# Nornir: a Power-Aware Runtime Support

**Daniele De Sensi**
University of Pisa, Department of Computer Science

## Target

Reconfigure parallel applications to satisfy user requirements on performance and power consumption

## Application's User

Specifies *performance* and *power consumption* constraints on the application

**Example:**

```
<?xml version="1.0" encoding="UTF-8"?>
<nornirParameters>
    <requirements>
        <bandwidth>MAX</bandwidth>
        <latency>30</latency>
        <powerConsumption>50</powerConsumption>
    <requirements>
</nornirParameters>
```

## Application's Programmer

Interfaces an existing application to Nornir, to enable it to collect performance data and to apply reconfiguration decisions

## Algorithm Designer

Implements new strategies by relying on the *monitor* and *execute* infrastructure provided by Nornir

## Conclusions

- Algorithms capable of finding **faster**, **better** solutions than state of the art algorithms
- Full **software stack** to apply existing algorithms on applications and to design and implement new algorithms
- Wide set of possibilities for **interfacing** the framework to existing applications
- Proper **benchmark** created
- Evaluation of **effectiveness** of existing reconfiguration mechanisms

## Applications

### Supported Frameworks (Parallel Design Patterns)[2,3,6,7,13,a]

**Pro**: More control knobs available

**Cons**: Re-implementing the application

Since there was no benchmark available for parallel patterns-based applications, we developed **P$^3$ARSEC**. By modelling **PARSEC** applications using parallel patterns, we are able to test some details of our reconfiguration algorithms

Tested on different architectures:
- Intel Xeon Server
- Intel Xeon Phi
- IBM Power8

### Instrumentation[b]

**Pro**: Less intrusive

**Cons**: Less knobs available

```
StreamElement* s;
while(s = receive()){
    process(s);
}
```

→

```
using nornir::Instrumenter;
Instrumenter r("parameters.xml");
StreamElement* s;
while(s = receive()){
    r.begin();
    process(s);
    r.end();
}
r.terminate();
```

### Black-Box

**Pro**: No intervention on existing applications

**Cons**: Less information provided to the reconfiguration algorithm. Fewer requirementes can be specified.

```
./manager-blackbox --parameters parameters.xml
                   --application ./foobar
```

## Algorithms

### Heuristic[10,6,13,c]

**Scenario**: Can provide explicit performance guarantees, while minimizing power consumption

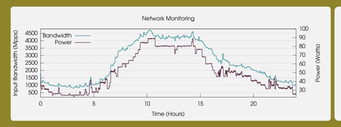
**Algorithm**: Change number of used cores (***n***) and clock frequency (***f***) according a naive analytical formula

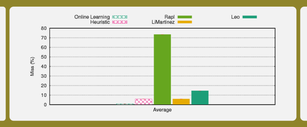
### Online Machine Learning[1,9,c]

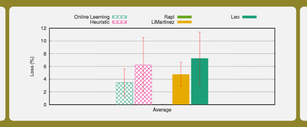
**Scenarios**: Can provide explicit performance and power consumption guarantees

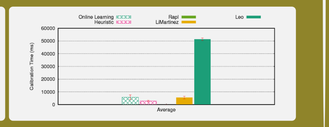
**Algorithm**: Obtain precise models for performance and power consumption by interpolating those of some configurations

**Real execution** on PARSEC applications, enforcing power consumptio or performance requirements



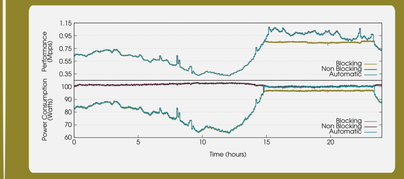




### Concurrency Control[5,12,c]

**Scenario**: Streaming applications, decide best synchronization algorithm to use when accessing shared communication queues

Switch between active wait (*Non Blocking*) and lock-based wait (*Blocking*) when accessing shared single-producer, single-consumer queues




## Mammut[4,d]

- C++ Library
- Uniform interface across different architectures
- Remote management

```
#include <mammut/mammut.hpp>

using namespace mammut;
using namespace std;

int main(int argc, char** argv){
    Mammut m;
    Energy* energy = m.getInstanceEnergy();
    CpuFreq* frequency = m.getInstanceCpuFreq();
    Topology* topology = m.getInstanceTopology();
    TasksManager* tasks = m.getInstanceTask();
    // Shutdown all the CPUs except CPU 0.
    Cpu* cpu = topology->getCpu(0);
    vector<Cpu*> cpus = topology->getCpus();
    for(size_t i = 0; i < cpus.size(); i++){
        if(cpus[i] != cpu){ cpus[i]->hotUnplug(); }
    }

    // Move this process on CPU 0.
    ProcessHandler* process = tasks->getProcessHandler(getpid());
    process->move(cpu);
    tasks->releaseProcessHandler(process);
    // Set the highest clock frequency to CPU 0.
    vector<Domain*> domains = frequency->getDomains(cpu);
    for(size_t i = 0; i < domains.size(); i++){
        domains[i]->setGovernor(GOVERNOR_USERSPACE);
        domains[i]->setHighestFrequencyUserspace();
    }
    // Execute workload and measure energy consumption of CPU 0.
    CounterCpus* c = (CounterCpus*) energy->getCounter(COUNTER_CPU5);
    if(c){ c->reset(); }
    // Do work
    if(c){ cout << "Joules: " << c->getJoulesCpu(cpu) << endl; }
}
```

## Hardware Configuration

- Number of Cores[8,14]
- Clock Frequency
- Thread mapping
- Hyperthreading
- Other Mechanisms

## References

1. De Sensi D., Torquati M., Danelutto M.
*A Reconfiguration Algorithm for Power-Aware Parallel Applications*
ACM Transactions on Architecture and Code Optimization, 2016

2. De Sensi D., De Matteis T., Torquati M., Mencagli G., Danelutto M.
*Bringing Parallel Patterns out of the Corner: the P$^3$ARSEC Benchmark Suite*
ACM Transactions on Architecture and Code Optimization, 2017

3. De Sensi D., De Matteis T., Danelutto M.
*Nornir: Autonomic and Power-Aware Computing for the Masses*
*Submitted* to Future Generation Computer Systems, 2017

4. De Sensi D., Torquati M., Danelutto M.
*Mammut: High-level management of system knobs and sensors*
SoftwareX, 2017

5. Torquati M., De Sensi D., Mencagli G., Aldinucci M., Danelutto M.
*Power-Aware Pipelining with Automatic Concurrency Control*
*Submitted* to Concurrency and Computation: Practice and Experience, 2017

6. Danelutto M., De Sensi D., Torquati M.
*A Power-Aware, Self-Adaptive Macro Data Flow Framework*
Parallel Processing Letters, 2016

7. Danelutto M., De Matteis T., De Sensi D., Mencagli G., Torquati M.
*P$^3$ARSEC: Towards Parallel Patterns Benchmarking*
Proceedings of the 32nd Annual ACM Symposium on Applied Computing (SAC), 2017

8. Danelutto M., De Matteis T., De Sensi D., Torquati M.
*Evaluating Concurrency Throttling and Thread Packing on SMT Multicores*
Proceedings of 25th Intl. Conf. on Parallel, Distributed and Network-Based Processing (PDP), 2017

9. De Sensi D.
*Predicting Performance and Power Consumption of Parallel Applications*
Proceedings of 24th Intl. Conf. on Parallel, Distributed and Network-Based Processing (PDP), 2016

10. Danelutto M., De Sensi D., Torquati M.
*Energy Driven Adaptivity in Stream Parallel Computations*
Proceedings of 23rd Intl. Conf. on Parallel, Distributed and Network-Based Processing (PDP), 2015

11. De Sensi D., De Matteis T., Danelutto M.
*Nornir: A Customisable Framework for Autonomic and Power-Aware Applications*
Proceedings of Euro-Par Workshops, 2017

12. Aldinucci M., Danelutto M., De Sensi D., Mencagli G., Torquati M.
*Towards Power-Aware Data Pipelining on Multicores*
Proceedings of HLPP: International Workshop on High-Level Parallel Programming, 2017

13. Danelutto M., De Sensi D., Torquati M.
*A Power-Aware, Self-Adaptive Macro Data Flow Framework*
Proceedings of HLPP: International Workshop on High-Level Parallel Programming, 2016

14. De Sensi D., Kilpatrick P., Torquati M.
*State-aware Concurrency Throttling*
Proceedings of Parallel Computing (ParCo) Conference, 2017

## Source Code

a. http://github.com/ParaGroup/p3arsec
b. http://github.com/DanieleDeSensi/knarr
c. http://github.com/DanieleDeSensi/nornir
d. http://github.com/DanieleDeSensi/mammut

## Contacts

**Personal page**: http://pages.di.unipi.it/desensi/
**e-mail**: desensi@di.unipi.it