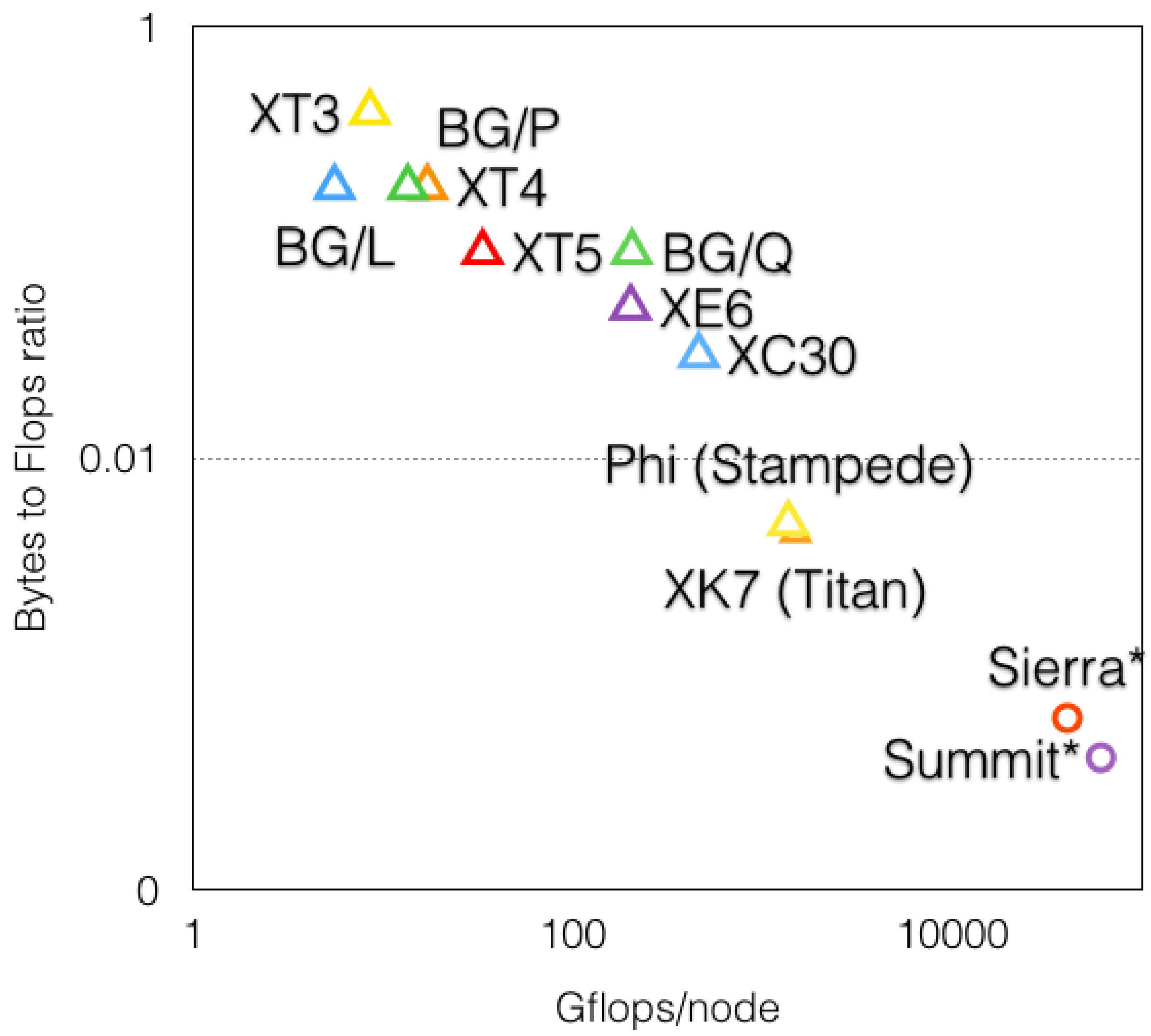
Flexible Hierarchical Execution of Parallel Task Loops

Michael Robson, Villanova University

Kavitha Chandrasekar, University of Illinois Urbana-Champaign

Injection Bandwidth vs CPU speeds



Motivation

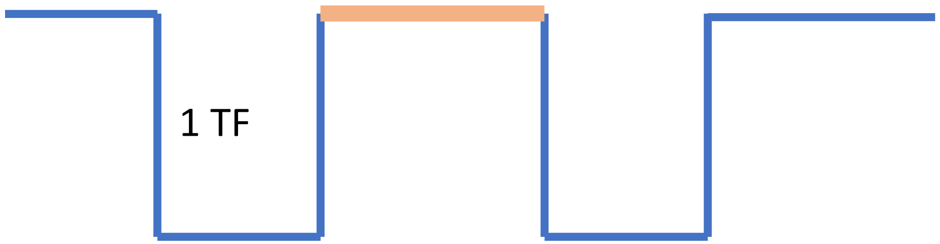
- Trend:
  - Deeper nodes
  - Thinner pipes
- Accelerators (e.g. GPUs)
- Increased Programmer effort

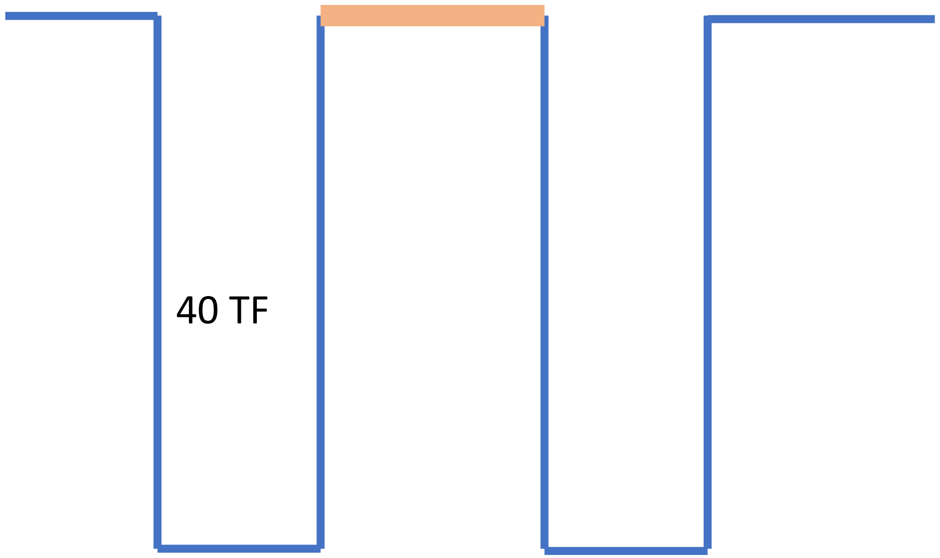
Year	Machine	Linpack (FLOPs)	FLOPs/ Local	FLOPs/ Remote
1988	Cray YMP	2.1 Giga	0.52	0.52
1997	ASCI Red	1.6 Tera	8.3	20
2011	Road-runner	1.0 Peta	6.7	170
2012	Sequoia	17 Peta	32	160
2013	Titan	18 Peta	29	490
2018	Summit	122 Peta	37	1060
2011	K-Comp	11 Peta	15	95
2013	Tianhe-2	34 Peta	22	1500
2016	Sunway	93 Peta	130	1500
2021	TBD	1.0 Exa	80	3200
2021	TBD	1.0 Exa	300	10000

Fat Nodes

First law of holes:

- If you find yourself in a hole, stop digging!

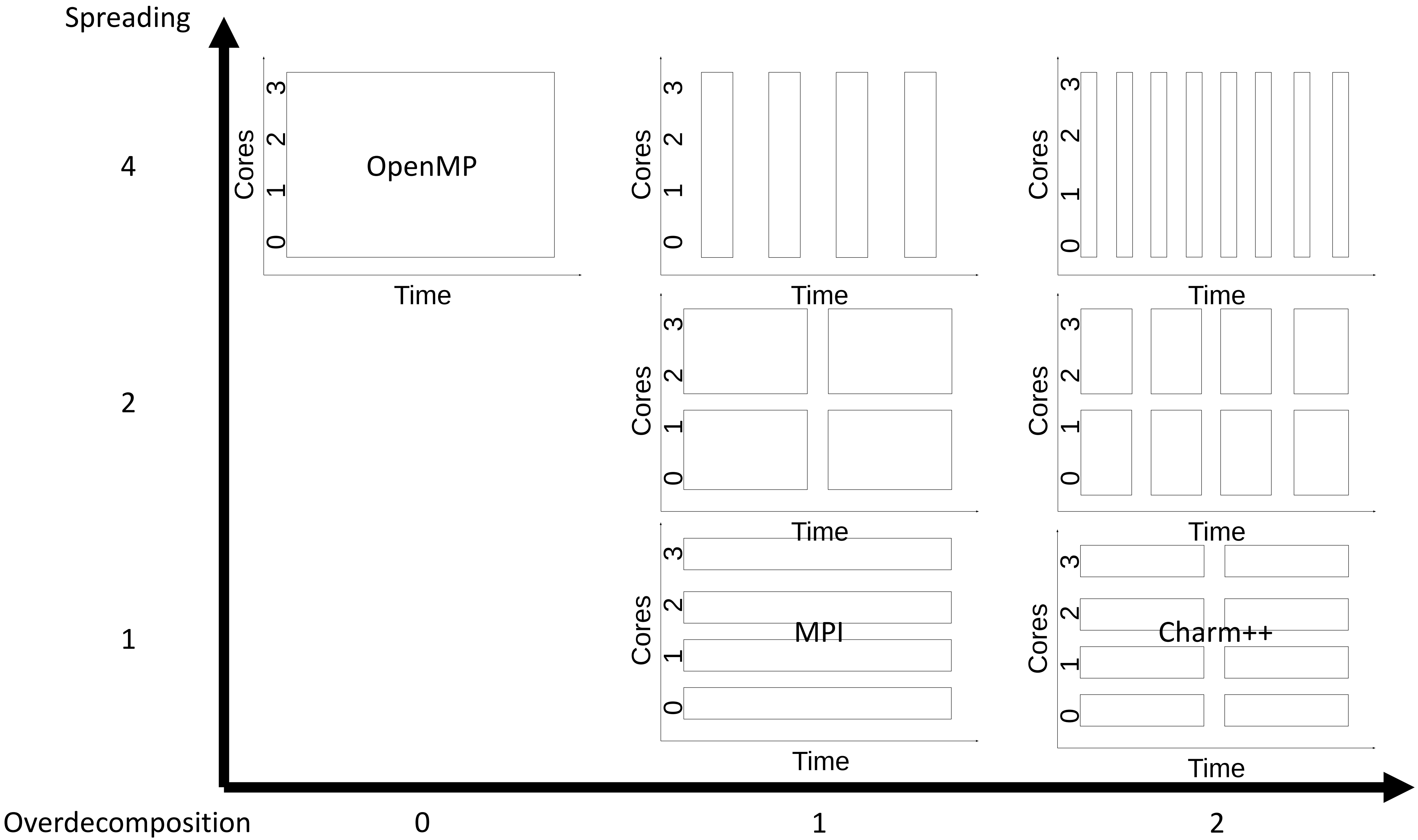




We are digging ourselves deeper into a node

Main Idea: *Spreading* Work Across Cores

- Speed up individual calculations via OpenMP
- FLOPs are cheap, need to inject early
- Better communication, computation overlap


Spreading
4
2
1
Overdecomposition
0
1
2
OpenMP
MPI
Charm++
Cores
Time
0
1
2
3

Motivation

New Axes of Optimization

- Problem Size Decomposition (Grain Size)
- Resources Assigned to a Task (Spreading)

Experimental Setup

- Charm Build
  - **Separate processes** (Non-SMP mode)
  - –O3 –with-production
  - PAMI-LRTS communication layer
- Five Runs
  - OpenMP Threads (Spreading) = 1, 2, etc
  - Grid Size = 178848^2 doubles (~90%)
  - Block Size = 7452, various
  - Chares (Objects) = 24^2
  - Iterations = 10-100
  - Nodes = 4

OpenMP Pragmas

- Schedule - Static
- Chunk Size (Iterations)
  - Default (Block / Cores)
  - 1
  - 16
  - 512
- Collapse

Machines

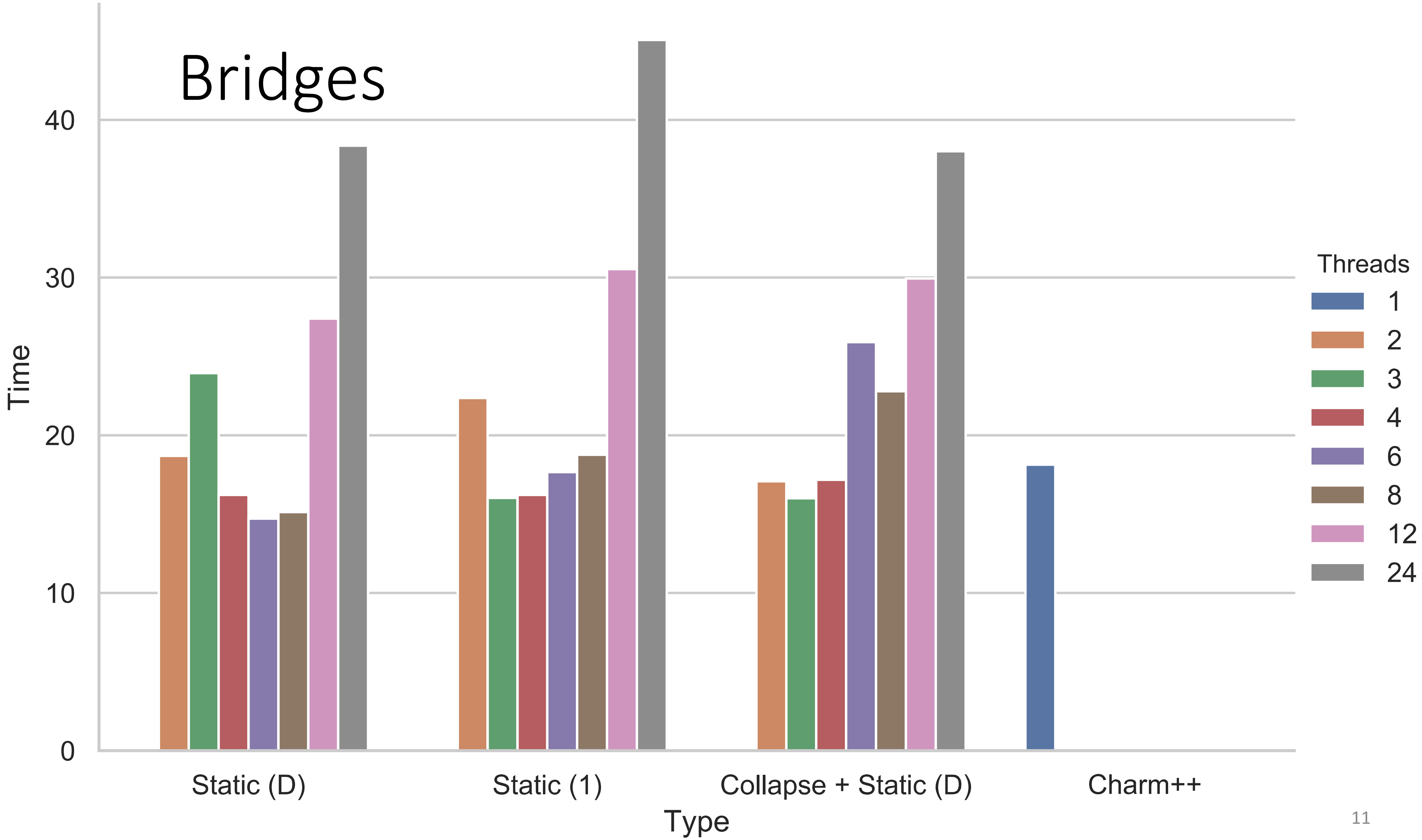
Bridges (PSC)

- 2 x 14-core Haswell E5-2695
- 128 GB DDR4

Summit (ORNL)

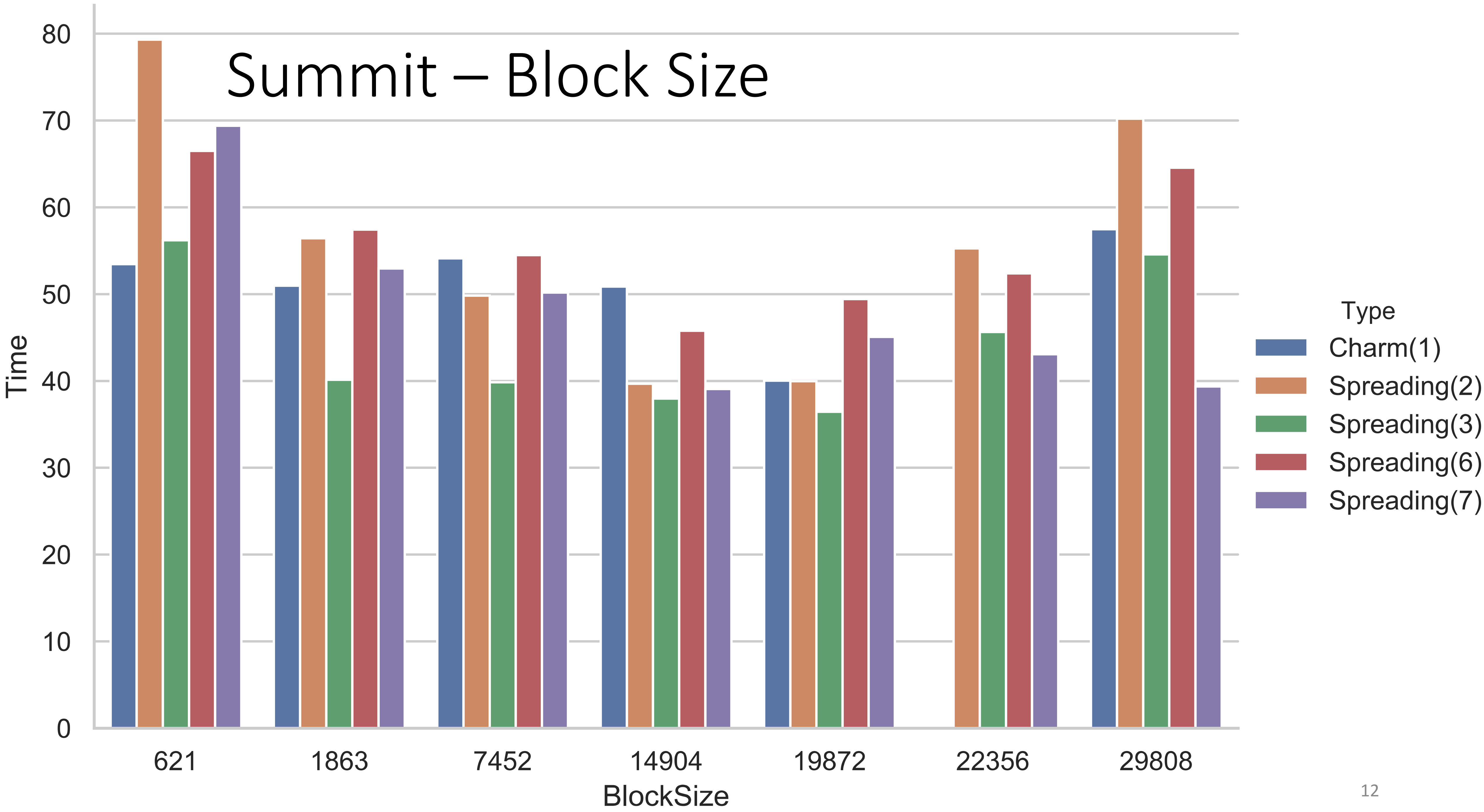
- 2 x 22-core IBM Power9
- 512 GB DDR4

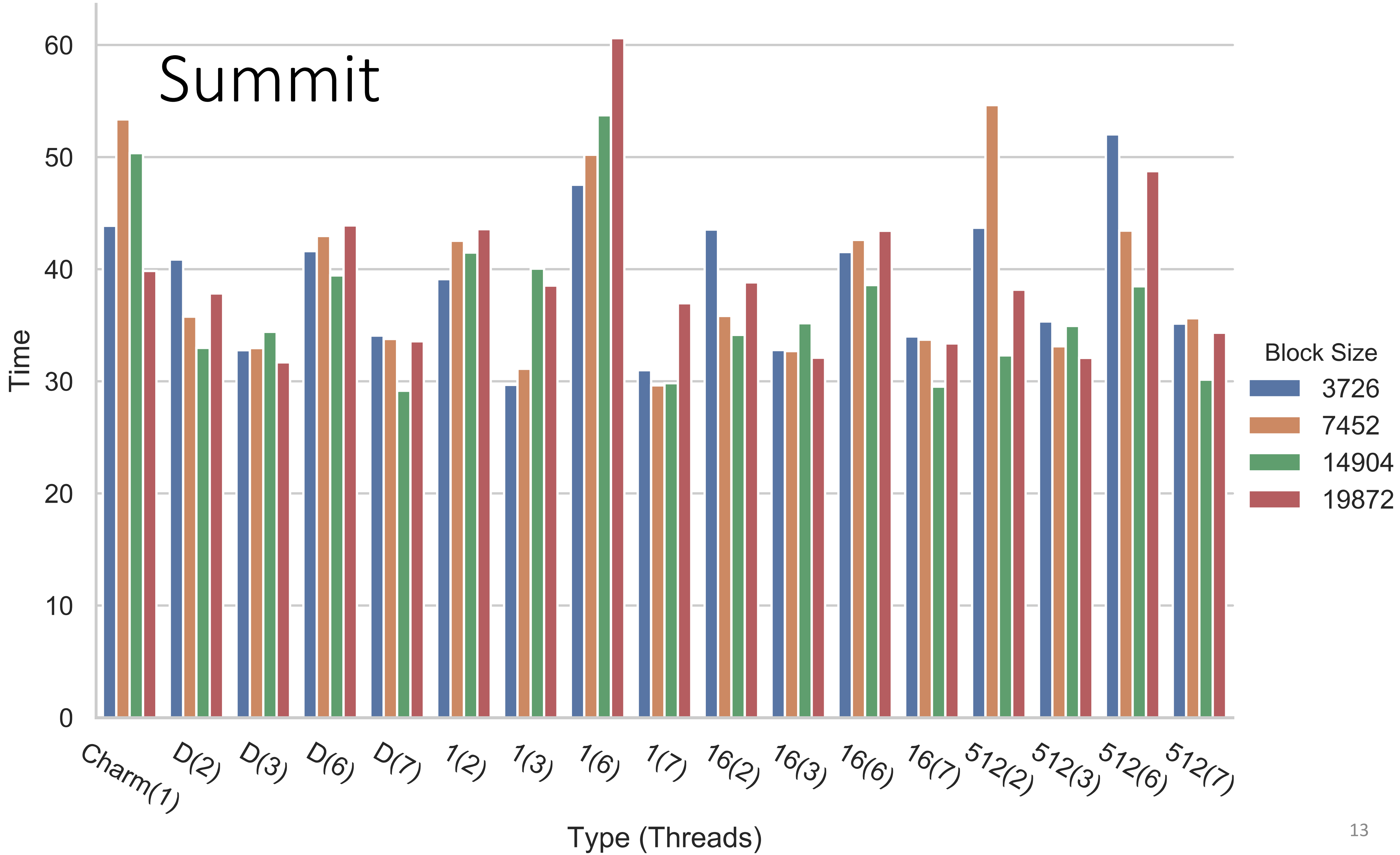
Bridges



Time
0
10
20
30
40
Static (D)
Static (1)
Collapse + Static (D)
Charm++
Type
Threads
1
2
3
4
6
8
12
24

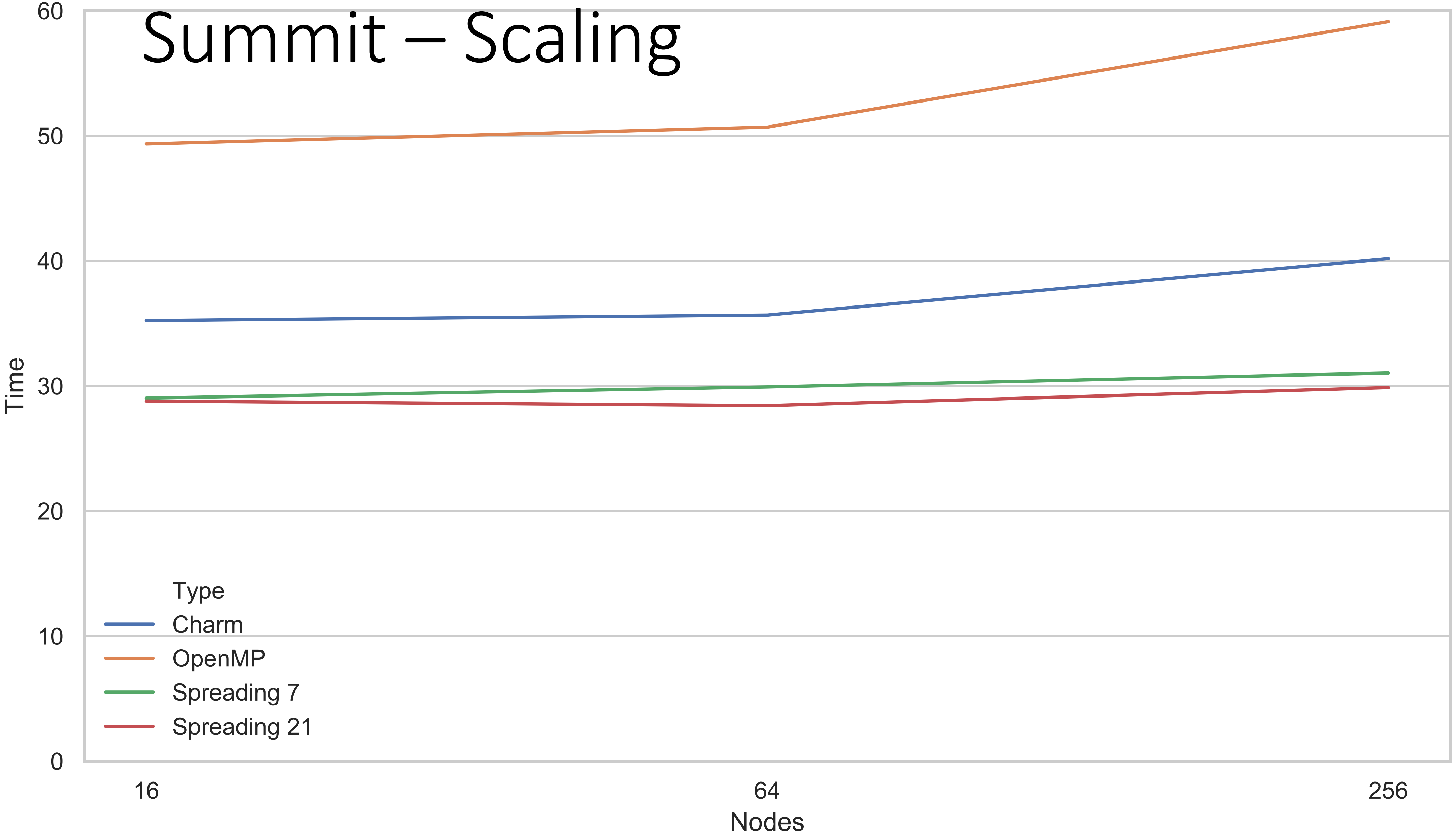
Summit – Block Size





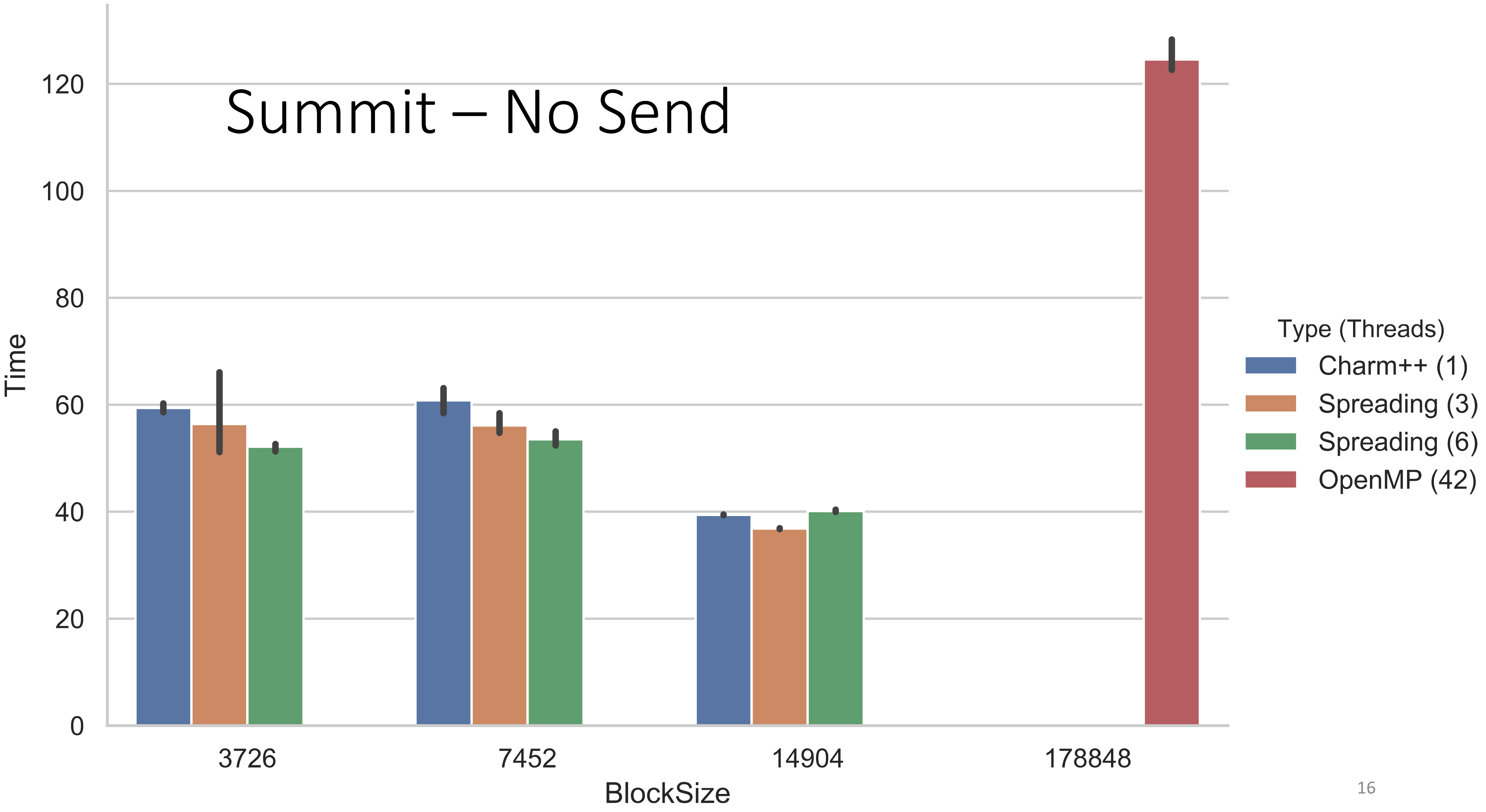
Summit
Time
60
50
40
30
20
10
0
Charm(1)
D(2)
D(3)
D(6)
D(7)
1(2)
1(3)
1(6)
1(7)
16(2)
16(3)
16(6)
16(7)
512(2)
512(3)
512(6)
512(7)
Type (Threads)
Block Size
3726
7452
14904
19872

Summit – Scaling



What happens when we eliminate communication?

i.e. are effects just from improved caching?

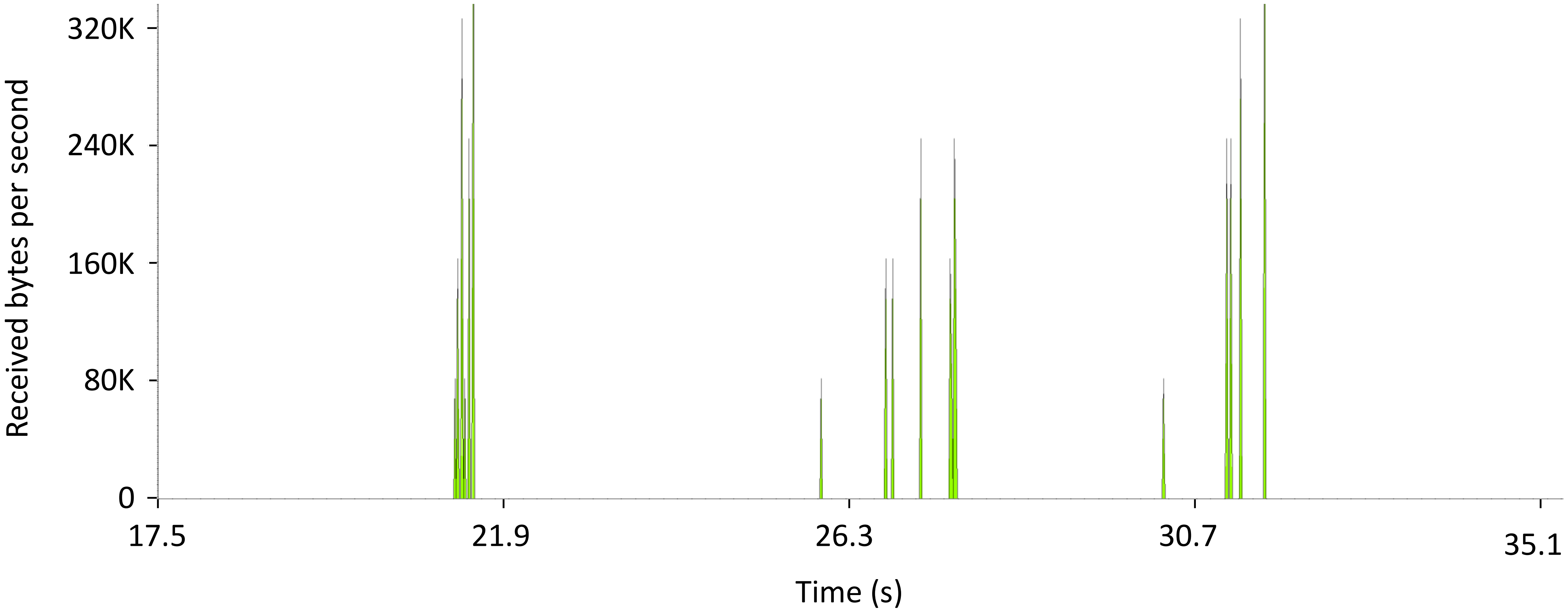


Summit – No Send
Time
120
100
80
60
40
20
0
3726
7452
14904
178848
BlockSize
Type (Threads)
Charm++ (1)
Spreading (3)
Spreading (6)
OpenMP (42)

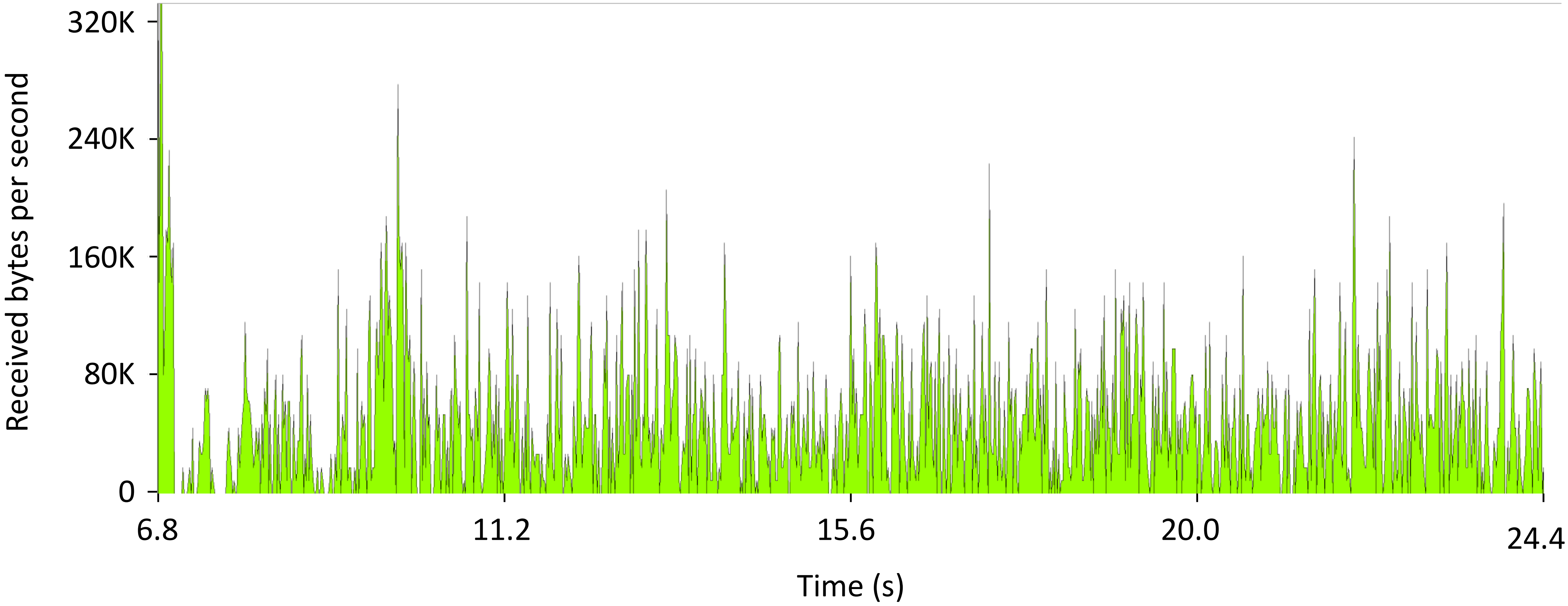
Lets look at communication performance…

using projections.

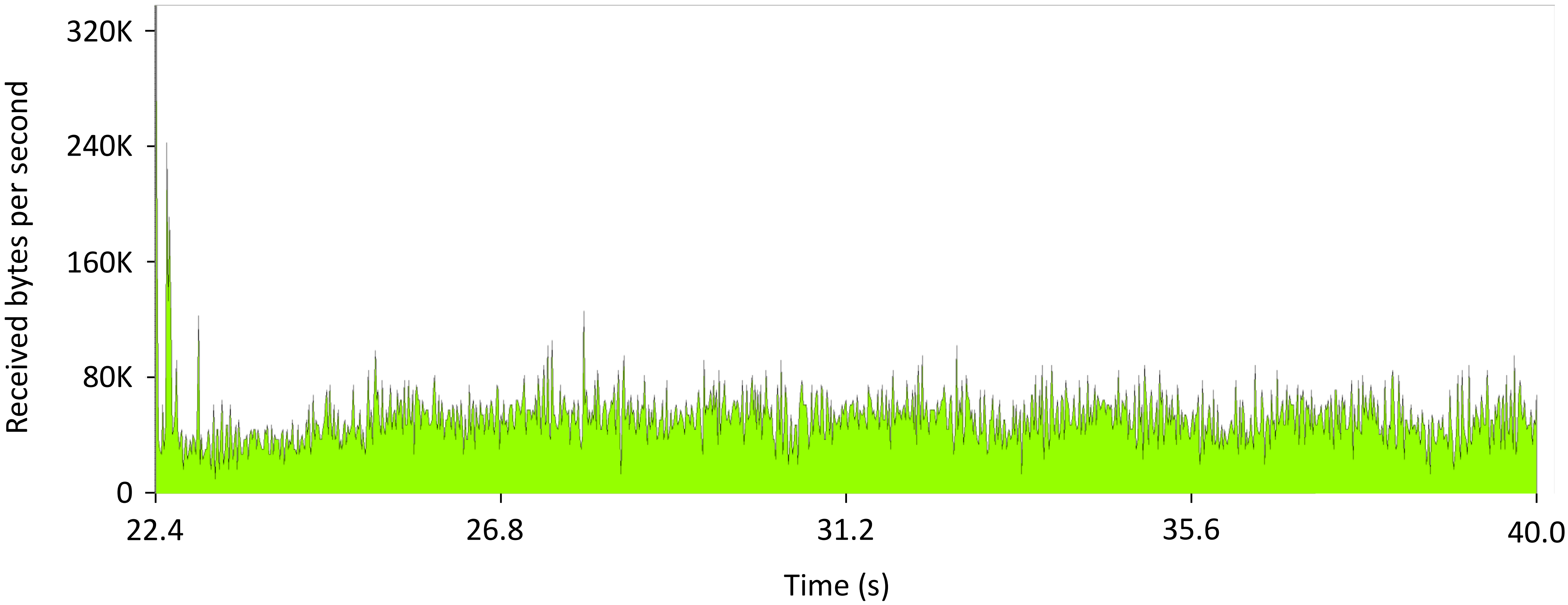
OpenMP Baseline

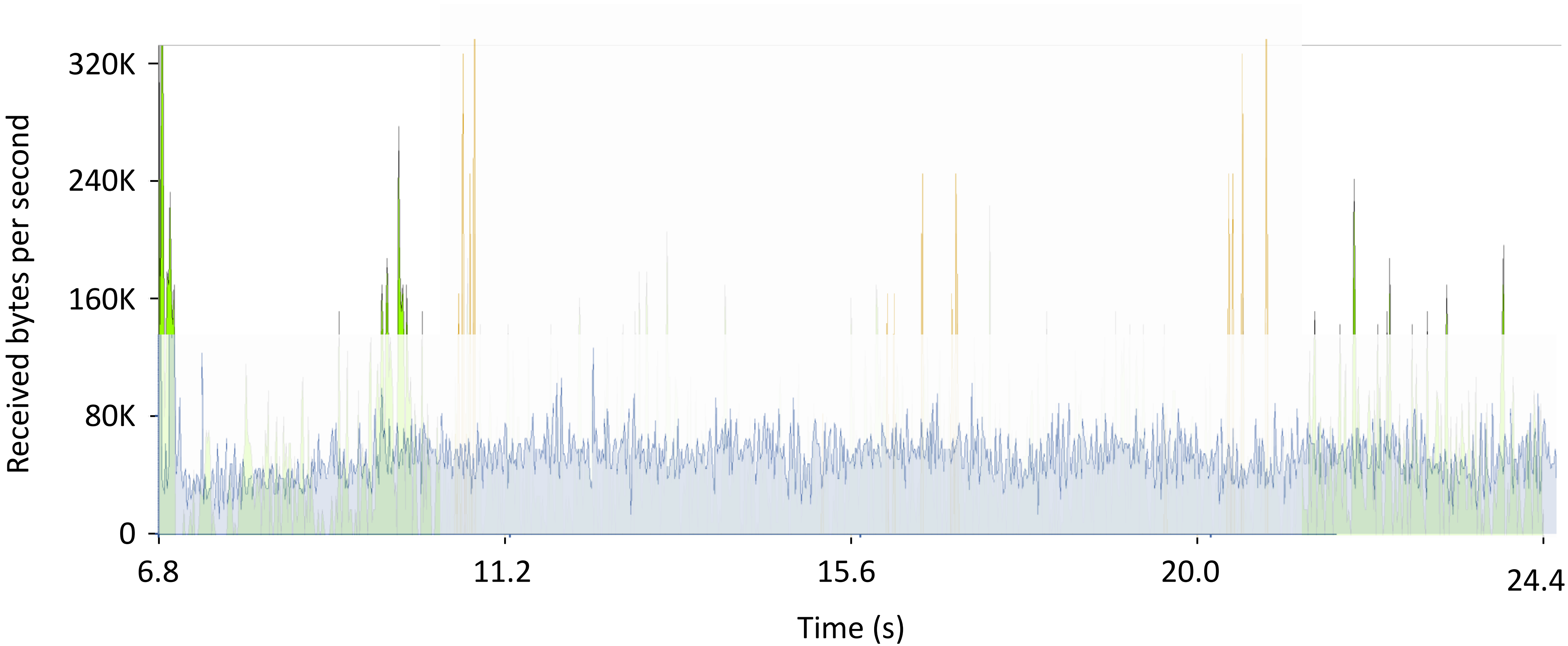


Charm++ Baseline



Spreading Technique




Received bytes per second
320K
240K
160K
80K
0
6.8
11.2
15.6
20.0
24.4
Time (s)

Runtime Integration

Automating teams configuration

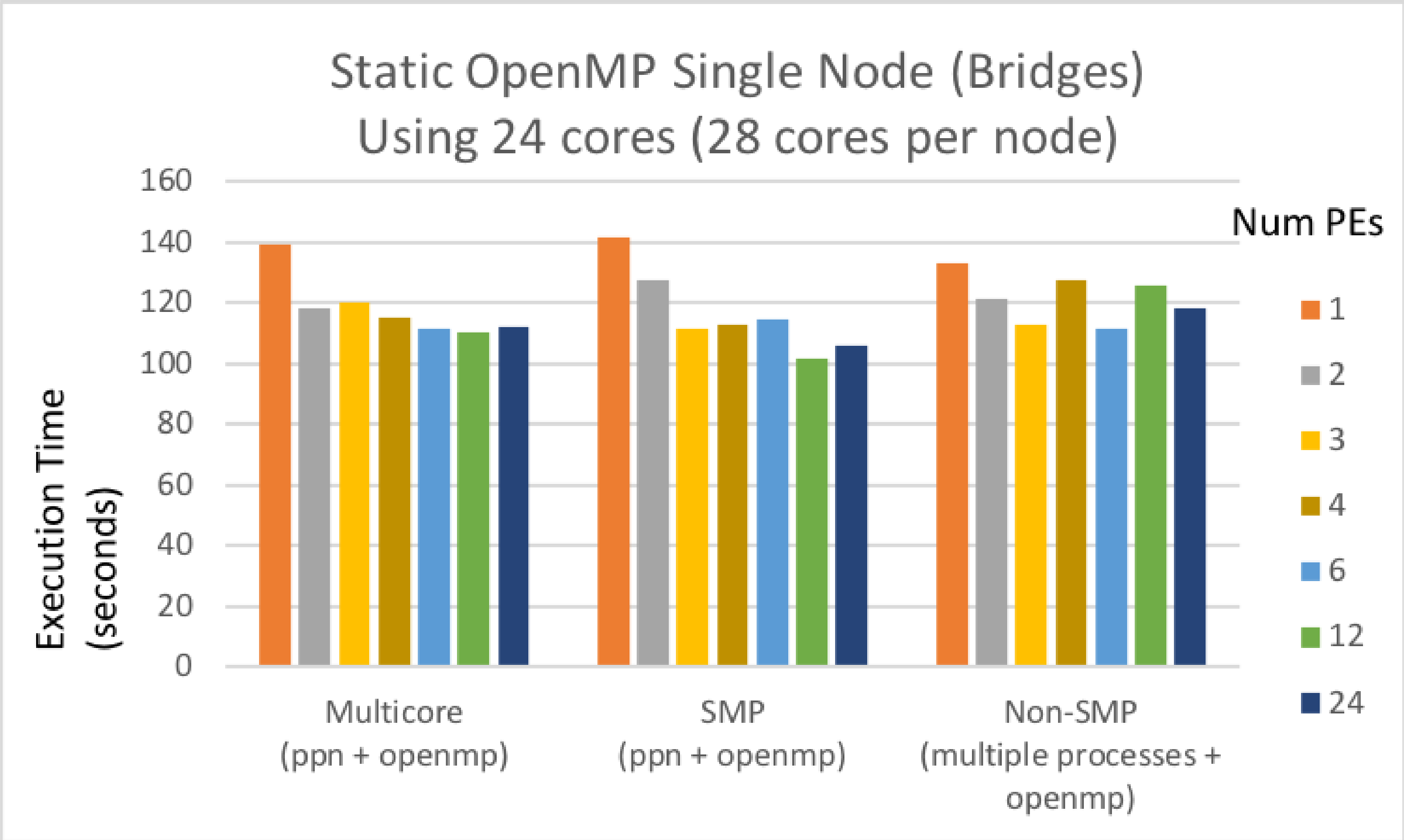
- Broader Agenda
  - Automate decisions -> easier for user
  - “Spread”: How many teams, i.e how many masters and how many drones?
  - Other runtime decisions:
  - How many ppn, i.e cores per process?
  - How many processes per node?
  - How many cores to turn off (memory bottleneck)?
  - Enable SMT or not?

Automating teams configuration

- Use OpenMP to create master thread on all cores
- Integrate with load balancing framework to change master thread count
- Use OpenMP nested parallelism to set/change number of drone threads within the application
  - Use pthread affinities instead of OpenMP affinity to update configurations at runtime
- Runtime selects the best performing configuration after testing with different configurations (one per LB step)

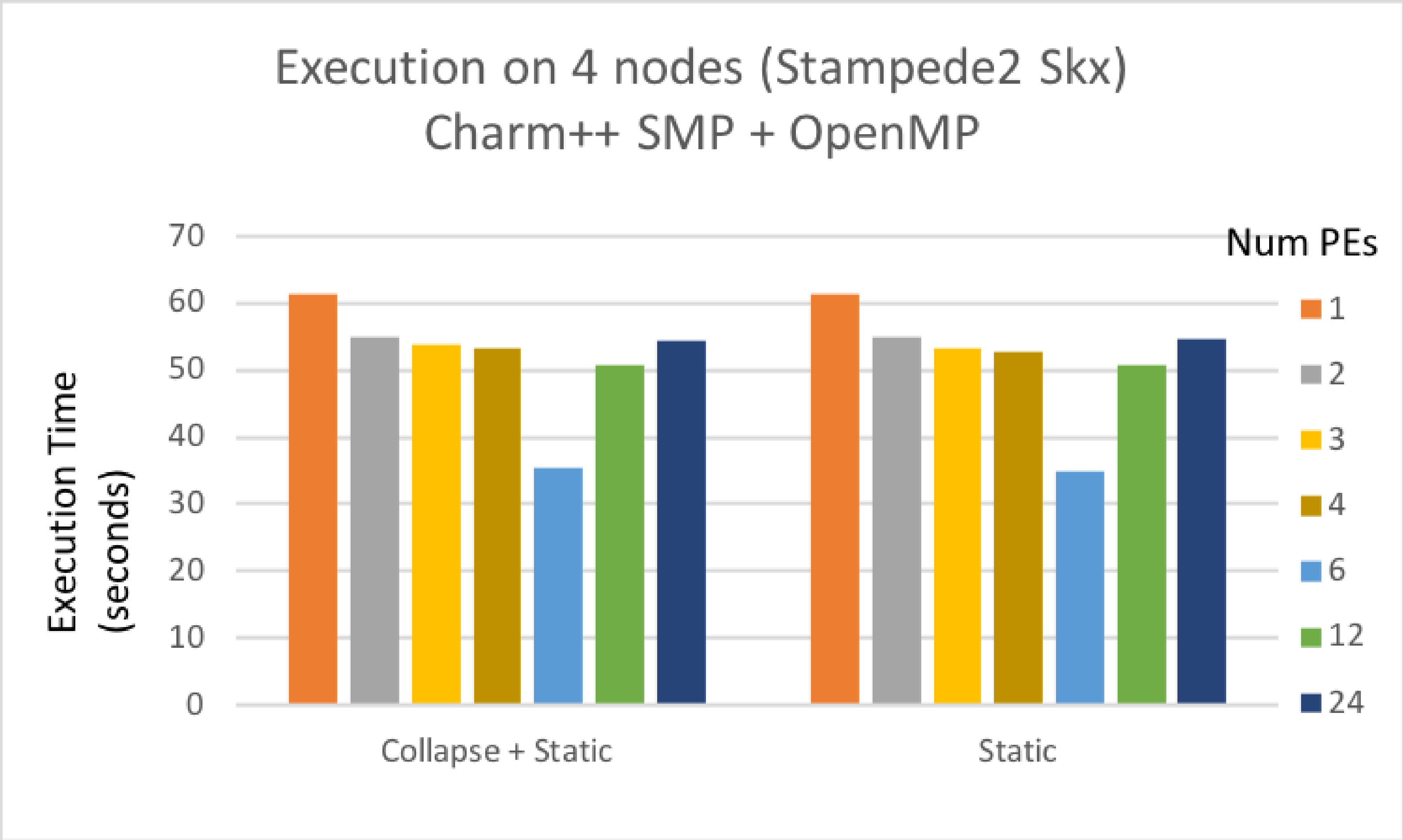
Using OpenMP with nested parallelism (static)

Bridges - single-node integrated OpenMP runs for SMP and Non-SMP builds



Using OpenMP with nested parallelism (static)

Stampede2 - Skylake 4-node run integrated OpenMP



OpenMP Implementation

machine-smp.C

```
int num_threads = tocreate + 1;
omp_set_dynamic(0);
omp_set_num_threads(num_threads);
#pragma omp parallel
{
  size_t i = omp_get_thread_num();
  call_startfn((void *)i);
}
```

jacobi2d.C

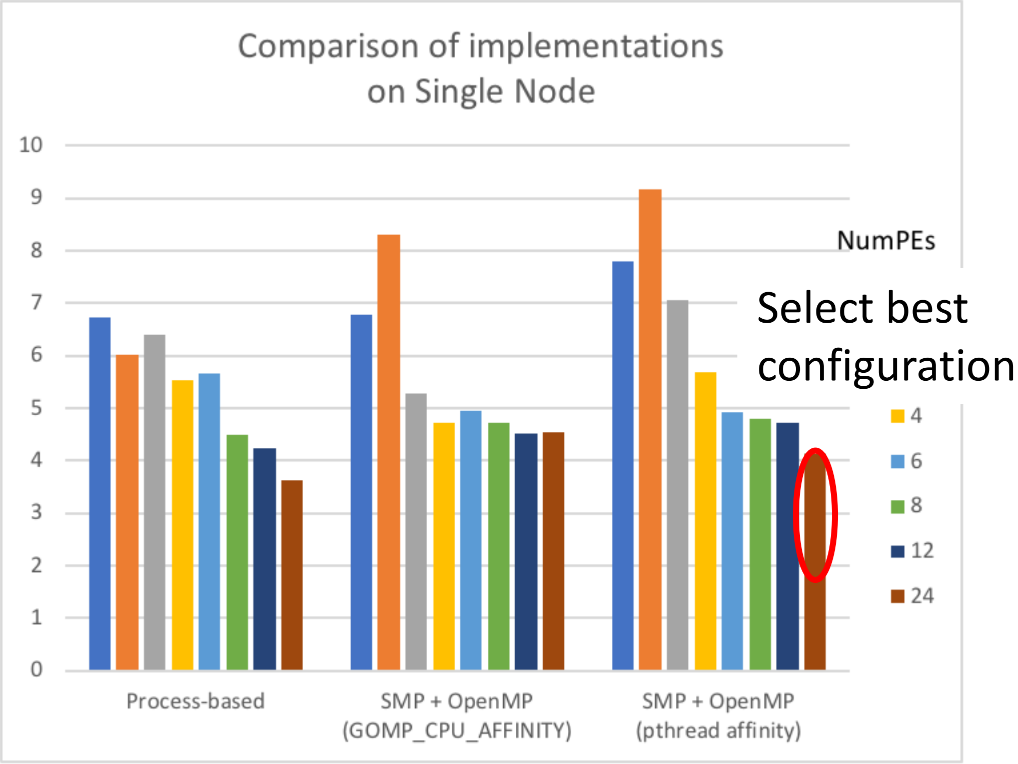
```
#pragma omp parallel for private(temperatureIth, \
                            difference) num_threads(drones_per_pe)
for(int i=istart; i<ifinish; ++i) {
  for(int j=jstart; j<jfinish; ++j) {
    temperatureIth=(temperature[i][j]
      + temperature[i-1][j]
      +  temperature[i+1][j]
      +  temperature[i][j-1]
      +  temperature[i][j+1]) * 0.2;
```

Static configuration:
OMP_NESTED=true
GOMP_CPU_AFFINITY=0-7 (eg. for 8 cores)
OMP_PROC_BIND=spread

Dynamic configuration:
pthread_setaffinity_np(thread, sizeof(cpu_set_t), &cpuset);

OpenMP implementation with pthread affinity

- Similar performance with process-based and OpenMP implementations
  - Some NUMA effects
- OpenMP Limitations:
  - Nested parallelism configurations cannot be dynamically changed
  - Affinities are set at the initialization and cannot be changed
- With Charm++ we are able to dynamically change OpenMP configurations and with pthread affinity we set affinities for each new configuration



Next steps

- Integrate the LB framework to fully automate configuration selection
  - Current implementation is able to dynamically set different configurations at runtime based on user input
  - Benefit over static OpenMP configuration – configurations and affinities can be changed at runtime
- Compare with CkLoop implementation in Charm++

Summary

- Spreading offers new optimization parameter
- Increases performance 20-30% in prototype application
- Spread factor is controllable at runtime
- Current integration into Charm++ ongoing

Questions

Michael Robson michael.robson@villanova.edu

Kavitha Chandrasekar kchndrs2@illinois.edu