Supernova: Multicore Support for SuperCollider

Tim Blechmann
tim@klingt.org

Linux Audio Conference, 2011
Outline

SuperCollider
  Overview
  Examples
  SC Node Graph
Outline

SuperCollider
  Overview
  Examples
  SC Node Graph

Supernova
  Overview
  Parallel Groups
  Satellite Nodes
  Best Practices
Outline

**SuperCollider**
- Overview
- Examples
- SC Node Graph

**Supernova**
- Overview
- Parallel Groups
- Satellite Nodes
- Best Practices

**SuperCollider 3.5 preview**
SuperCollider

- Object-oriented programming language, real-time safe, designed for audio synthesis.
- Audio synthesis server for dynamically changing synthesis graphs.
- Separation between language and server, communication via OSC.
- Written by James McCartney for MacOS9 in the 1990s.
- Released as open source in early 2000, ported to Linux by Stefan Kersten.
Modular System

Synthesis Server

**Scsynth**, audio synthesis server with OSC interface. **Supernova** is a drop-in replacement. Unit Generators are loaded as plugins.

Synthesis Control Language

Typically **sclang** with its class library. Alternative scsynth clients in other languages like scala, scheme, clojure, haskell, . . .

Editors/IDEs

Scapp (OSX), Scate (Kate), Sced (Gedit), Scvim (Vim), Scel (Emacs), PsyCollider (windows), Eclipse.

GUI Systems

Cocoa (Scapp), Swing (Java), Qt.
Some SC Concepts I

**SynthDefs**

Instrument definitions, created in the language, loaded into the server.

**Synths**

SynthDef instantiation on the server.

**Busses**

Server-side routing system, used for synth communication.
Some SC Concepts II

- “There’s more than one way to do it”
- Raw OSC Communication, Classes, Events, Patterns, JitLib.
- Many implicit assumptions and syntactic sugar.
Listing 1: Creating Synths

// Raw OSC Messages
s.sendMsg("/s_new", "default", s.nextNodeID, 0, 1);

// SCLang nodes
a = Synth.new("default"); a.set("freq", 442)

// Events
().play; (freq: 440).play // implicit

// Functions
{
    Pulse.ar(440 ! 2,
            LFTri.kr({Rand(0.1, 0.2)}! 2)) * 0.1;
}.play
Patterns

Listing 2: Patterns

```plaintext
// Patterns
p = Pmono(default,
  dur, Pwrand([0.4, 0.05, 0.025],
    [0.5, 2, 2].normalizeSum,
    inf),
  degree, Pseq([
    Prand([0, 2, 3, 5], 6),
    Prand([0, 4, 6], 6)
  ], inf),
  pan, Prand([-1, 0, 1], inf),
  scale, Scale.new((0..11), 12, \just)
).play
```
SC Node Graph

Groups
Lists of nodes, define order of execution, address multiple nodes as one entity.

Nodes
Synths and Groups.

Node Hierarchy
Nodes form a tree hierarchy, each server has a root group and a node tree.
Order of Execution

User is responsible to ensure correct node order

Listing 3: Node Ordering

```plaintext
g = Group(s);
a = Synth.head(g, \default);
b = Synth.after(a, \fx);
```

Tim Blechmann tim@klingt.org

Supernova: Multicore Support for SuperCollider
Outline

**SuperCollider**
- Overview
- Examples
- SC Node Graph

**Supernova**
- Overview
- Parallel Groups
- Satellite Nodes
- Best Practices

**SuperCollider 3.5 preview**
Supernova

- Reimplementation of scsynth with a multiprocessor-aware synthesis engine.
- Loads slightly patched unit generators.
- Implements Node Graph Extensions to express parallelism explicitly.
Parallel Groups

- Groups without node ordering constraint
- Group elements can be executed in parallel
- Parallel groups can be nested to build more complex structures
Parallel Groups - An Example

Listing 4: Node graph with 4 generators and 1 effect

```
var gen_group, fx;
gen_group = ParGroup.new;
4.do {
    Synth.head(gen_group, \myGenerator)
};
fx = Synth.after(gen_group, \myFx);
```

Tim Blechmann tim@klingt.org

Supernova: Multicore Support for SuperCollider
Parallel Groups

Pro:

- Easy to use & understand
- Compatible with groups
Parallel Groups

Pro:
- Easy to use & understand
- Compatible with groups

Contra:
- User has to ensure correctness
- Each node has two dependency relations
Beyond Parallel Groups

- Parallel Groups are not optimal for all use cases.
- Some synths just need ordering constraints in relation with one node.
- Synths without input signals (generators).
- Synths without output signals (peak meters, disk/sample recording synths).
Beyond Parallel Groups

- Parallel Groups are not optimal for all use cases.
- Some synths just need ordering constraints in relation with one node.
- Synths without input signals (generators).
- Synths without output signals (peak meters, disk/sample recording synths).

**satellite nodes**
Satellite Nodes - Proposed Interface

- 2 new node constructors: `Node.preceding` & `Node.succeeding`
- 2 new add actions

Listing 5: Satellite node example

```plaintext
var fx = Synth.new("myFx");
4. do {
    var gen = Synth.preceding(fx, "myGenerator")
    Synth.succeeding(gen, "peakMeterSynth")
}
```
Satellite Nodes - Dependency Graph

**Figure**: Dependency Graph for Satellite Nodes Example

- Synth: myGenerator
- Synth: myGenerator
- Synth: myGenerator
- Synth: myGenerator
- Synth: PeakMeter
- Synth: PeakMeter
- Synth: myFx
- Synth: PeakMeter
- Synth: PeakMeter
- Synth: some other synth

Tim Blechmann tim@klingt.org

Supernova: Multicore Support for SuperCollider
Satellite Nodes - Semantics

- **Satellite nodes** are ordered in relation with one other node
- Each node can have multiple *satellite predecessors* and *satellite successors*
- Satellite nodes may have satellite nodes themselves
- Satellite nodes can be addressed by the parent group of their reference node
- Satellite nodes are freed, if their reference node are freed.
Satellite Nodes

Pro:
▶ Increases the parallelism for many use cases
▶ Optimized for ‘node graph progress’: Satellite nodes have a lower priority than other nodes
▶ The combination of satellite nodes and parallel groups can handle most use cases in a nearly optimal manner

Contra:
▶ Incompatible with scsynth (unless one implements it!)
▶ Increased complexity (dependency graph vs hierarchy tree)
▶ Experimental implementation in Supernova
Best Practices

- Keep the CPUs busy: use parallel node graph extensions.
Best Practices

- Keep the CPUs busy: use parallel node graph extensions.
- Avoid scheduling overhead: don’t parallelize light-weight work like control-rate synths.
Best Practices

- Keep the CPUs busy: use parallel node graph extensions.
- Avoid scheduling overhead: don’t parallelize light-weight work like control-rate synths.
- Avoid contention: accessing to the same resources from parallel synths may harm the performance.
Best Practices

- Keep the CPUs busy: use parallel node graph extensions.
- Avoid scheduling overhead: don’t parallelize light-weight work like control-rate synths.
- Avoid contention: accessing to the same resources from parallel synths may harm the performance.
- Do not use satellite nodes if the code should run on scsynth.
Best Practices

- Keep the CPUs busy: use parallel node graph extensions.
- Avoid scheduling overhead: don’t parallelize light-weight work like control-rate synths.
- Avoid contention: accessing to the same resources from parallel synths may harm the performance.
- Do not use satellite nodes if the code should run on scsynth.
- Parallel groups are well tested, satellite nodes are experimental.
Outline

SuperCollider
  Overview
  Examples
  SC Node Graph

Supernova
  Overview
  Parallel Groups
  Satellite Nodes
  Best Practices

SuperCollider 3.5 preview
SuperCollider 3.5 will be a **big** release!

- Migration to git and cmake
- Supernova: alternative to scsynth
- QtCollider: new cross-platform GUI system based on Qt
- ScDoc: new help system
QtCollider

- Developed by Jakob Leben, merged late 2010
- Qt-based implementation of the Cocoa widgets
- Better sclang integration, cocoa compatibility and responsiveness compared to SwingOSC.
Developed by Jonatan Liljedahl, merged early 2011
Browser-based doc system with on-the-fly rendering
Simple markup language
Conclusion

Thanks!
Questions?
Conclusion

Thanks!

Questions?