flow is guided by events
    - e.g., user interface actions, sensor I/O, messages from elsewhere, etc.

```java
Button button = (Button) findViewById(R.id.loadButton);

button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        ...
    }
});
```

A GUI component sending an event to its registered listener

GUI Component (e.g., a button)
Overview of Event-Driven Programs

- Many software programs have historically been structured via a purely event-driven programming model
  - The program flow is guided by events
  - Event-driven programming is a common model in GUIs that perform actions in response to user input

E.g., a single event loop in the Android UI thread processes all the user-facing events

See developer.android.com/training/multiple-threads/communicate-ui.html
Limitations with Purely Event-Driven Programs
Limitations with Purely Event-Driven Programs

- Event-driven software can be hard to understand for certain types of apps.
Limitations with Purely Event-Driven Programs

- Event-driven software can be hard to understand for certain types of apps
  - e.g., consider an app that downloads & displays images from remote web servers
Limitations with Purely Event-Driven Programs

- Event-driven software can be hard to understand for certain types of apps
  - e.g., consider an app that downloads & displays images from remote web servers

Moreover, there's just a single thread, i.e., the UI thread!
Limitations with Purely Event-Driven Programs

- Event-driven software can be hard to understand for certain types of apps
  - e.g., consider an app that downloads & displays images from remote web servers

Long-duration operations are run asynchronously by posting messages in the message queue of the UI thread’s event loop.
Limitations with Purely Event-Driven Programs

- Event-driven software can be hard to understand for certain types of apps
  - e.g., consider an app that downloads & displays images from remote web servers

  e.g., a message requesting an asynchronous read of a socket could be posted to the UI thread’s event loop
Limitations with Purely Event-Driven Programs

- Event-driven software can be hard to understand for certain types of apps
  - e.g., consider an app that downloads & displays images from remote web servers

Asynchronous operation completion is handled later via messages in the UI thread’s event loop
Limitations with Purely Event-Driven Programs

- Event-driven software can be hard to understand for certain types of apps
  - e.g., consider an app that downloads & displays images from remote web servers

  **e.g., the completion of the asynchronous read indicates how many bytes were read from the socket**
Limitations with Purely Event-Driven Programs

- Event-driven software can be hard to understand for certain types of apps
  - e.g., consider an app that downloads & displays images from remote web servers

See en.wikipedia.org/wiki/Lather,_rinse,_repeat
Limitations with Purely Event-Driven Programs

- Event-driven software can be hard to understand for certain types of apps
- e.g., consider an app that downloads & displays images from remote web servers

These steps may be repeated multiple times until the image is completely downloaded

Note the disconnect in both time & space between operation invocation & operation completion
Limitations with Purely Event-Driven Programs

- Poorly structured event-driven software often leads to “spaghetti code” or a “big ball of mud”

Limitations with Purely Event-Driven Programs

- Poorly structured event-driven software often leads to “spaghetti code” or a “big ball of mud”
- This type of code is hard to understand since there’s little/no structure or linear flow of control to guide developers
Limitations with Purely Event-Driven Programs

- Poorly structured event-driven software often leads to “spaghetti code” or a “big ball of mud”
- This type of code is hard to understand since there’s little/no structure or linear flow of control to guide developers
- Single-threaded programs that are driven purely by events can obscure the flow of control in time & space
Limitations with Purely Event-Driven Programs

- Poorly structured event-driven software often leads to “spaghetti code” or a “big ball of mud”
- This type of code is hard to understand since there’s little/no structure or linear flow of control to guide developers
- Single-threaded programs that are driven purely by events can obscure the flow of control in time & space

It’s hard to understand software when its flow of control “bounces” all over the place
Limitations with Purely Event-Driven Programs

- Poorly structured event-driven software often leads to “spaghetti code” or a “big ball of mud”
- This type of code is hard to understand since there’s little/no structure or linear flow of control to guide developers
  - Single-threaded programs that are driven purely by events can obscure the flow of control in time & space
  - It’s also hard to sustain since small changes can break nearly anything due to non-intuitive dependencies
Simplifying Program Structure with Concurrency
Simplifying Program Structure with Concurrency

- Concurrency mechanisms & frameworks can help overcome drawbacks with purely event-driven programs
Simplifying Program Structure with Concurrency

• Concurrency mechanisms & frameworks can help overcome drawbacks with purely event-driven programs

• Multi-threading enables more intuitive means of structuring software

See en.wikipedia.org/wiki/Multithreading_(computer_architecture)
Simplifying Program Structure with Concurrency

- Concurrency mechanisms & frameworks can help overcome drawbacks with purely event-driven programs
  - Multi-threading enables more intuitive means of structuring software
  - e.g., operations that run for long-durations can block synchronously
Concurrency mechanisms & frameworks can help overcome drawbacks with purely event-driven programs.

Multi-threading enables more intuitive means of structuring software.

- e.g., operations that run for long-durations can block synchronously.

Synchronous blocking within a thread can often yield a more natural & collaborative control flow.
Simplifying Program Structure with Concurrency

- Android’s HaMeR concurrency framework can help simplify program structure.
Simplifying Program Structure with Concurrency

- Android’s HaMeR concurrency framework can help simplify program structure.

Operations that run for long-durations can block in threads.
Simplifying Program Structure with Concurrency

- Android’s HaMeR concurrency framework can help simplify program structure

**Short-duration operations run in the UI thread & don’t block**
Operations that run for long-durations can block in threads

Short-duration operations run in the UI thread & don’t block

Android’s HaMeR concurrency framework supports a “hybrid” programming model, i.e., part event-driven & part concurrent
Simplifying Program Structure with Concurrency

- Android’s HaMeR concurrency framework can help simplify program structure

No more spaghetti code or big ball of mud since the app is neatly organized in a more “linear” structure that doesn’t bounce around!
Simplifying Program Structure with Concurrency

- Android’s HaMeR concurrency framework can help simplify program structure
- This framework implements various POSA concurrency patterns
  - e.g., Command Processor, Active Object, Half-Sync/Half-Async

See [www.dre.vanderbilt.edu/~schmidt/POSA](http://www.dre.vanderbilt.edu/~schmidt/POSA) & [en.wikipedia.org/wiki/Concurrency_pattern](http://en.wikipedia.org/wiki/Concurrency_pattern)
End of Motivations for & Benefits of Concurrency in Android (Part 4)
Motivations for & Benefits of Concurrency in Android (Part 5)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

- Understand the motivations for developing concurrent software on Android
- Recognize how concurrency can improve performance on Android
- Recognize how concurrency can improve responsiveness on Android
- Recognize how concurrency can improve program structure on Android
- Know how concurrency benefits can be applied to avoid overly complex & tangled event-driven solutions
Learning Objectives in this Part of the Module

- Understand the motivations for developing concurrent software on Android
- Recognize how concurrency can improve performance on Android
- Recognize how concurrency can improve responsiveness on Android
- Recognize how concurrency can improve program structure on Android
- Know how concurrency benefits can be applied to avoid overly complex & tangled event-driven solutions

e.g., an app that concurrently downloads an image from a remote web server & displays it to the user
Analyzing the Image Download App
Analyzing the Image Download App

- This app downloads images from remote web servers & displays them.

Analyzing the Image Download App

• This app downloads images from remote web servers & displays them
• Uses a Java Thread with Runnable implementation to download images

Allows image downloading to take advantage of multi-core processors
Analyzing the Image Download App

• This app downloads images from remote web servers & displays them
• Uses a Java Thread with Runnable implementation to download images
  • The Runnable implementation specifies what the Thread should do

Runnable downloadRunnable =
    new Runnable() {
        public void run() {
            /* Code to run goes here */
        }
    };
Analyzing the Image Download App

• This app downloads images from remote web servers & displays them

• Uses a Java Thread with Runnable implementation to download images
  • The Runnable implementation specifies what the Thread should do

Runnable downloadRunnable = new Runnable() {
    public void run() {
        /* Code to run goes here */
    }
};

• Thread class is used to create & control the Thread that processes the Runnable

new Thread(downloadRunnable).start();

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html
Analyzing the Image Download App

- This app downloads images from remote web servers & displays them
- Uses a Java Thread with Runnable implementation to download images

See upcoming lesson on “Overview of Java Threads”
Analyzing the Image Download App

• This app downloads images from remote web servers & displays them
  • Uses a Java Thread with Runnable implementation to download images
  • Uses Android “HaMeR” concurrency framework to pass results to UI thread

See code.tutsplus.com/tutorials/concurrency-on-android-using-hamer-framework--cms-27129
Analyzing the Image Download App

- This app downloads images from remote web servers & displays them
- Uses a Java Thread with Runnable implementation to download images
- Uses Android “HaMeR” concurrency framework to pass results to UI thread

Downloading of images is done in background threads
Analyzing the Image Download App

- This app downloads images from remote web servers & displays them
- Uses a Java Thread with Runnable implementation to download images
- Uses Android “HaMeR” concurrency framework to pass results to UI thread

Displaying of images is done in the UI thread
Analyzing the Image Download App

- This app downloads images from remote web servers & displays them
  - Uses a Java Thread with Runnable implementation to download images
  - Uses Android “HaMeR” concurrency framework to pass results to UI thread
  - This solution intentionally doesn’t handle interrupts or runtime configuration changes

DEFECTIVE BY DESIGN

See upcoming lesson on “Managing the Java Thread Lifecycle”
Analyzing the Image Download App

- As a reminder, the purely event-driven solution was *not* cohesive

Note disconnect between operation invocation & operation completion in both time & space

See the previous part of this lesson for details
Analyzing the Image Download App

• The structure of this concurrent app is more cohesive in time & space

... Called when user presses download button

```java
public void downloadImage(View view) {
    ...
    Uri url = getUrl();
    ...
    showDialog("downloading via HaMeR");
    startDownload(url);
    ...
}
```
public void downloadImage(View view) {
    ... 
    Uri url = getUrl();  // Get the URL entered by user
    ... 
    showDialog("downloading via HaMeR");
    startDownload(url);
    ...
The structure of this concurrent app is more cohesive in time & space

```java
public void downloadImage(View view) {
    Uri url = getUrl();
    showDialog("downloading via HaMeR");
    startDownload(url);
}
```

Display dialog to user while download is in progress
• The structure of this concurrent app is more cohesive in time & space

```
public void downloadImage(View view) {
    Uri url = getUrl();
    showDialog("downloading via HaMeR");
    startDownload(url);
    ...
```

Initiate downloading of the image at the url
Analyzing the Image Download App

• The structure of this concurrent app is more cohesive in time & space

... Performs the image download

public void startDownload(Uri url) {
    Runnable downloadRunnable =
    new Runnable() {

        public void run() {
            Uri pathname =
            downloadImage
            (this, url);

            runOnUiThread(new Runnable() {
                public void run() {
                    displayImage(pathname);
                }});

            new Thread(downloadRunnable).start();
...


public void startDownload(Uri url) {
    Runnable downloadRunnable =
    new Runnable() {
        public void run() {
            Uri pathname =
            downloadImage
            (this, url);

            runOnUiThread(new Runnable() {
                public void run() {
                    displayImage(pathname);
                }
            });
        }
    });

    new Thread(downloadRunnable).start();
    ...
}

Create a Runnable command
Start a new thread

new Thread(downloadRunnable).start();

...
Analyzing the Image Download App

• The structure of this concurrent app is more cohesive in time & space

... 

public void startDownload(Uri url) {
    Runnable downloadRunnable =
    new Runnable() {

        public void run() {
            Uri pathname =
            downloadImage
            (this, url);

            runOnUIThread(new Runnable() {
                public void run() {
                    displayImage(pathname);
                }
            });

            new Thread(downloadRunnable).start();
...
public void startDownload(Uri url) {
    Runnable downloadRunnable =
    new Runnable() {
        public void run() {
            Uri pathname =
            downloadImage
            (this, url);

            runOnUiThread(new Runnable() {
                public void run() {
                    displayImage(pathname);
                }
            });
        }
    };

    new Thread(downloadRunnable).start();
    ...

    Download image (blocking)
Analyzing the Image Download App

• The structure of this concurrent app is more cohesive in time & space

... public void startDownload(Uri url) {
    Runnable downloadRunnable =
    new Runnable() {

        public void run() {
            Uri pathname =
                downloadImage
                (this, url);

            runOnUIThread(new Runnable() {
                public void run() {
                    displayImage(pathname);
                }
            });

            new Thread(downloadRunnable).start();
    }
}

See www.dre.vanderbilt.edu/~schmidt/CommandProcessor.pdf
public void startDownload(Uri url) {
    Runnable downloadRunnable =
        new Runnable() {

            public void run() {
                Uri pathname =
                    downloadImage
                        (this, url);

                runOnUIThread(new Runnable() {
                    public void run() {
                        displayImage(pathname);
                    }
                });
            }

            new Thread(downloadRunnable).start();
        };
}
public void startDownload(Uri url) {
    Runnable downloadRunnable =
    new Runnable() {
        public void run() {
            Uri pathname =
            downloadImage
            (this, url);

            runOnUIThread(new Runnable() {
                public void run() {
                    displayImage(pathname);
                }
            });

            new Thread(downloadRunnable).start();
        }
    };
}
Analyzing the Image Download App

• The structure of this concurrent app is more cohesive in time & space

... public void startDownload(Uri url) {
    Runnable downloadRunnable =
    new Runnable() {
        public void run() {
            Uri pathname =
                downloadImage
                (this, url);
            runOnUiThread(new Runnable() {
                public void run() {
                    displayImage(pathname);
                }
            });
        }
    }
    new Thread(downloadRunnable).start();
...

The HaMeR-based version of this code is easier to understand than the purely event-driven version since it doesn’t bounce around!
The structure of this concurrent app is more cohesive in time & space

```
public void startDownload(Uri url) {
    new Thread(() -> {
        Uri pathname = downloadImage(this, url);

        runOnUiThread(() ->
            displayImage(pathname));
    }).start();
    ...
```

The Java 8 version using lambda expressions is even more concise since it omits all unnecessary syntax!
End of Motivations for & Benefits of Concurrency in Android (Part 5)