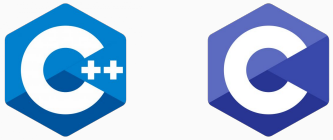
Analyzing the Loop Scheduling Mechanisms on Julia Multithreading

Diana A. Barros
Computational Sciences Program
State University of Rio de Janeiro

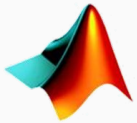
Cristiana Bentes
Systems Engineering Department
State University of Rio de Janeiro

“Come for the syntax, stay for the speed”

Performance



Productivity



MATLAB

Why Julia?

Type Inference

Avoids the allocation of variable size heaps and dynamic checks

Reduces the need for JIT compilation

Takes advantage of the LLVM optimization passes

Multiple Dispatch

Polymorphism - function behaves differently according to the types of the parameters

Improves compiler performance

Metaprogramming

Evaluates code at parse time

Reduces runtime overhead

Multithreading in Julia

Task usage - AKA Coroutines

```
julia> t = @task begin; sleep(5); println("done"); end
Task (runnable) @0x00007f13a40c0eb0

julia> schedule(t); wait(t)
```

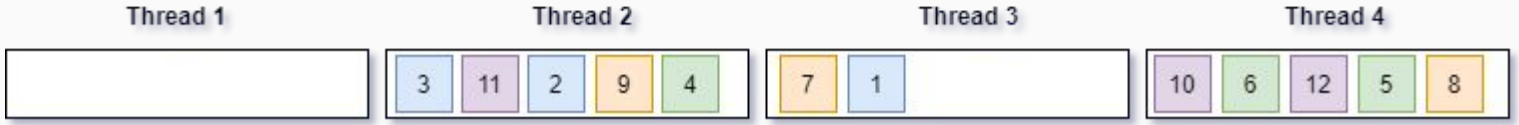
JULIA_NUM_THREADS defines the number of threads to be used in your code

Macros: Threads.**@threads**
Threads.**@spawn**

Dynamic and Static Scheduling on Julia

Threads.**@threads** - Static

Threads.**@spawn** - Dynamic




```
function fib(n::Int)
    if n < 2
        return n
    end
    t = @spawn fib(n - 2)
    return fib(n - 1) + fetch(t)
end
```

Parallelizing a Loop

```
Threads.@threads for i = 1:N
    a[i] = b[i] + c[i]
end
```

```
@sync for i = 1:N
    Threads.@spawn begin
        a[i] = b[i] + c[i]
    end
end
```

Experimental Results

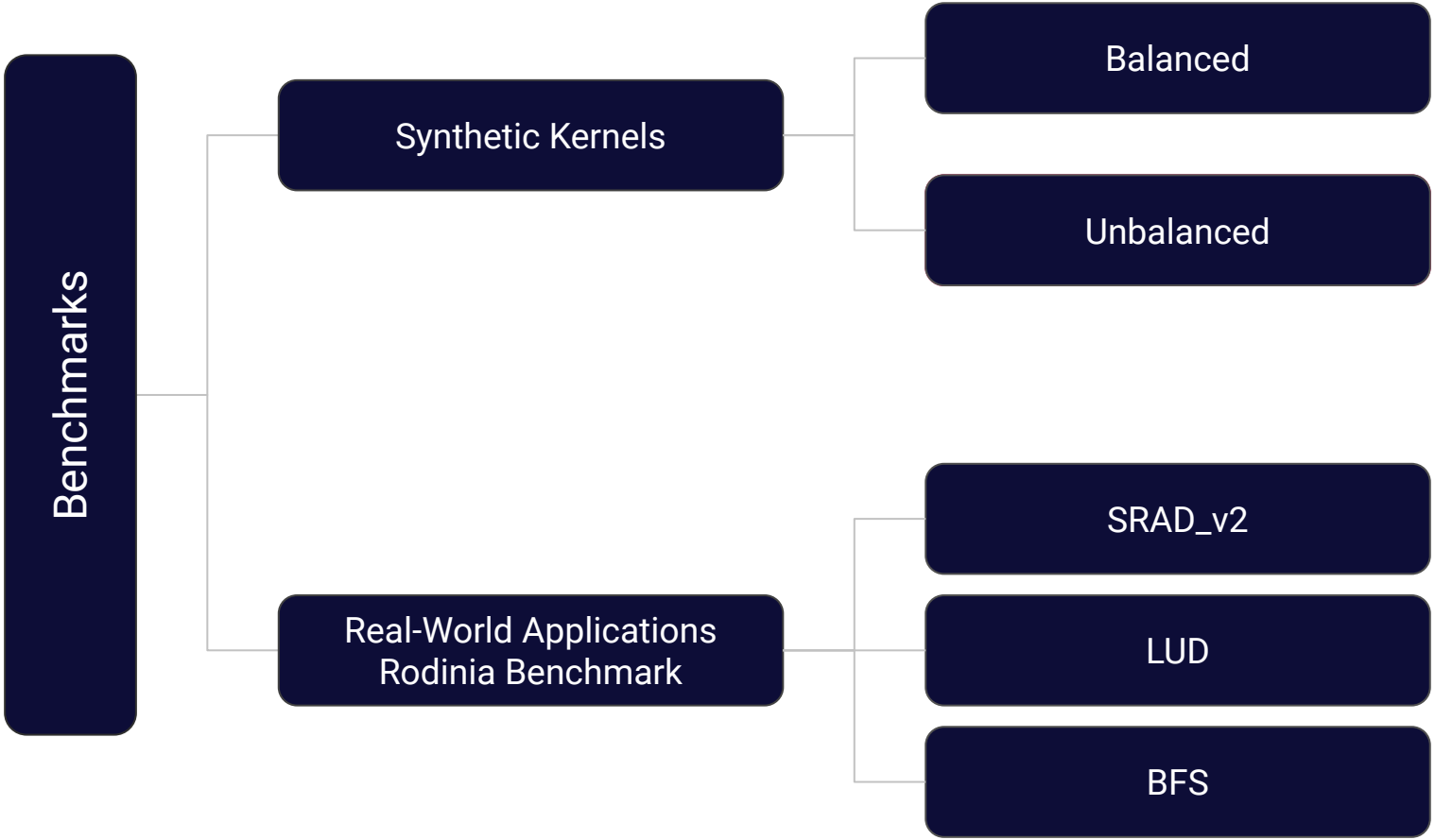
Intel i5-8250U CPU → 4 cores - hyperthreading → 8 threads → 8 GB RAM

JULIA_NUM_THREADS = 8

50 executions → Mean value

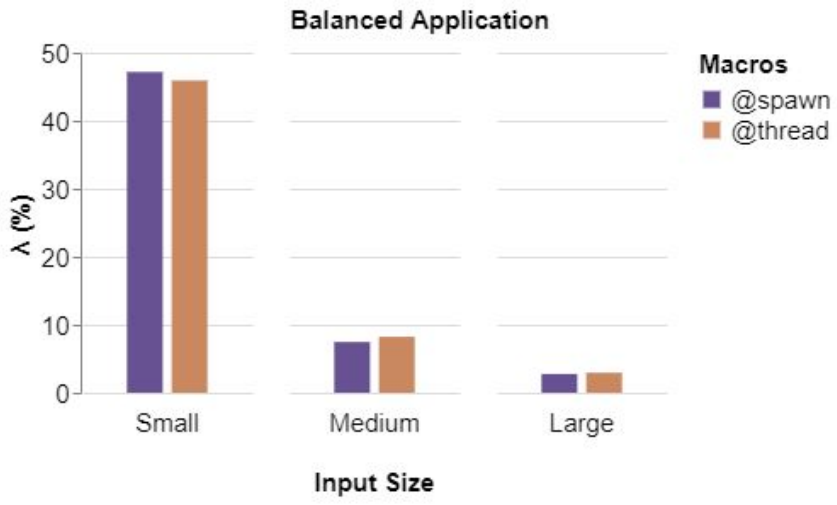
@timed

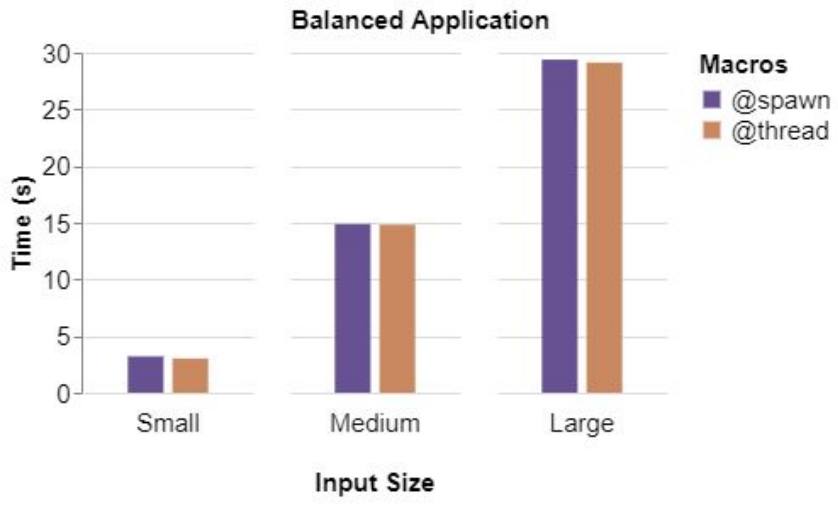
Load Imbalance Percentage → $\lambda = \frac{L_{max}}{\bar{L}} - 1 \times 100\%$


Benchmarks
Synthetic Kernels
Balanced
Unbalanced
Real-World Applications
Rodinia Benchmark
SRAD_v2
LUD
BFS

Synthetic Applications Results

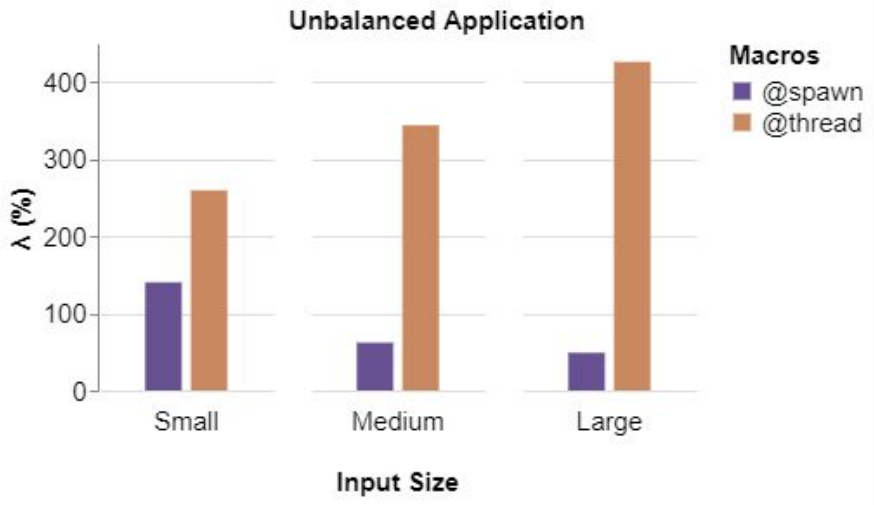
Balanced

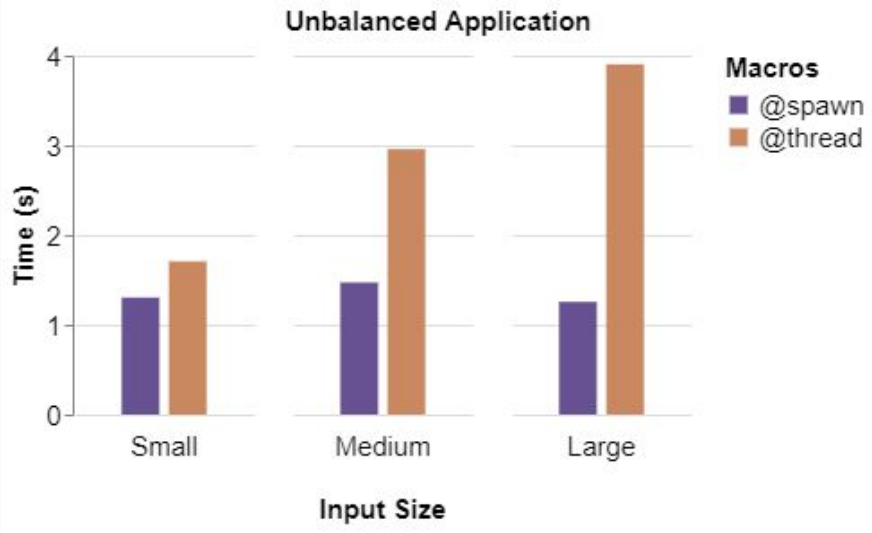




Synthetic Applications Results

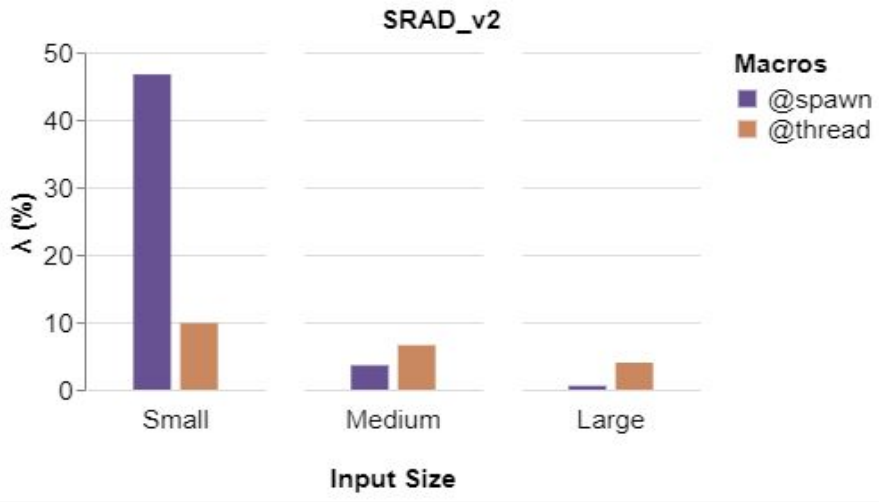
Unbalanced

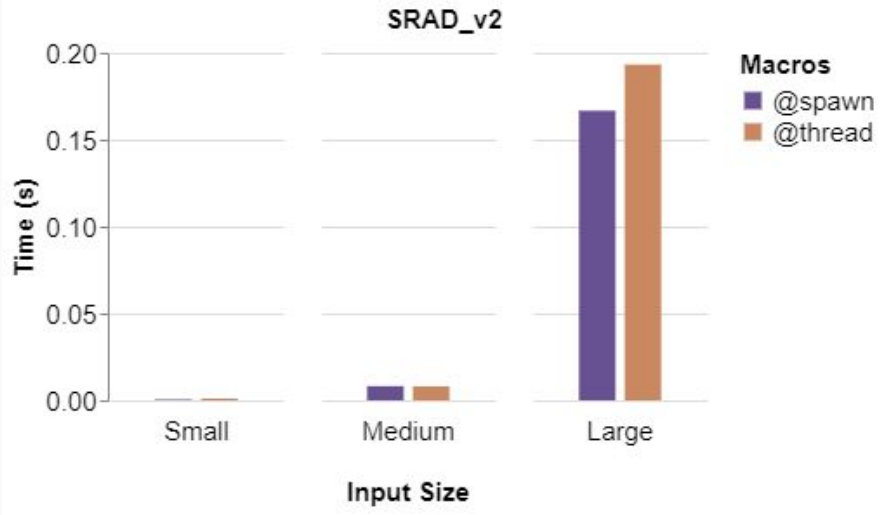




Real-world Applications Results

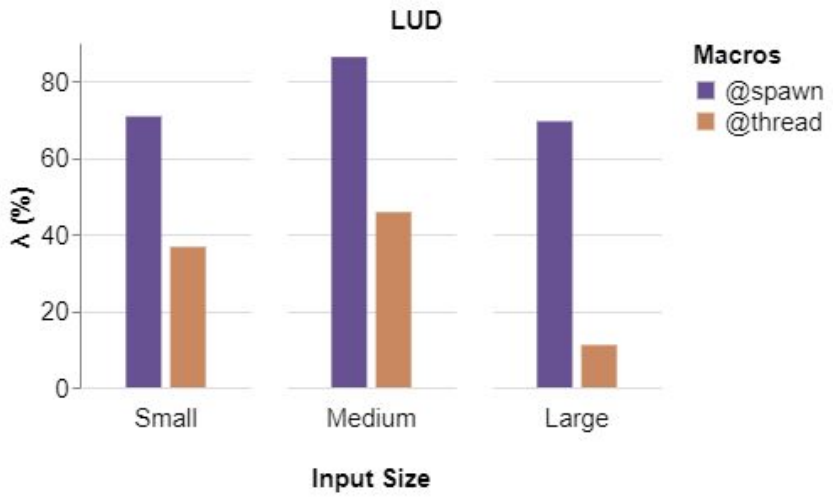
SRAD_v2

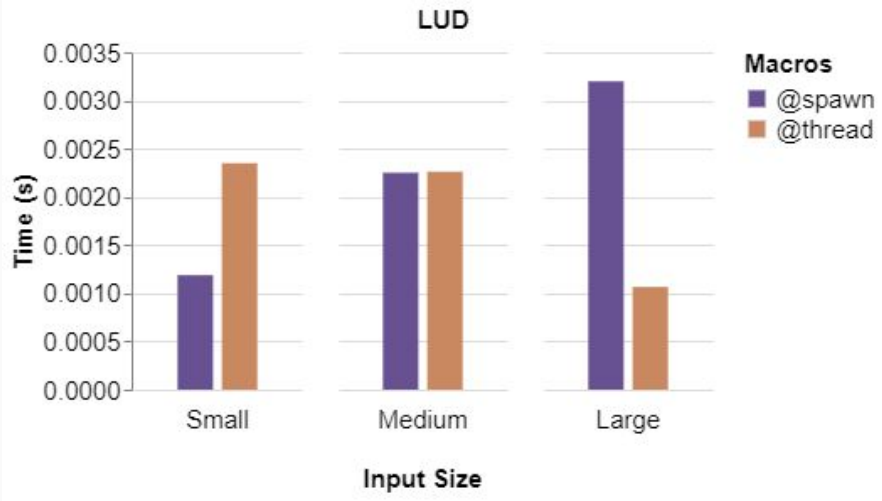




Real-world Applications Results

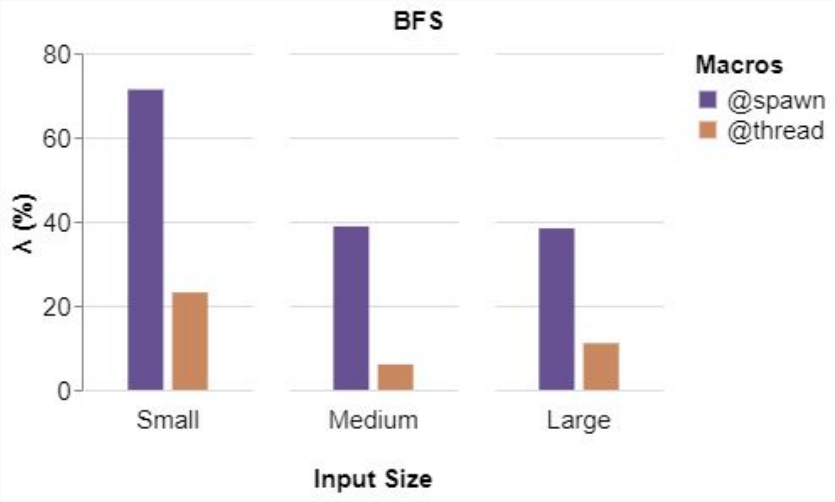
LUD

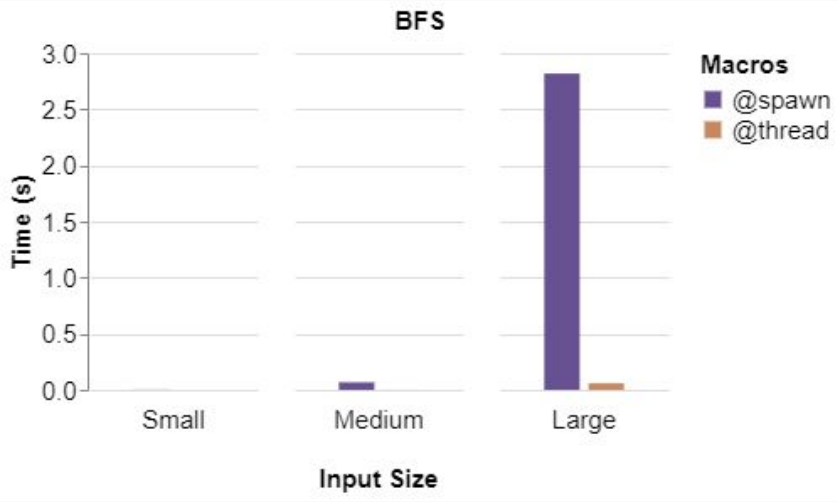




Real-world Applications Results

BFS





Conclusions

- The schedule choice depends on the behavior of the application
- Static scheduling → Balanced applications
- Dynamic scheduling → Unbalanced applications
- To be considered:
  - Work stealing overhead
  - Input size

Thank you!

diana.ab@live.com
cris@eng.uerj.br