supernova - A Multiprocessor Aware Real-Time Audio Synthesis Engine For SuperCollider

Tim Blechmann
tim@klingt.org

Linux Audio Conference, 2010

Outline

SuperCollider

- **sclang**, a real-time scripting language, with a huge class library
- **scsynth**, an audio synthesis engine
- a huge number of unit generators, provided as plugins
- a gui system (with 2 implementations)
- several IDEs

supernova

- a multi-processor aware replacement for scsynth
- exposes parallelism to the user of the language
- designed for low-latency real-time applications

Parallelism

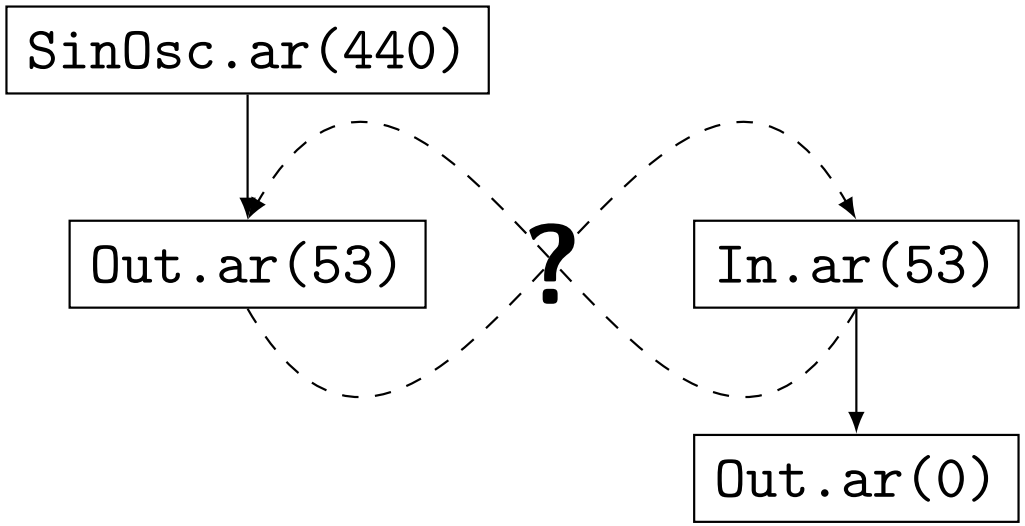
- channels
- voices, polyphonic music
- tracks/buses
- additive synthesis
- ...

Signal Graph Parallelism

- What is the size of the synthesis graph?
- How big are the CPU expenses of the nodes?
- Graph nodes can be combined to avoid scheduling overhead.

Node Dependencies

- There may be implicit dependencies based on resource access!
- A dependency analysis may not be trivial
- Automatic dependency analysis may be difficult or impractical to implement



Real-Time Node Scheduling

- How can the node graph be traversed in a real-time context?
- Deadlines are almost one milliseconds (64 samples)!
- Scheduling latency may be hundreds of microseconds (unless we are on highly tuned hardware with RT preemption patches).
- Avoid locks!

SuperCollider Node Graph

- SuperCollider uses **groups** for structuring the audio synthesis
- Groups are linked lists of **nodes**
- Nodes can be groups or **synths**.

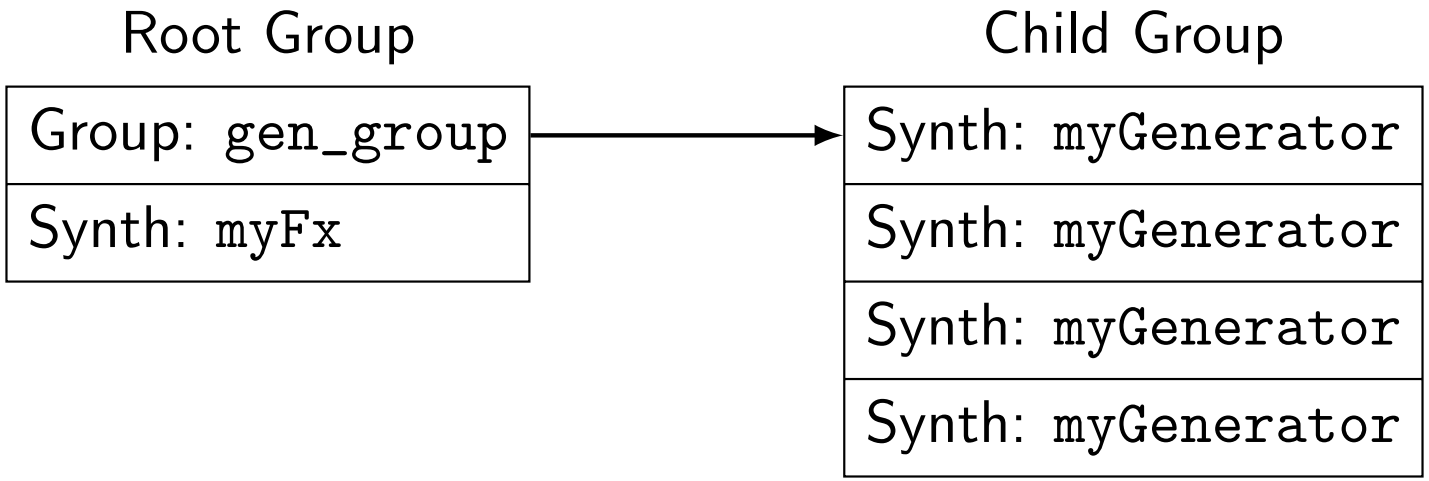
Groups - Features

- Form a tree hierarchy with a group as root
- Syntactic sugar for organizing the audio synthesis
- Multiple nodes can be addressed as one entity
- Expose the order of execution explicitly to the user

Groups - An Example

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

Groups - An Example (2)



Parallel Groups

How to introduce parallelism to concept of the SuperCollider node graph?

Parallel Groups

How to introduce parallelism to concept of the SuperCollider node graph?

Parallel Groups

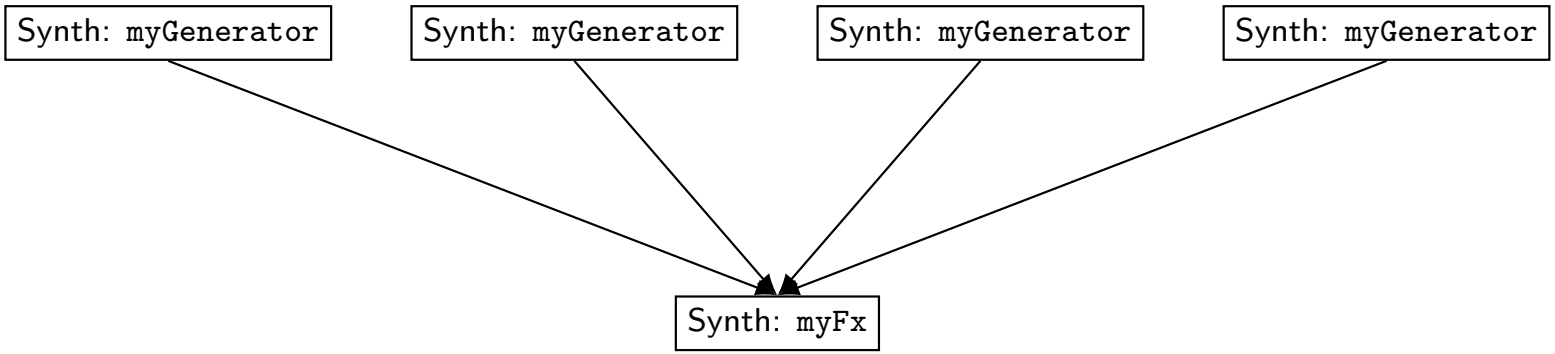
Parallel Groups

- Semantics similar to groups
- Nodes can be executed in parallel
- Integrates well into the SuperCollider node graph:
  One more class, one more OSC command for the server
- Backward compatible: Old code still keeps its semantics
- Forward compatible: Parallel groups can easily be emulated with groups

Parallel Groups - An Example

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

Parallel Groups - An Example (2)



Architecture of supernova: Features

- Focuses on latency instead on throughput
- No pipelining, so no additional latency
- Mostly lock-free synchronization (boost.lockfree)
- SC node graph needs to be transferred into a dependency graph representation.

Architecture of supernova: Issues

- Idle threads perform busy waiting (produces heat, takes resources)
- No automatic dependency checking
- Some use cases may be difficult to formulate to get the best performance
- ... and no non-rt synthesis (yet)

SuperCollider Unit Generators

- Supernova can load SuperCollider Unit Generators
- Unit generators need to be adapted to ensure data consistency
- All unit generators from the SuperCollider distribution have been ported to supernova
- Some unit generators from the sc3-plugins as well

Resource Consistency

- Reader-writer spinlocks are used to ensure data consistency
- Reading the same resource from parallel synths is safe
- Writing to the same resource from parallel synths may be safe:
  `Out.ar` is safe
  `ReplaceOut.ar` is not
- Writing to the same resource from parallel synths increases contention!

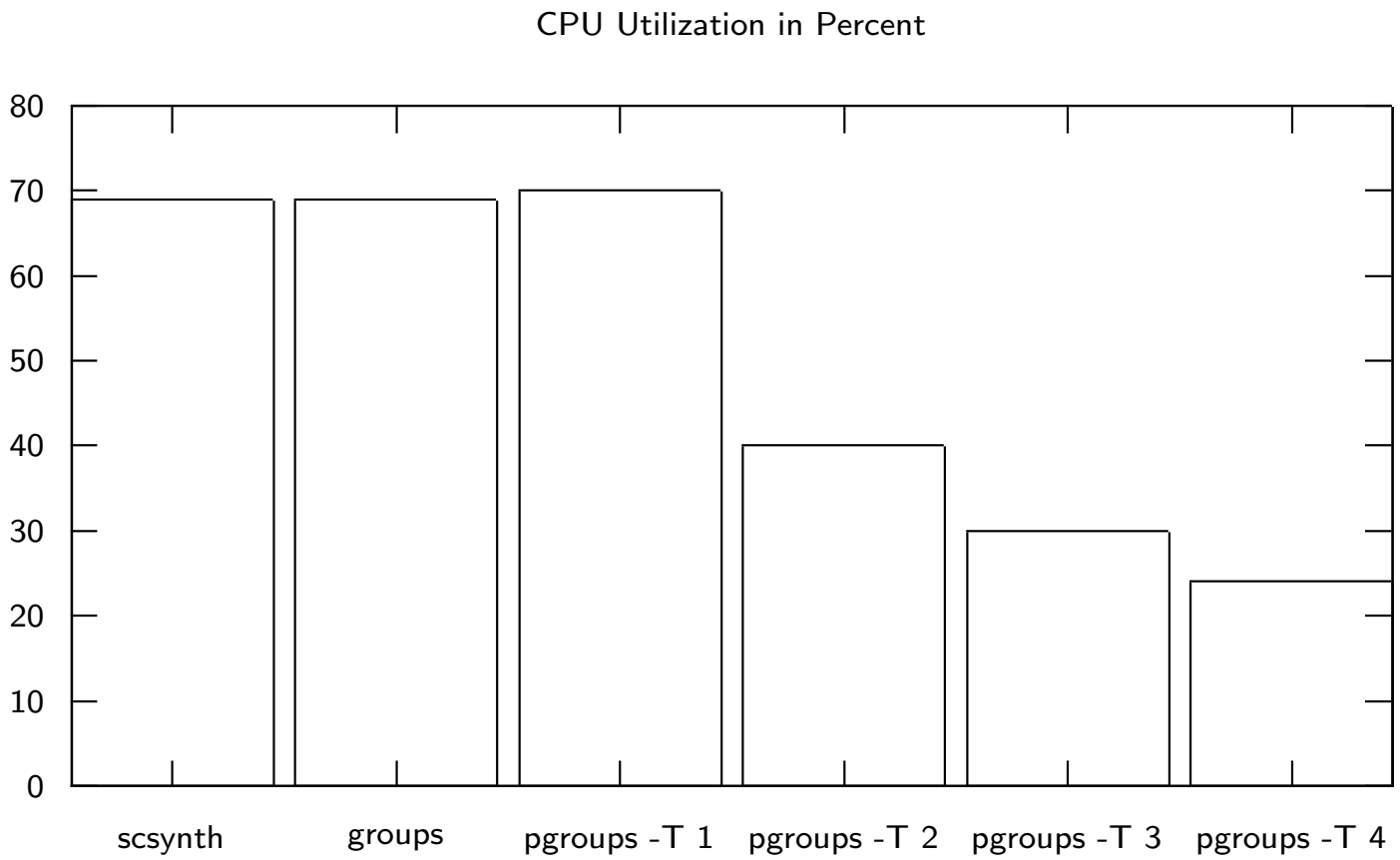
SuperCollider Unit Generators: An Example

```
void In_next_a(IOUnit *unit, int inNumSamples)
{
    [...]

    for (int i=0; i<numChannels; ++i,
         in += bufLength) {


        if (touched[i] == bufCounter)
            Copy(inNumSamples, OUT(i), in);
        else
            Fill(inNumSamples, OUT(i), 0.f);

    }
}
```

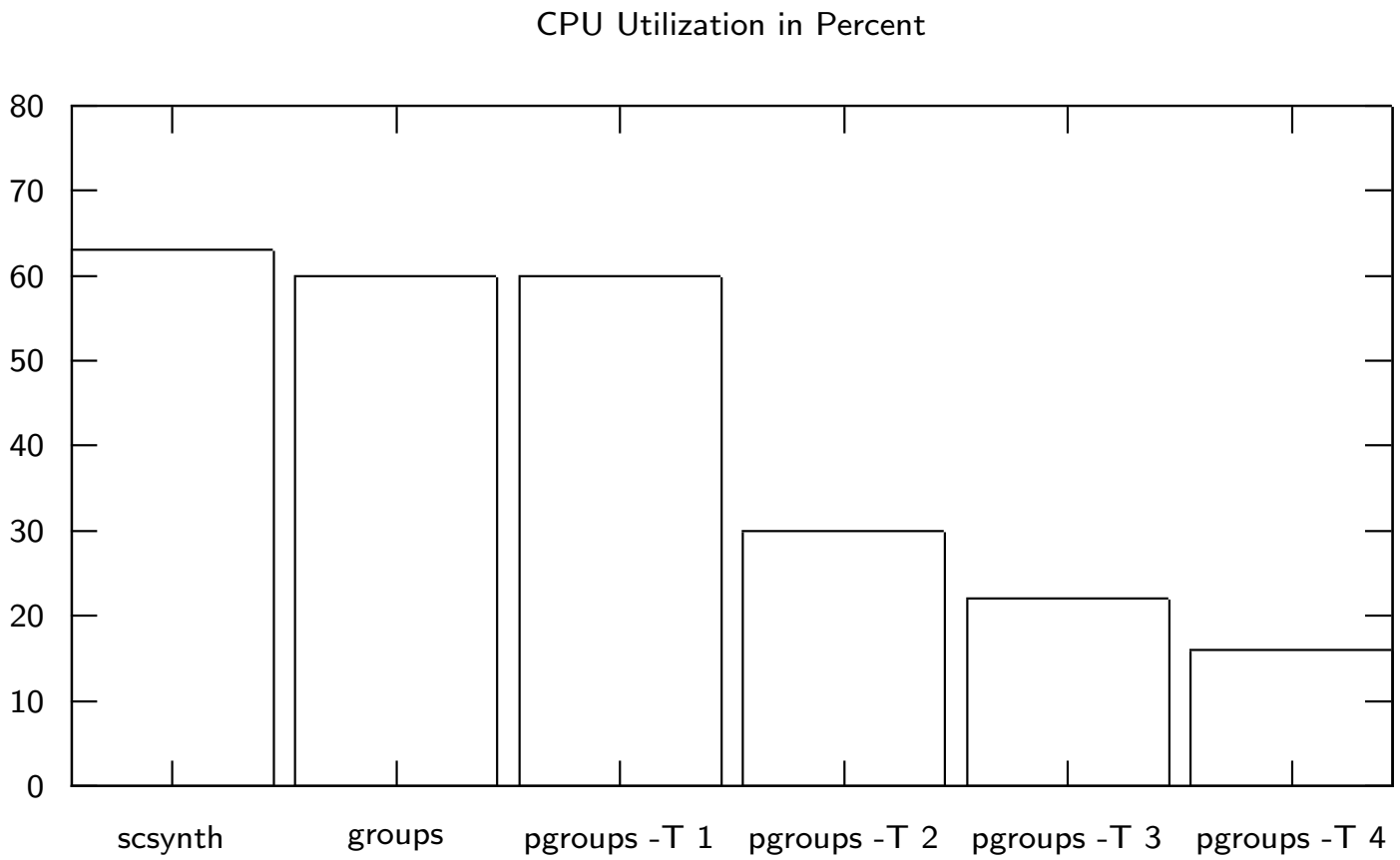
SuperCollider Unit Generators: An Example

```
void In_next_a(IOUnit *unit, int inNumSamples)
{
    [...]

    for (int i=0; i<numChannels; ++i,
         in += bufLength) {
        int32 busChannel = (int32)fbusChannel + i;
        ACQUIRE_BUS_AUDIO_SHARED(busChannel);
        if (touched[i] == bufCounter)
            Copy(inNumSamples, OUT(i), in);
        else
            Fill(inNumSamples, OUT(i), 0.f);
        RELEASE_BUS_AUDIO_SHARED(busChannel);
    }
}
```

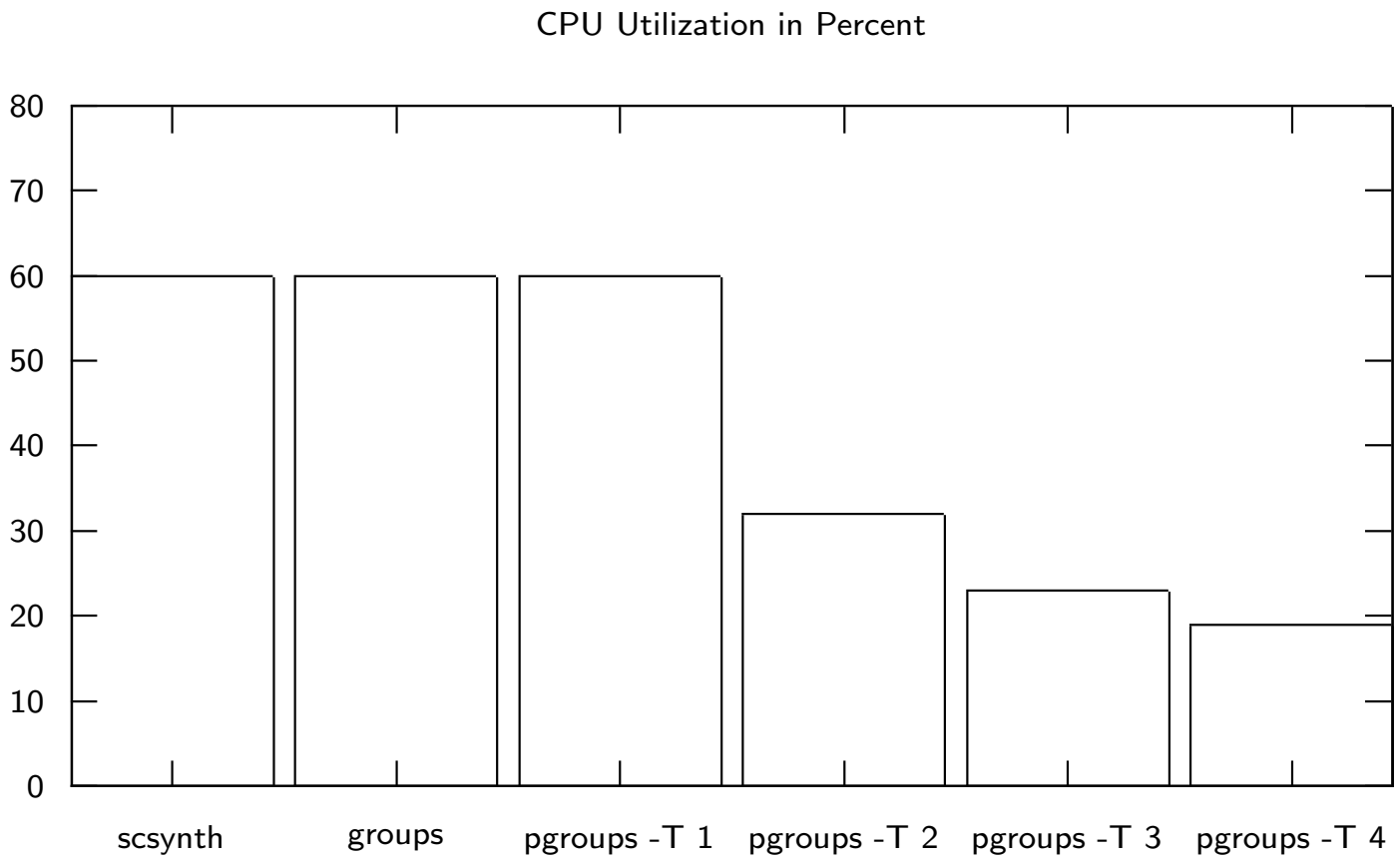
1024 Lightweight Synths



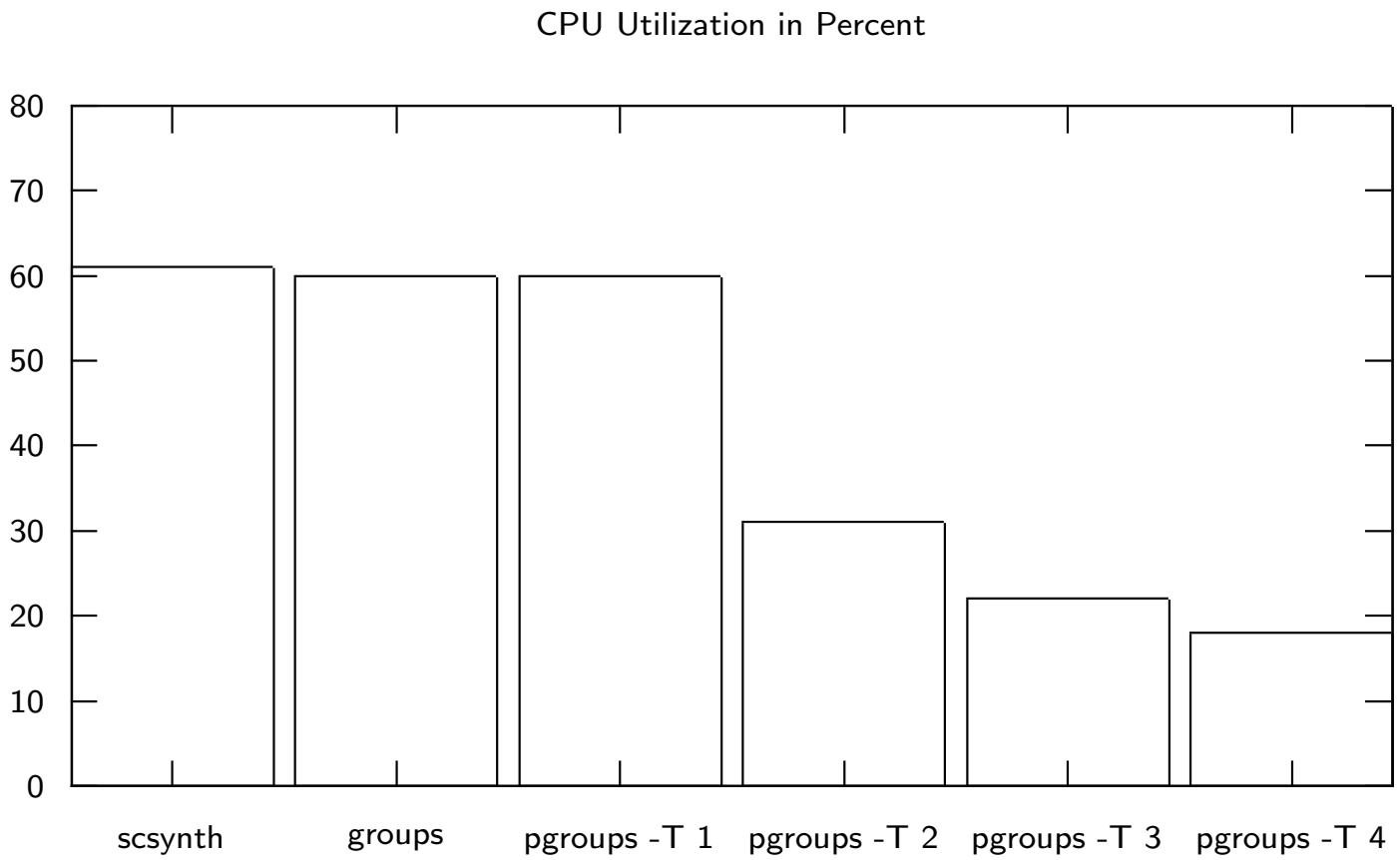
30 Heavyweight Synths



128 Synths with High Resource Contention



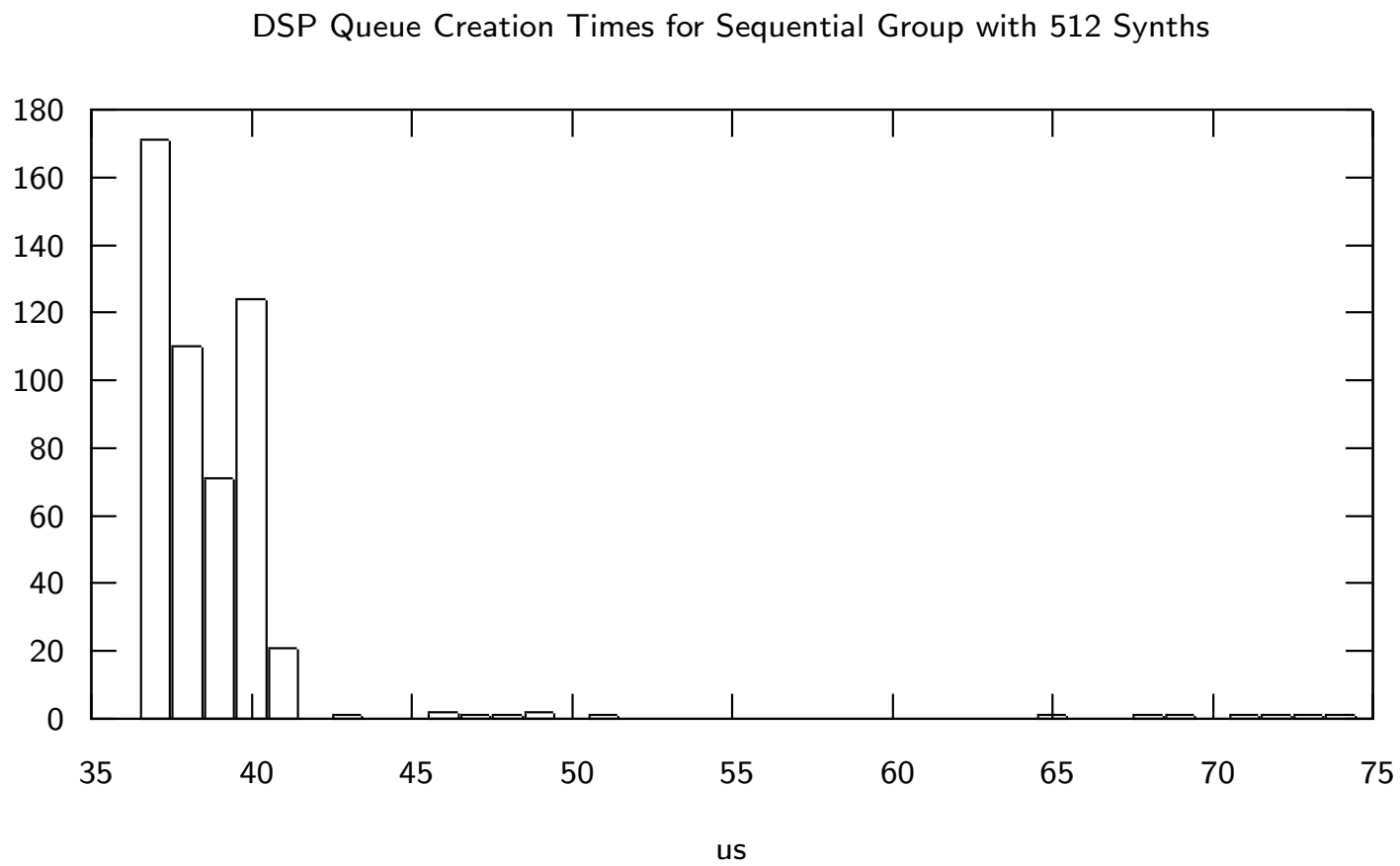
128 Synths with Low Resource Contention



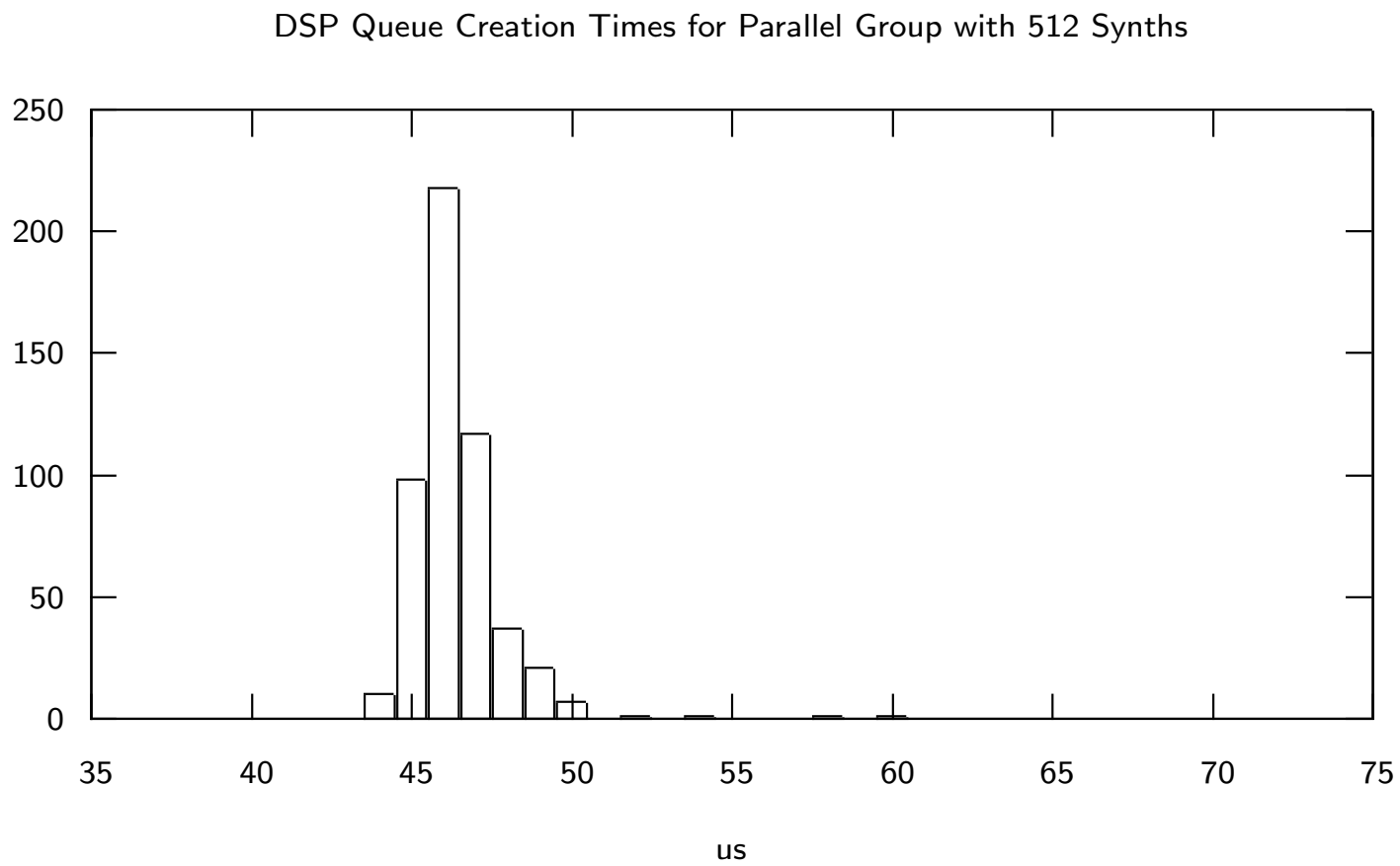
Speedup Overview

	Small Synths	Large Synths	High Contention	Low Contention
scsynth	1	0.95	1	0.98
sequential groups	1	1	1	1
parallel groups, 1 thread	0.98	1	1	1
parallel groups, 2 threads	1.72	2	1.87	1.94
parallel groups, 3 threads	2.33	2.73	2.61	2.73
parallel groups, 4 threads	2.88	3.75	3.16	3.33

Node Graph Parsing, Group



Node Graph Parsing, Parallel Group



Summary

- supernova is a drop-in replacement for scsynth
- supernova extends the SuperCollider node graph by a simple concept
- not a single crash during a concert

Conclusion

Thanks!
Questions?