Benchmarking the state-of-the-art Task Execution Frameworks of Many-Task Computing

Thomas Dubucq – Tony Forlini – Virgile Landeiro Dos Reis – Isabelle Santos

Introduction

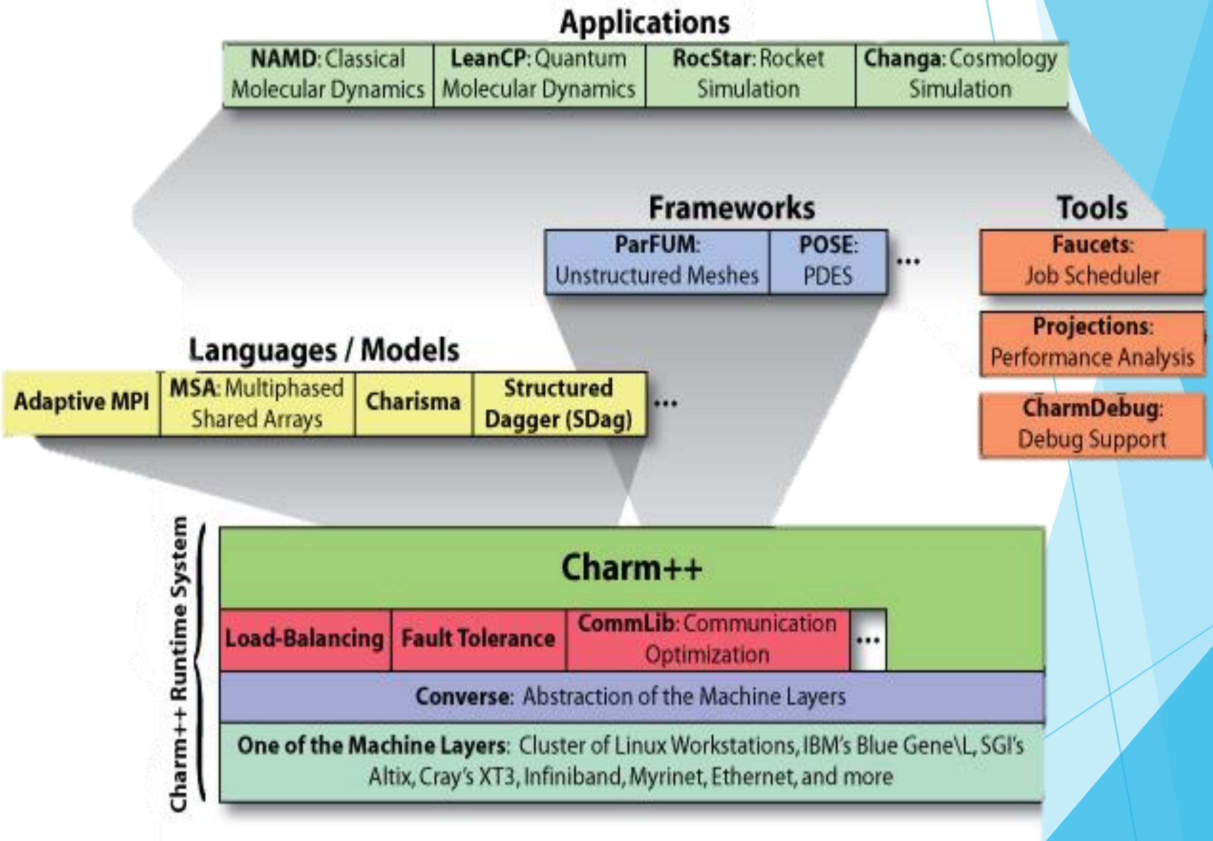
- Systems:
  - Charm++
  - HPX
  - Legion
  - MATRIX
  - Sparrow
  - Swift/T

- Testbeds:
  - From 1 to 128 nodes
  - C3.large instances
  - 2 cores per node
  - Weak scaling

- Metrics:
  - Throughput
  - Efficiency
  - Task latency

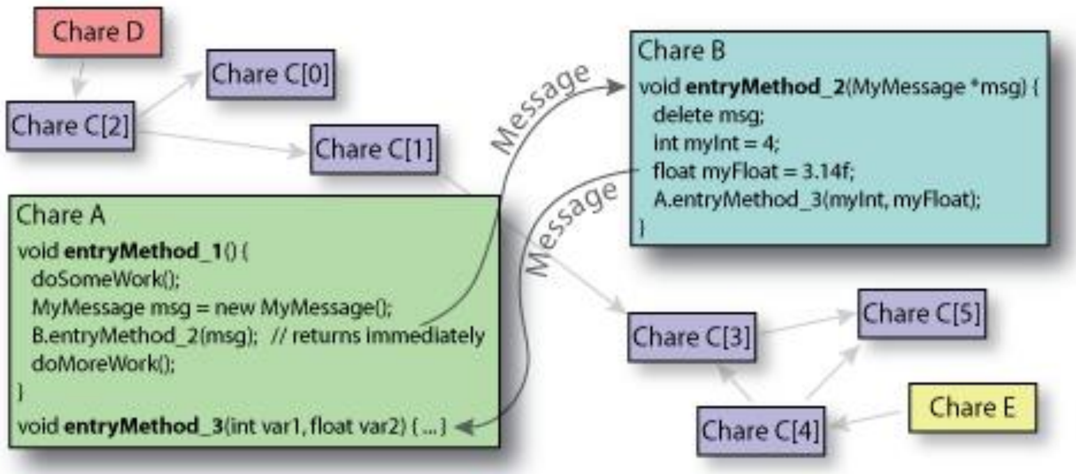
Charm++

- Explicit parallelism
- Applications:
  - NAMD molecular dynamics
  - LeanCP quantum molecular dynamics
- Tools:
  - Faucets job scheduler
  - CharmDebug
  - Projections


Applications
NAMD: Classical Molecular Dynamics
LeanCP: Quantum Molecular Dynamics
RocStar: Rocket Simulation
Changa: Cosmology Simulation
Frameworks
ParFUM: Unstructured Meshes
POSE: PDES
...
Tools
Faucets: Job Scheduler
Projections: Performance Analysis
CharmDebug: Debug Support
Languages / Models
Adaptive MPI
MSA: Multiphased Shared Arrays
Charisma
Structured Dagger (SDag)
...
Charm++ Runtime System
Charm++
Load-Balancing
Fault Tolerance
CommLib: Communication Optimization
...
Converse: Abstraction of the Machine Layers
One of the Machine Layers: Cluster of Linux Workstations, IBM's Blue Gene\L, SGI's Altix, Cray's XT3, Infiniband, Myrinet, Ethernet, and more

Charm++

- Asynchronous message passing parallel programming paradigm
- Tasks defined as chare objects
- Chare objects communicate with each other by sending messages



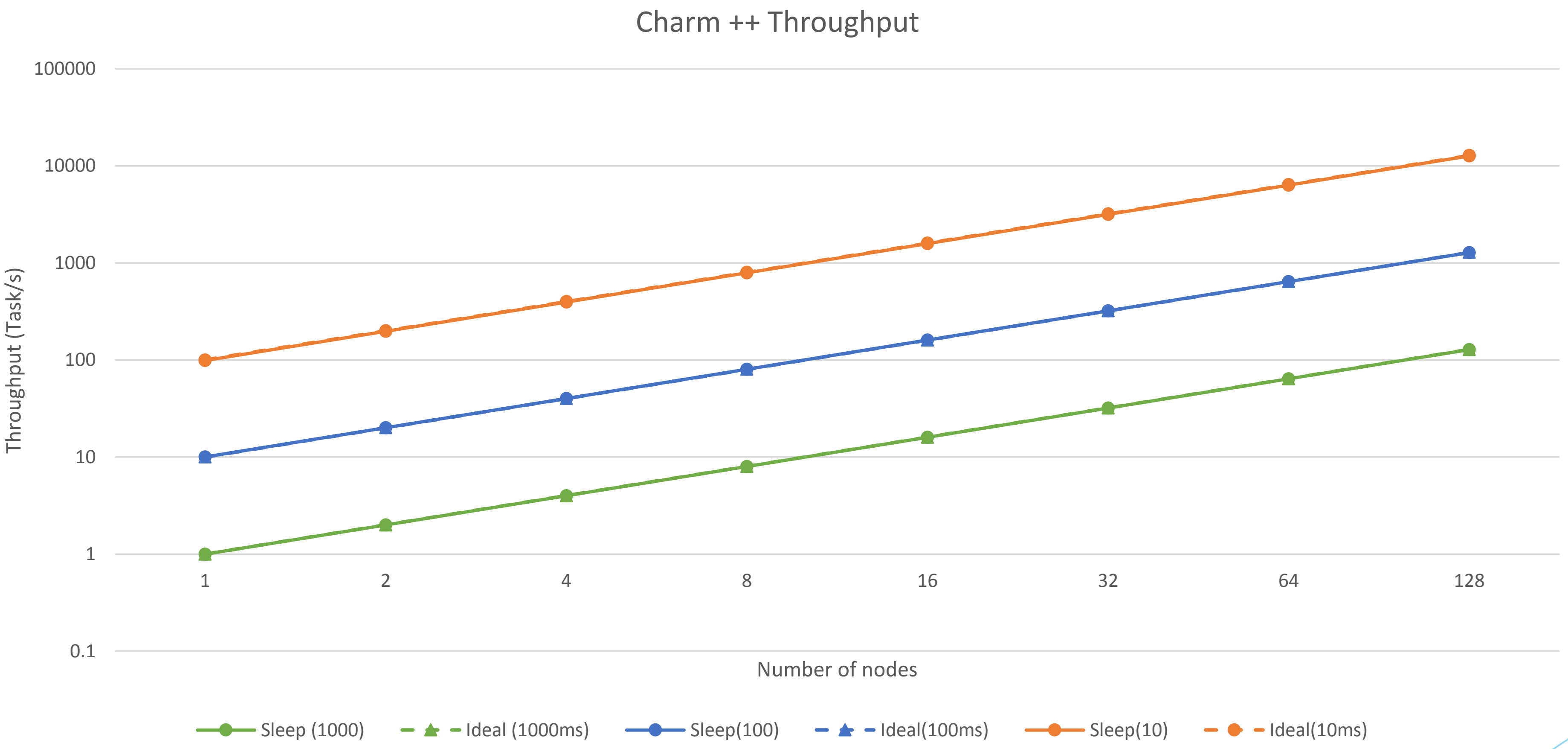
Chare A

```
void entryMethod_1() {
  doSomeWork();
  MyMessage msg = new MyMessage();
  B.entryMethod_2(msg);   // returns immediately
  doMoreWork();
}
void entryMethod_3(int var1, float var2) { ... }
```

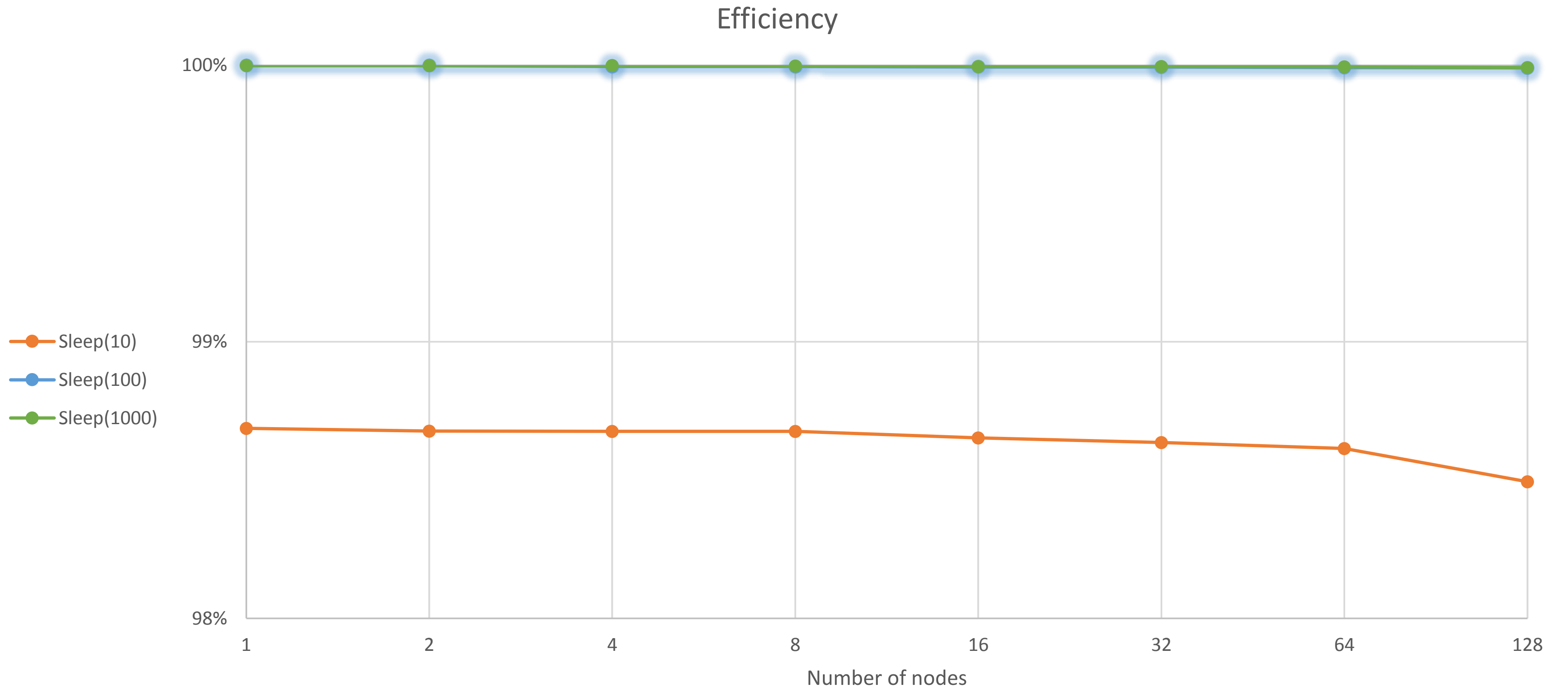
Chare B

```
void entryMethod_2(MyMessage *msg) {
  delete msg;
  int myInt = 4;
  float myFloat = 3.14f;
  A.entryMethod_3(myInt, myFloat);
}
```

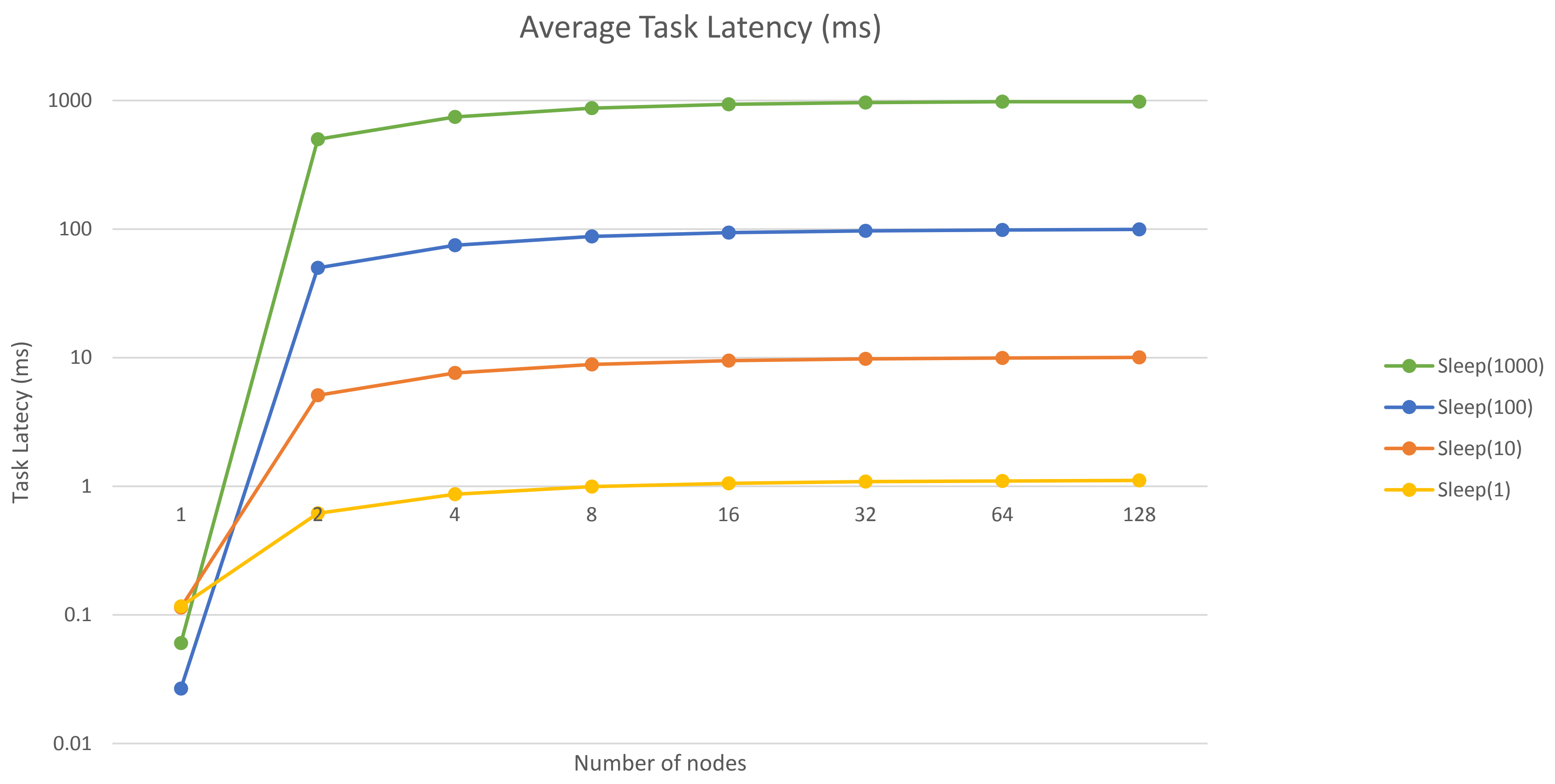
Charm++ - Results



Charm++ - Efficiency

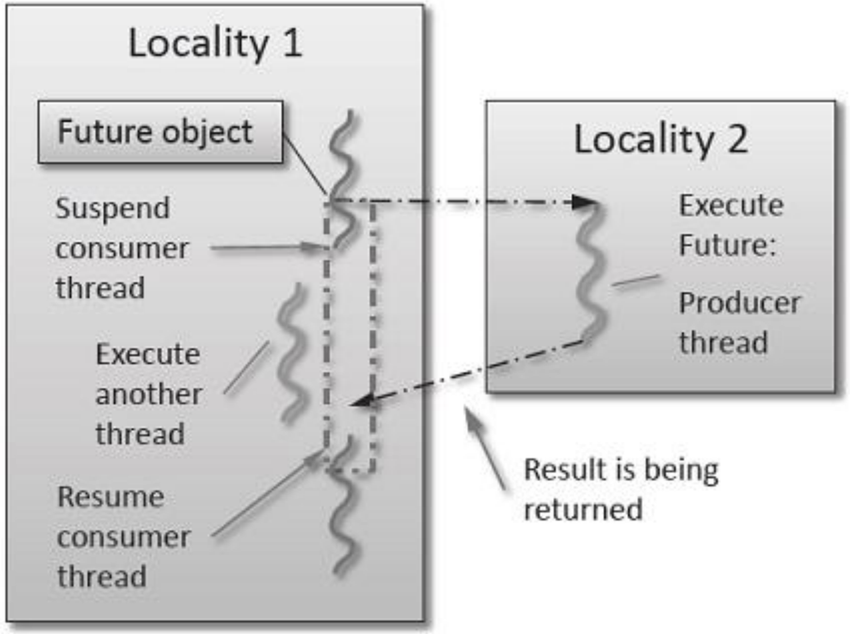


Charm++ - Results



HPX (High Performance ParalleX)

- Boost C++ runtime system for parallel and distributed applications
- ParalleX execution model
  - hide latency by switching tasks
  - fine-grained tasks
  - reduce overhead
- Futures
  - delayed computation
  - reader thread & producer thread



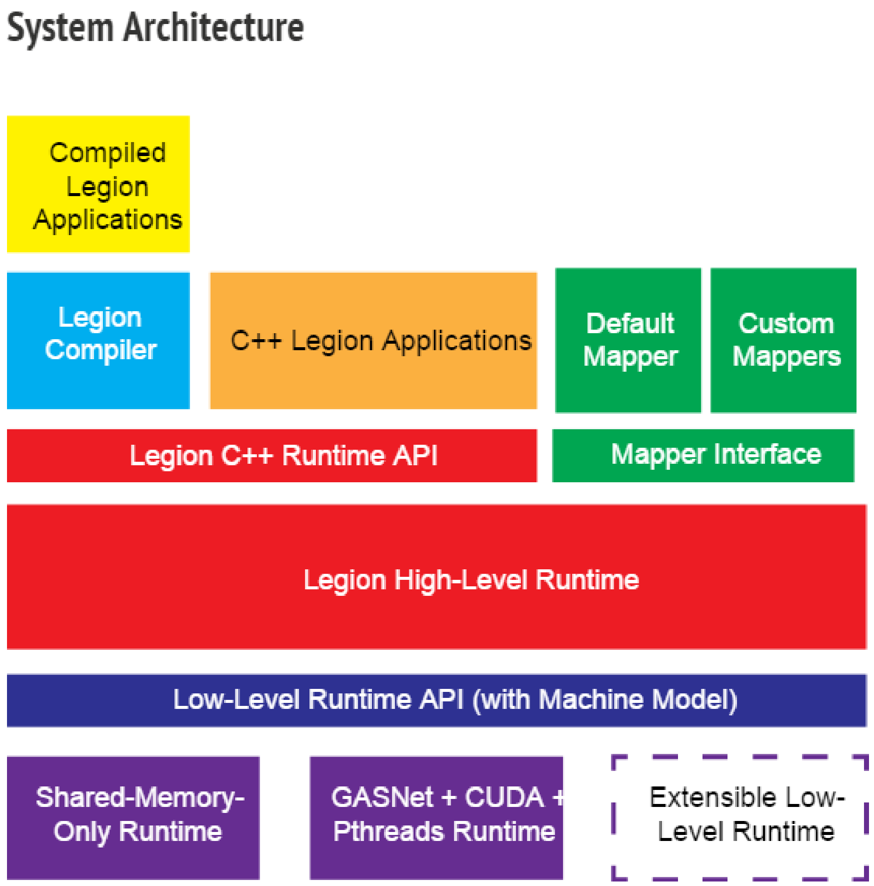
HPX: Challenges

- Never previously deployed on EC2
- Incomplete documentation
  - unmentioned dependencies
  - typos in command lines
- Long compilation time
- Memory requirements
- Version compatibility among dependencies

Legion

- Programming model for heterogeneous, distributed machines
- Built on top of GASNet communication system
- Heterogeneous
  - Mixed CPUs and GPUs
- Distributed
  - Large spread and variability of communication latencies
- Implicit parallelism

Legion



System Architecture

Compiled Legion Applications

Legion Compiler

C++ Legion Applications

Default Mapper

Custom Mappers

Legion C++ Runtime API

Mapper Interface

Legion High-Level Runtime

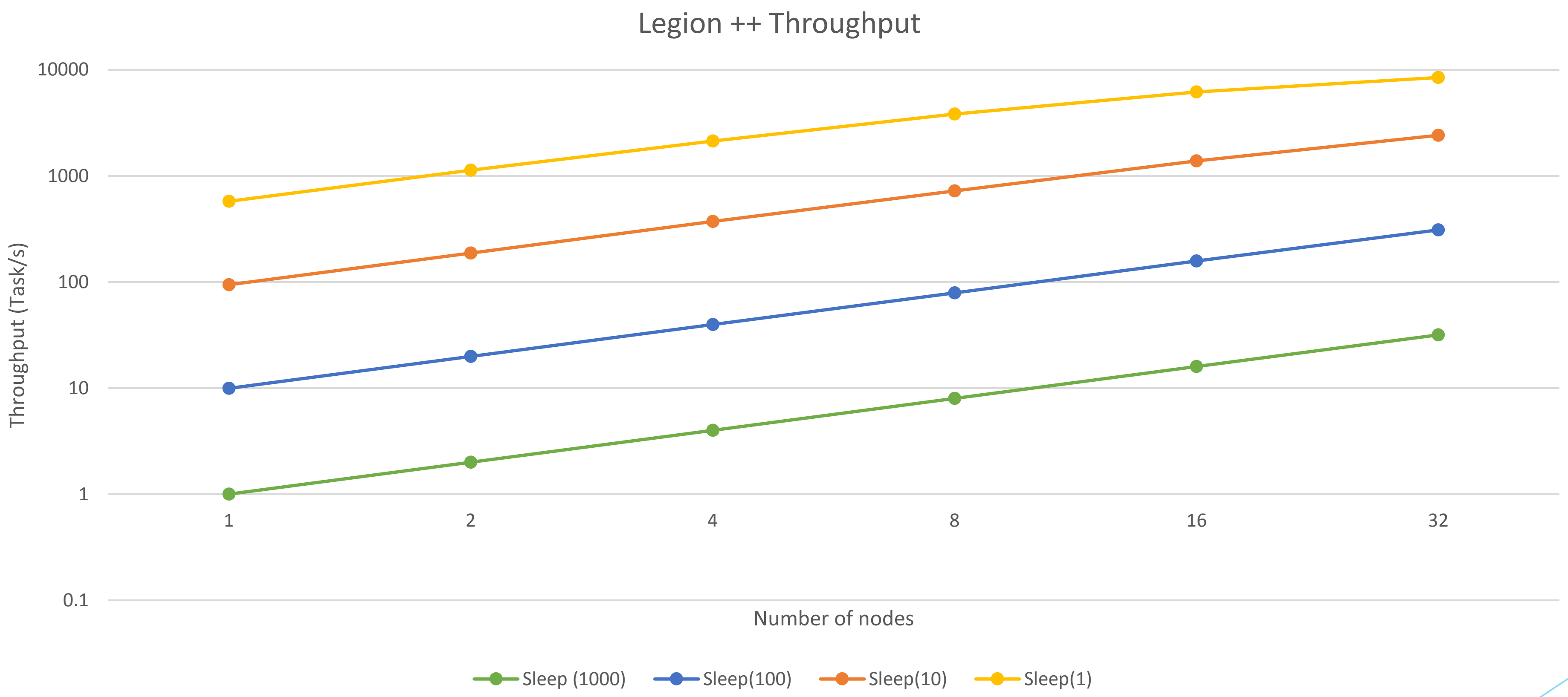
Low-Level Runtime API (with Machine Model)

Shared-Memory-Only Runtime

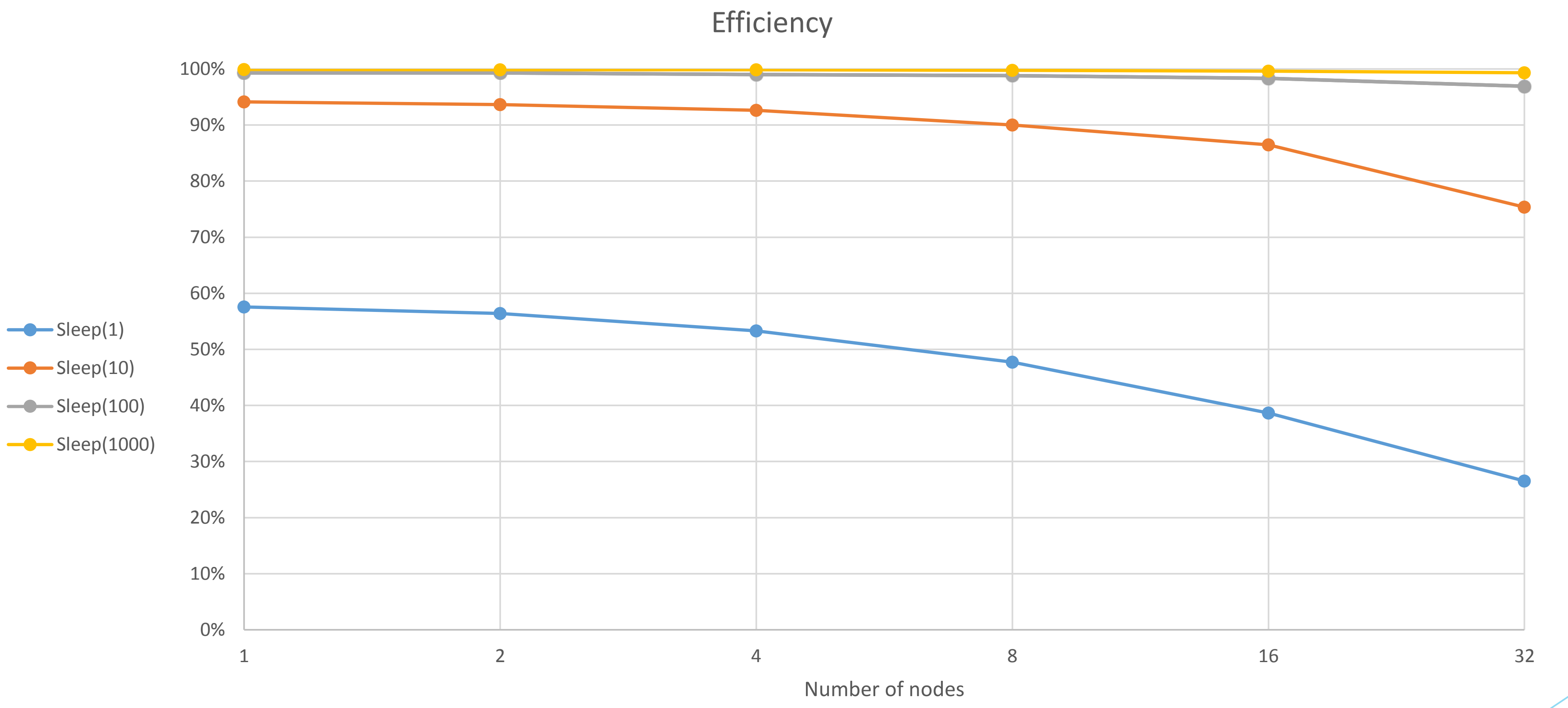
GASNet + CUDA + Pthreads Runtime

Extensible Low-Level Runtime

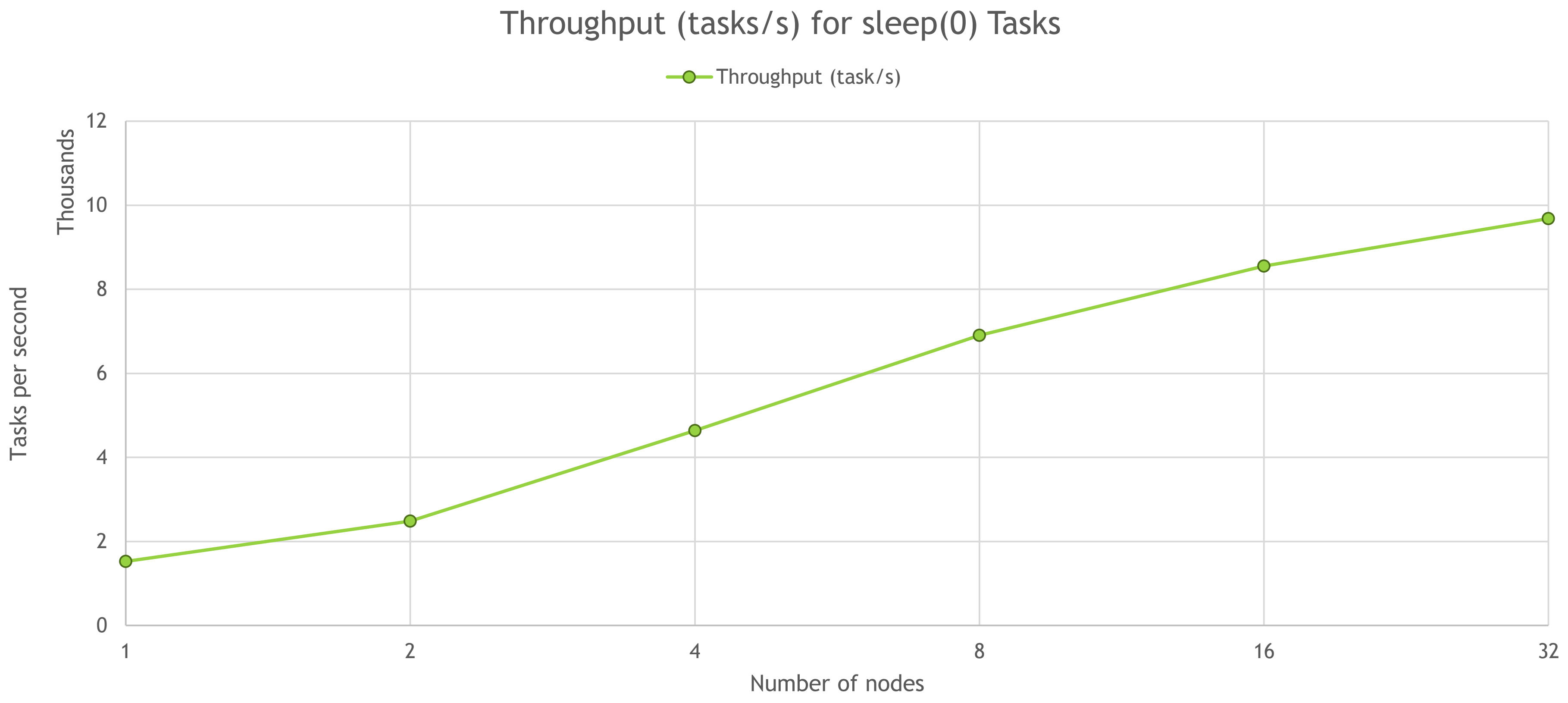
Legion - Results



Legion - Results



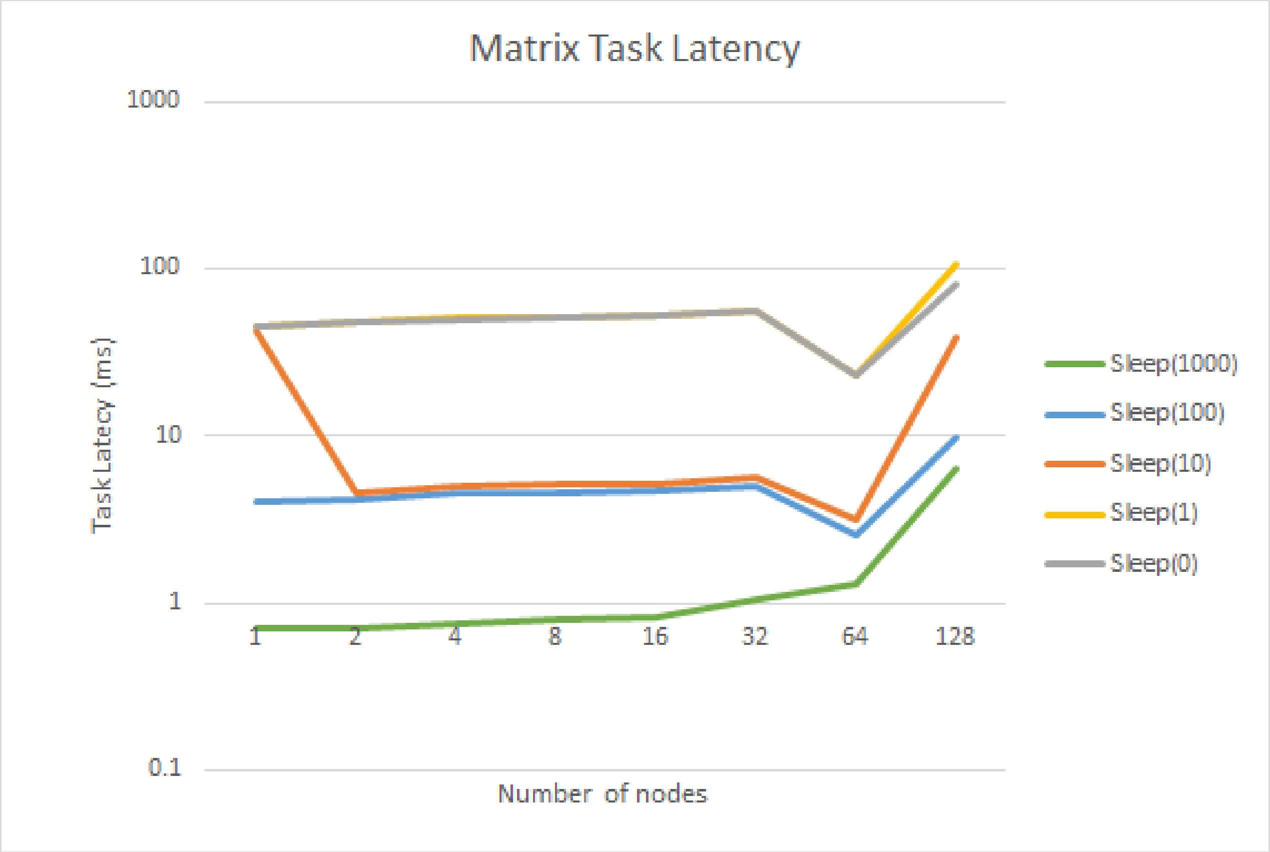
Legion - Results



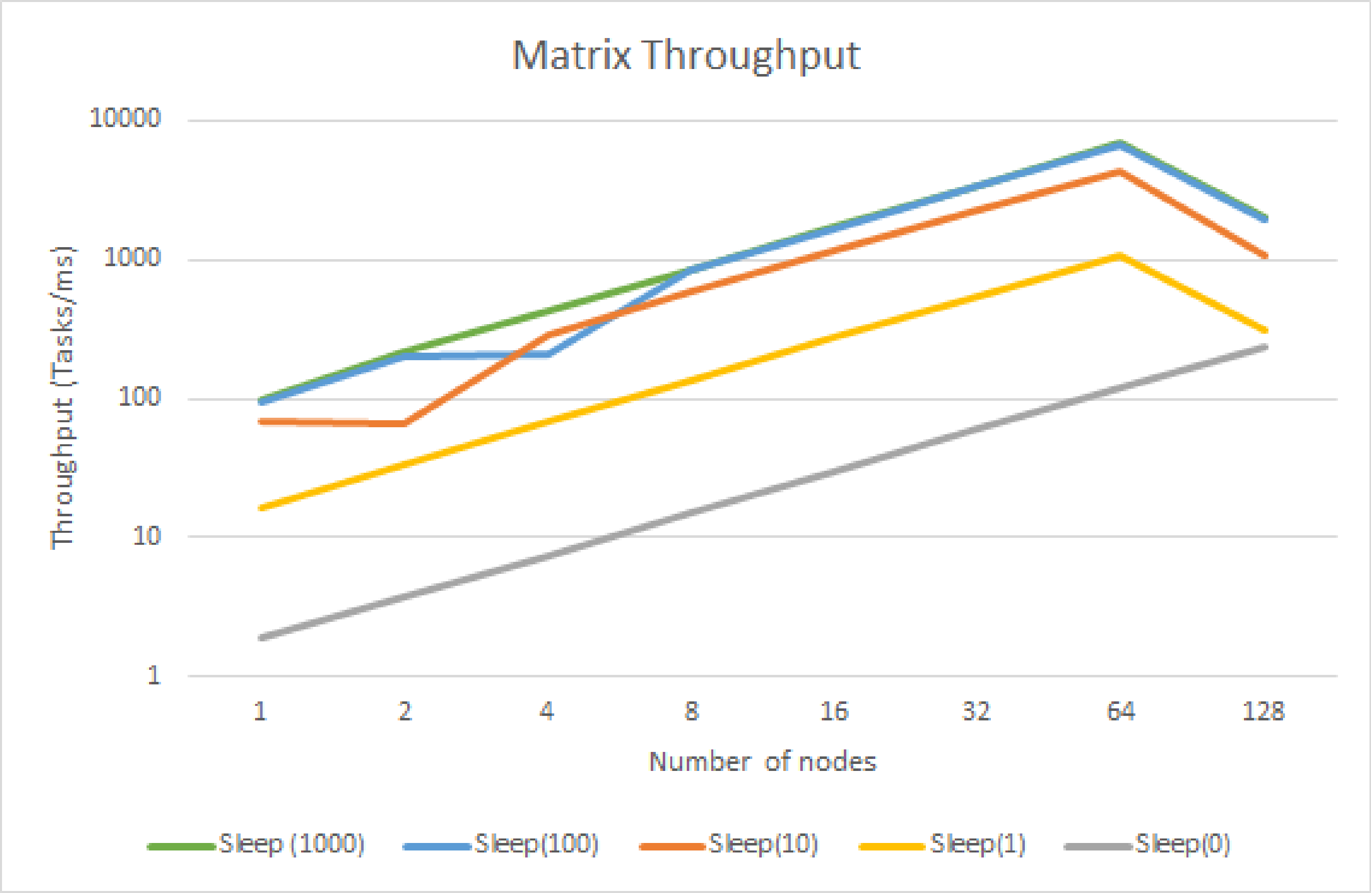
MATRIX

- Distributed data-aware execution fabric
- Work stealing
- Built on top of ZHT
- Supports MTC and HPC workloads
- Coded in C++

MATRIX - Results

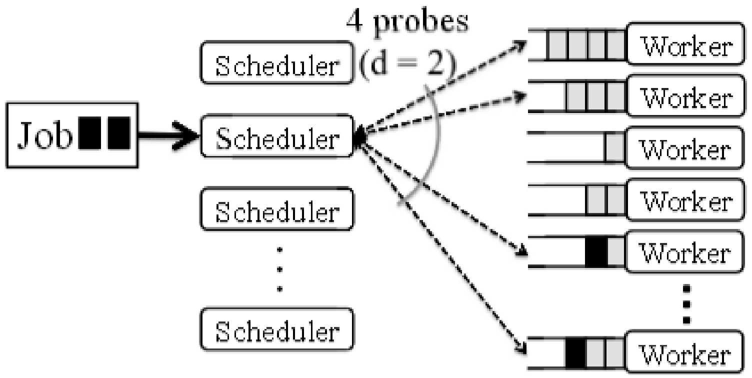


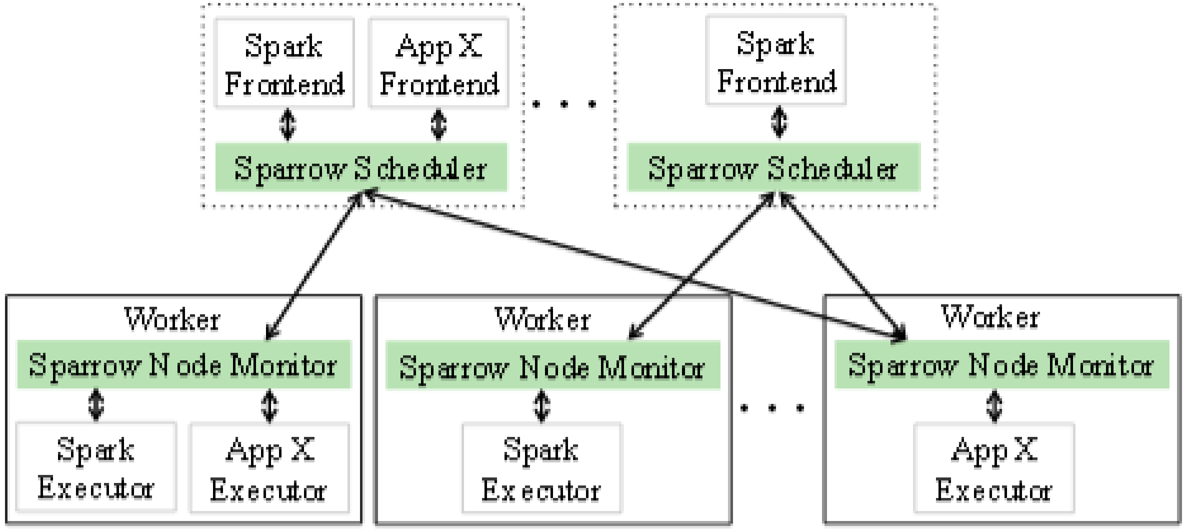
MATRIX - Results



Sparrow

- Stateless distributed scheduler
- Late binding
- Batch sampling
- Coded in Java

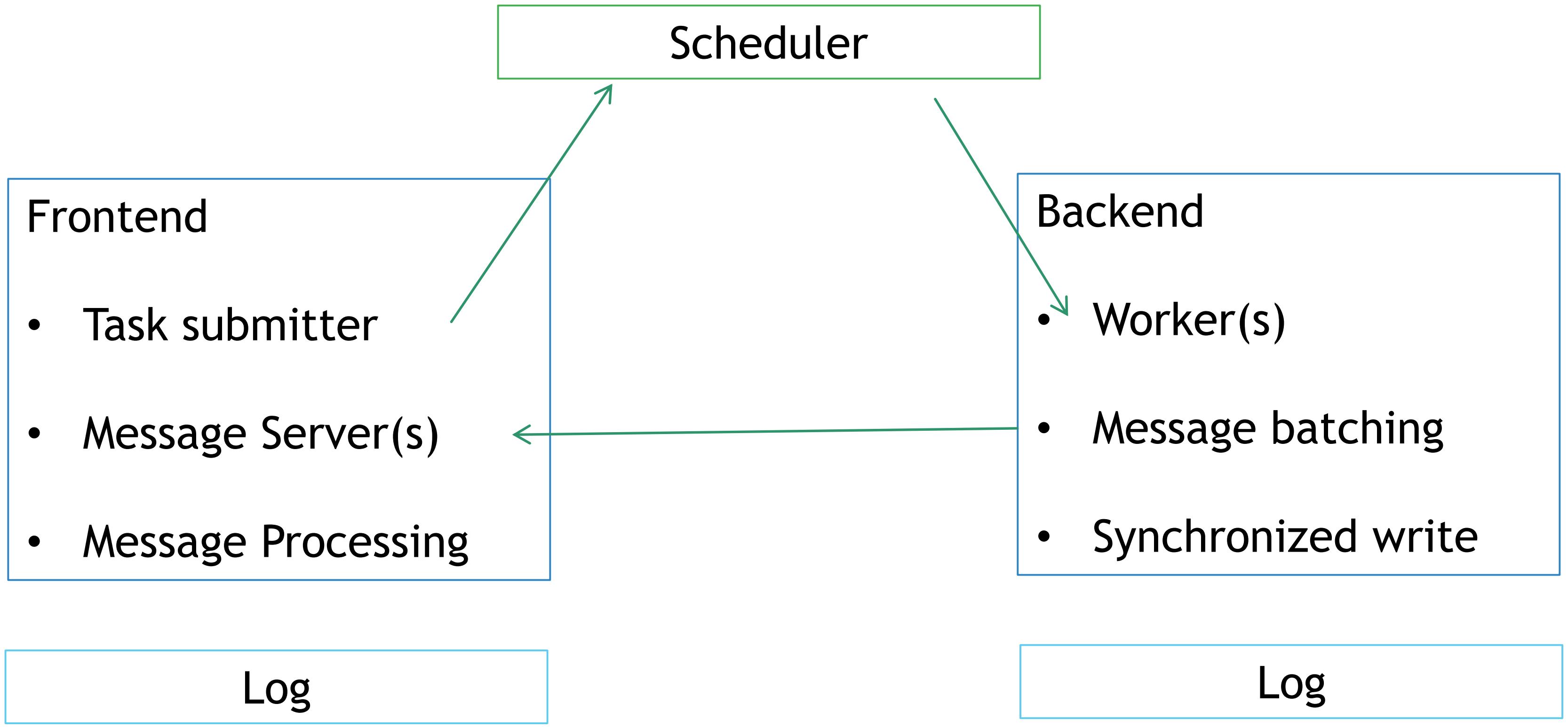




Sparrow - Challenges

- Backend and Frontend provided in the code source don't match the benchmark metrics
- Necessity of task completion acknowledgement
- Dense workload handling

Sparrow



Scheduler

Frontend

- Task submitter
- Message Server(s)
- Message Processing

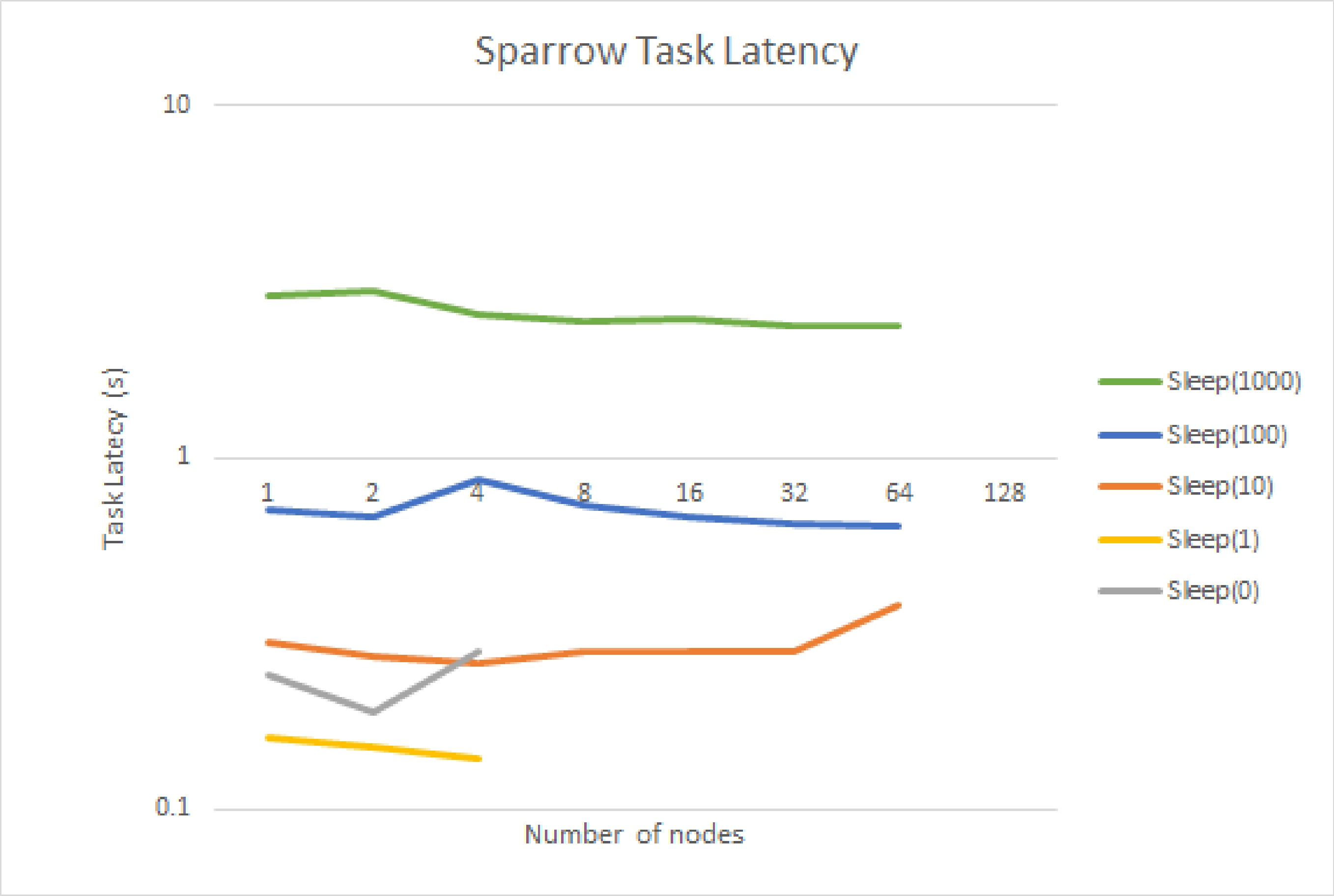
Backend

- Worker(s)
- Message batching
- Synchronized write

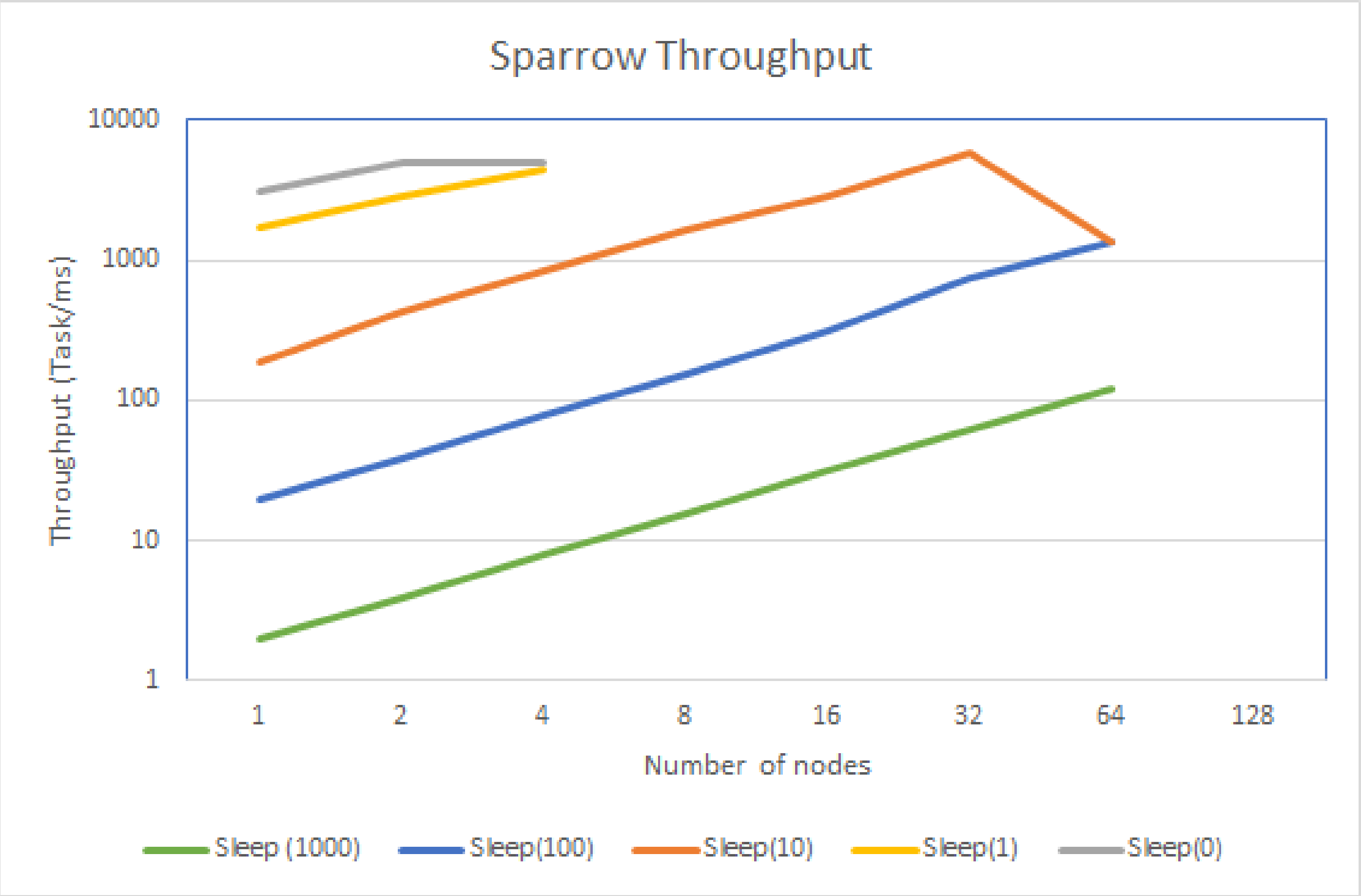
Log

Log

Sparrow - Results

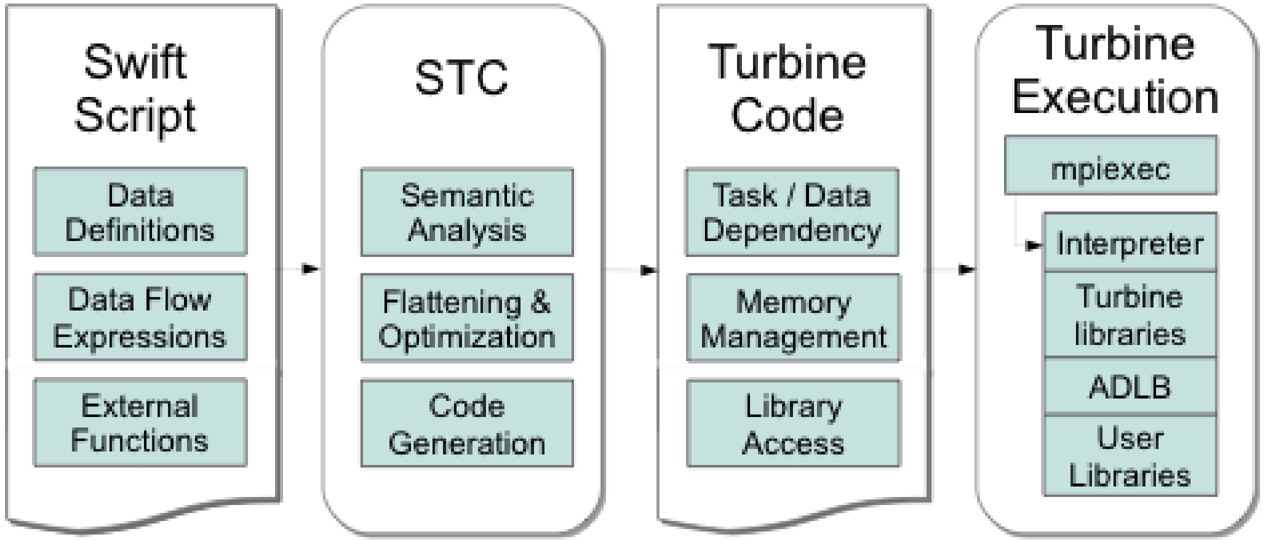


Sparrow - Results



Swift/T

- Workflow system
- Built on top of MPI
- Highly programmable

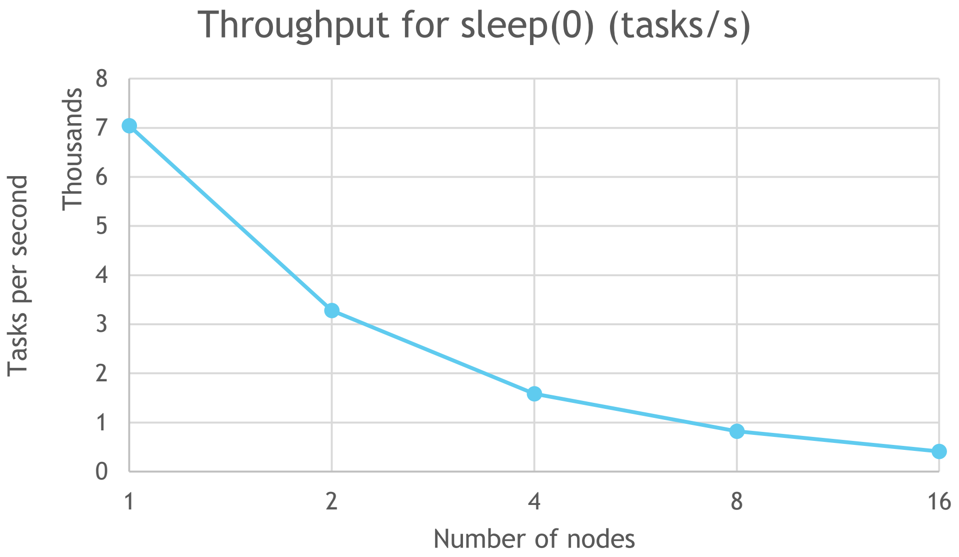


- Improve from Swift/K by introducing a parallel evaluation of complex scripts
- Two main tools:
  - STC: Java-based compiler that produces Turbine code
  - Turbine: execution combining MPI and ADLB

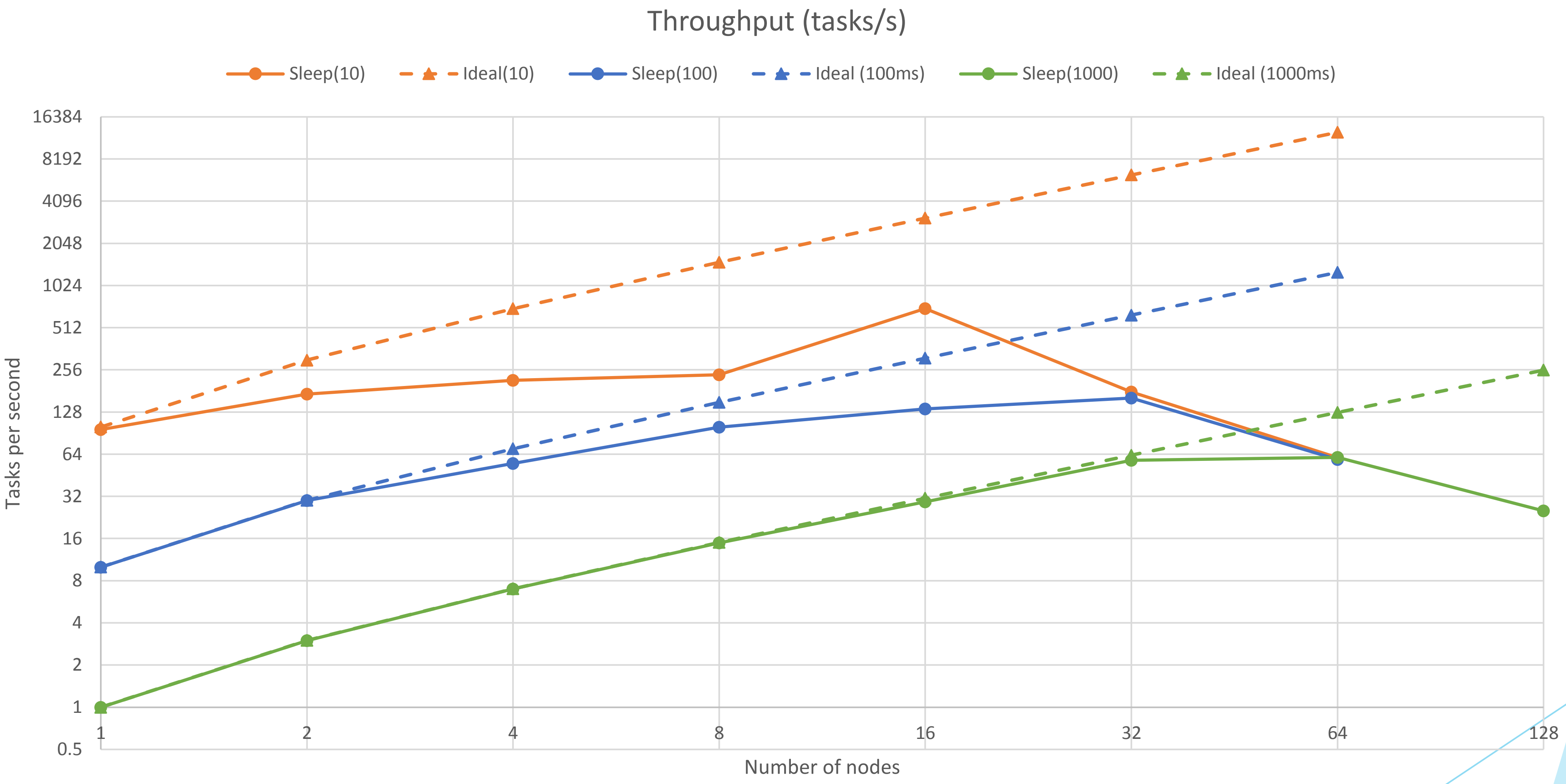
Swift/T - Challenges

- Very sparse documentation to deploy on EC2 but very reactive support by Justin M. Wozniak.
- Sleep(0) tasks do not scale. Have not found why yet but suspect the following piece of code:

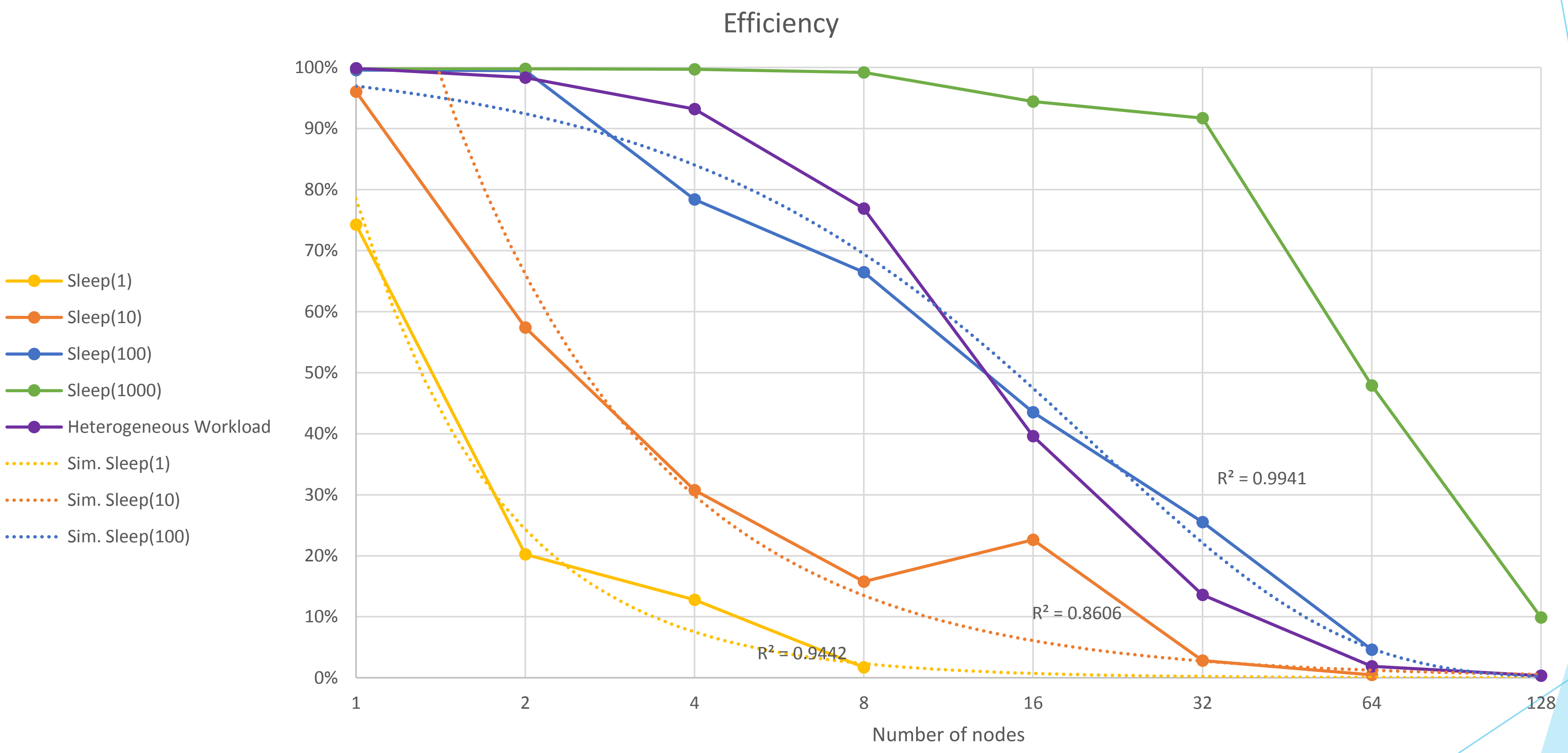
```
@dispatch=WORKER
(void v) sleep(float seconds) "turbine" "0.0.4" "sleep" [
  "if { <<seconds>> > 0 } { after [ expr {round(<<seconds>> * 1000)} ] }“
];
```



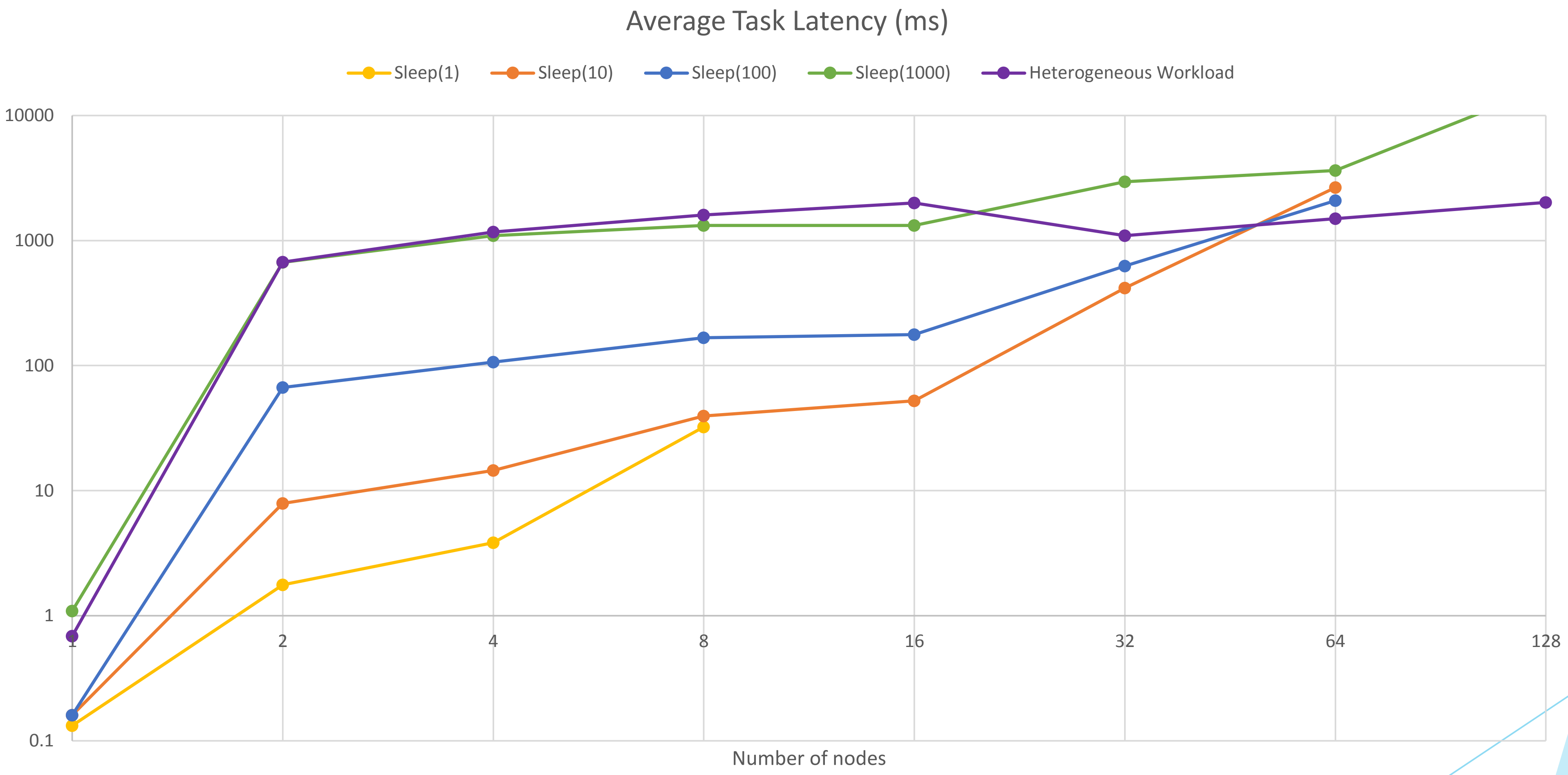
Swift/T - Results



Swift/T - Results



Swift/T - Results



Conclusion

- Performance ranking:
  - Charm++ ↑ best
  - Legion
  - Sparrow
  - MATRIX
  - Swift/T worst
- Non evaluated systems:
  - HPX