Is your scheduling good?
How would you know?

14th Scheduling for Large Scale Systems Workshop

Anthony Danalis, Heike Jagode, Jack Dongarra

Bordeaux, France
June 26-28, 2019

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Scheduling question

Q: What is the optimal task scheduling algorithm?

Scheduling question

Q: What is the optimal task scheduling algorithm?

A: There is no such thing. Optimality is case specific.

Scheduling question(s)

Q: What is the optimal task scheduling algorithm?

A: There is no such thing. Optimality is case specific.

Q2: How should the scheduler of a runtime work?

Scheduling question(s)

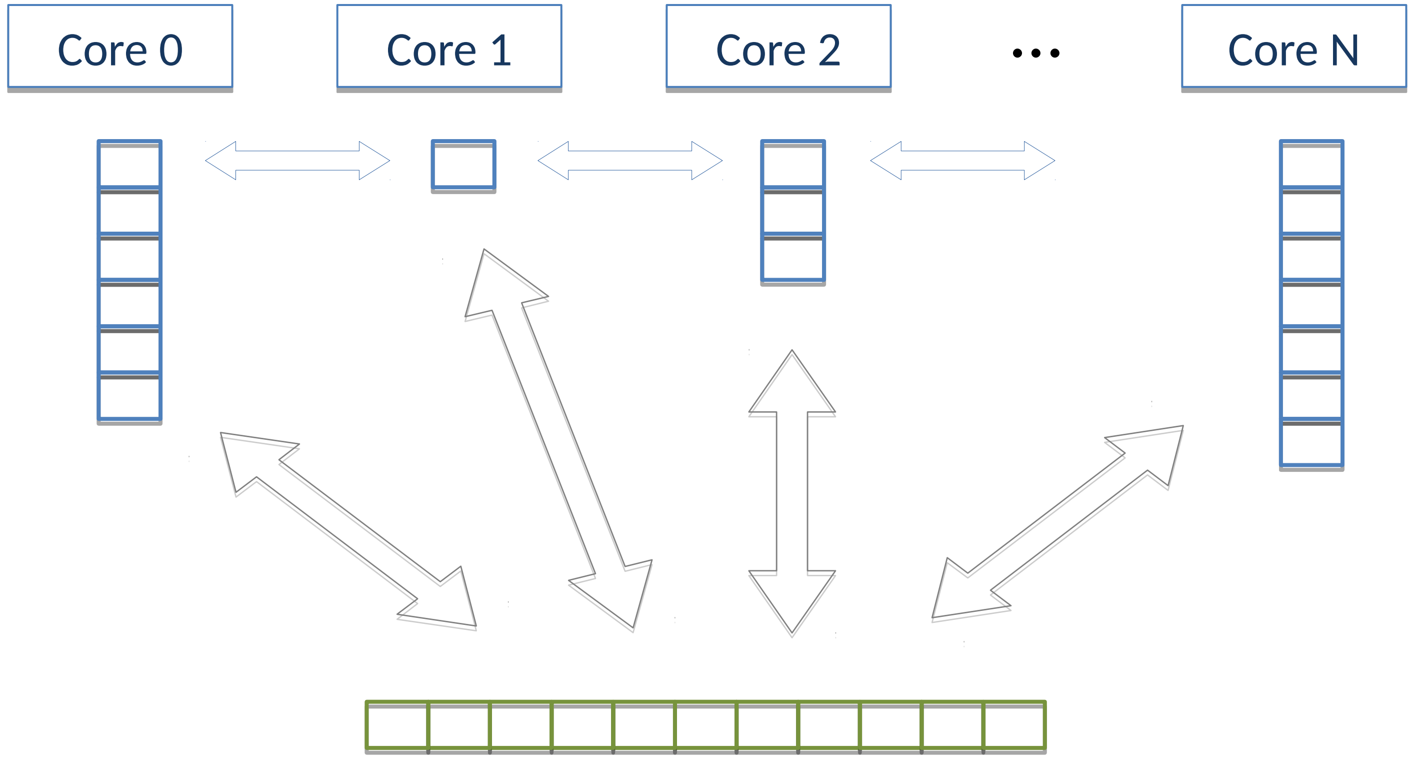
Q: What is the optimal task scheduling algorithm?

A: There is no such thing. Optimality is case specific.

Q2: How should the scheduler of a runtime work?

Q2a: How should I choose a scheduler for my problem?

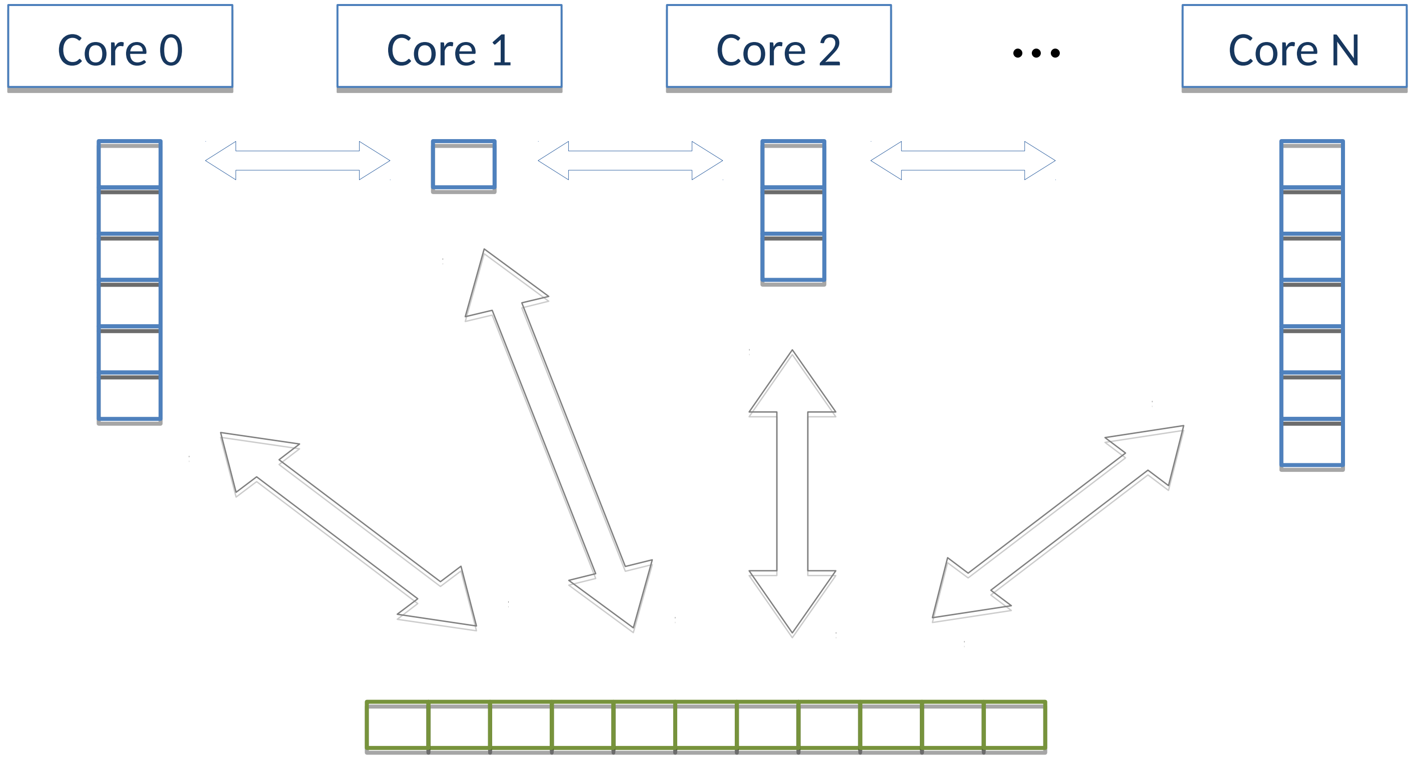
Case study: PaRSEC's LFQ



Core local queues

Shared Global queue (overflow)

Case study: PaRSEC's LFQ



Core local queues

Shared Global queue (overflow)

Thread Local Queues => High Locality
Overflow & Work Stealing => Load Balance

(More) Scheduling questions

Q3: How long should the local queues be?

Q4: Should a thread first steal from a close queue, any queue, or the shared queue?

(More) Scheduling questions

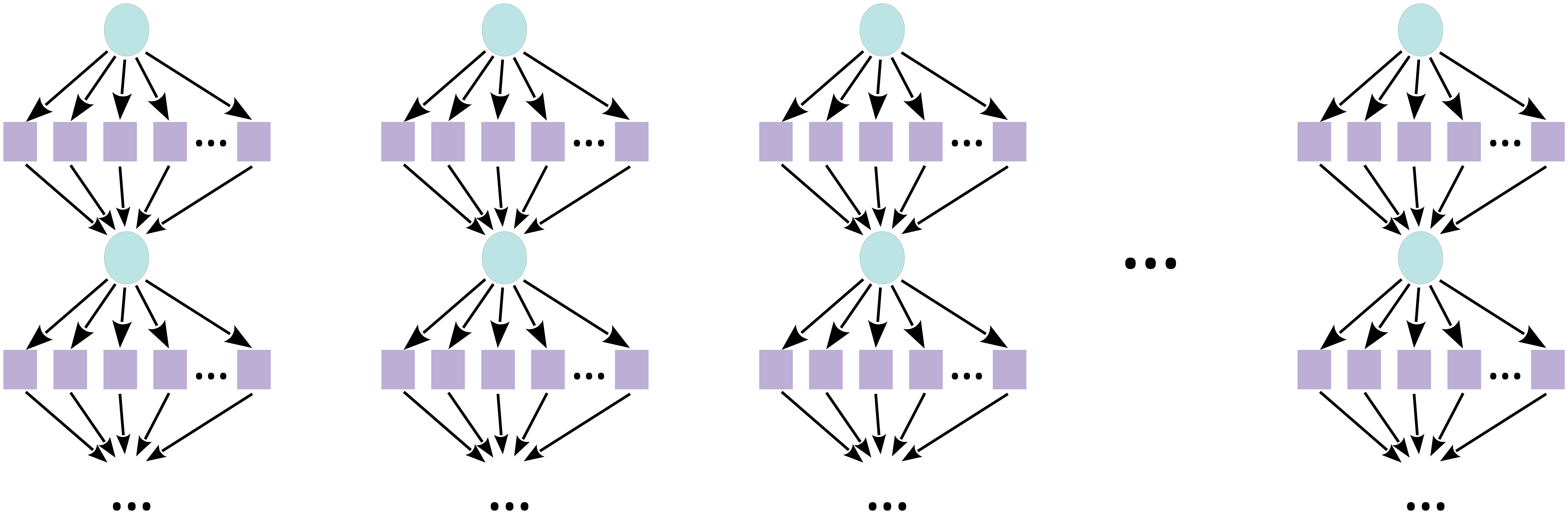
Q3: How long should the local queues be?

A: 4*Core_Count

Q4: Should a thread first steal from a close queue, any queue, or the shared queue?

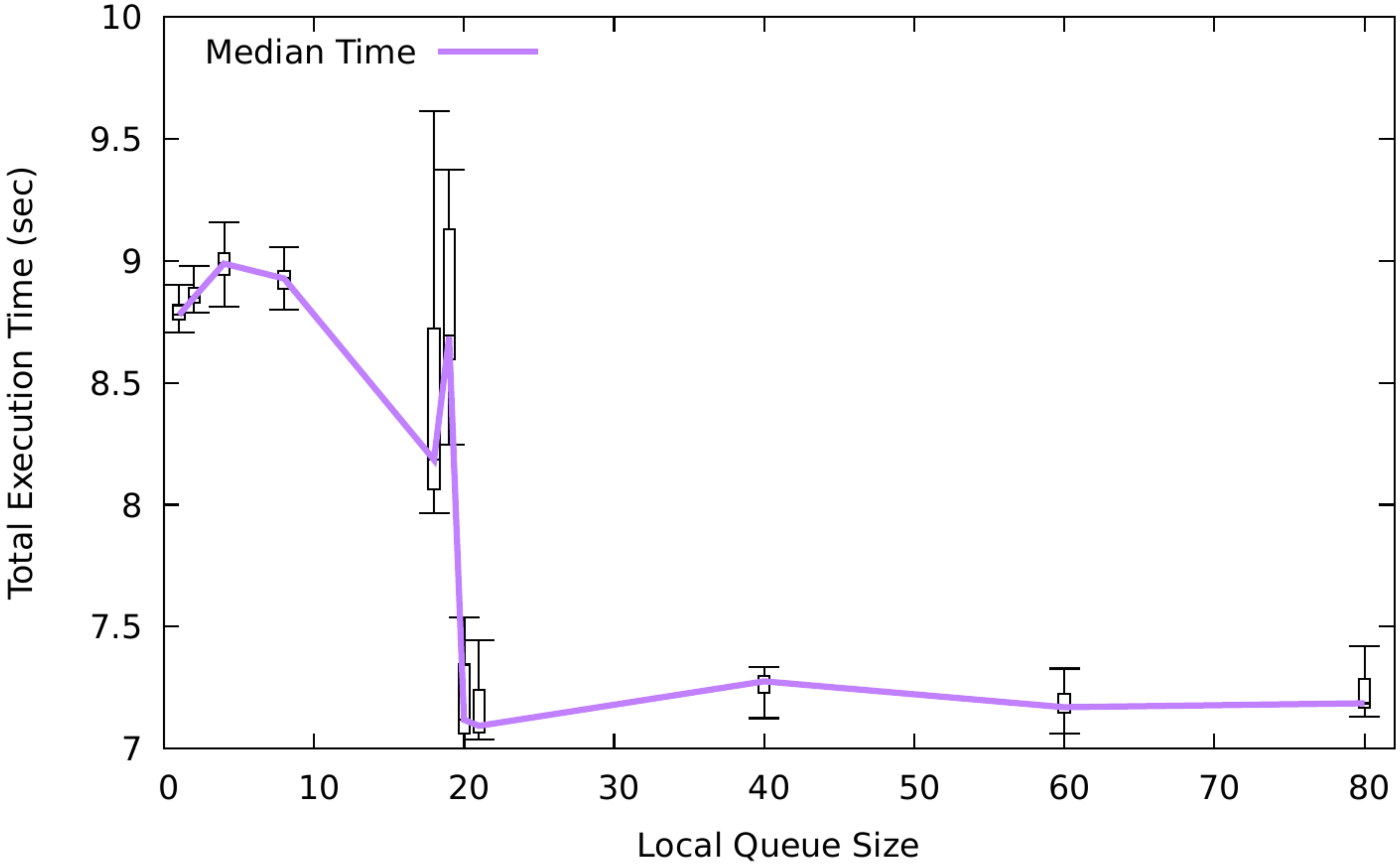
A: Any local queue (closest to farthest), then shared queue.

Testing Benchmark

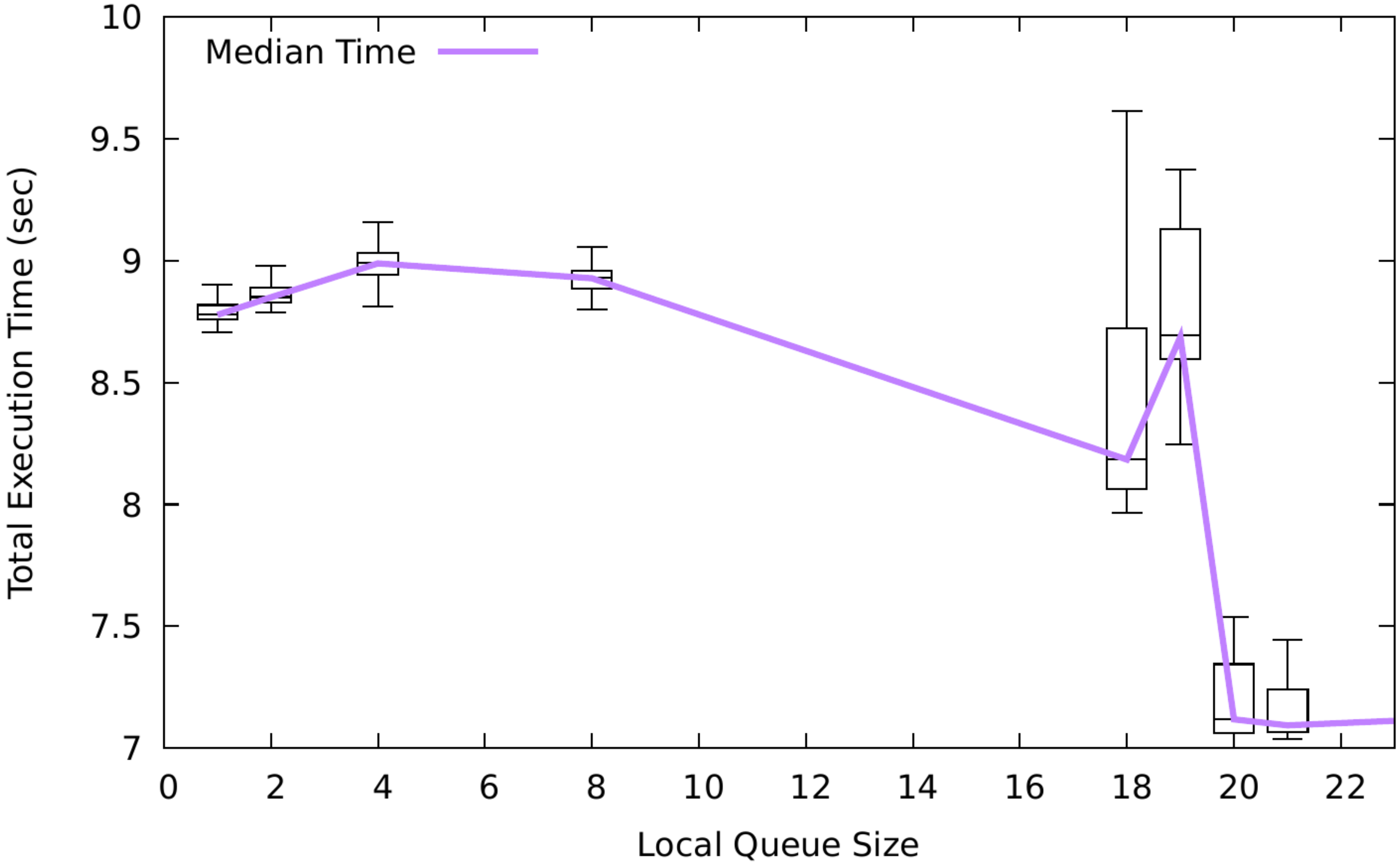


- 20 Independent Fork-Join chains x 20 Tasks per fork.
- Memory bound kernel, with good cache locality.
- 20 Cores on testing node.

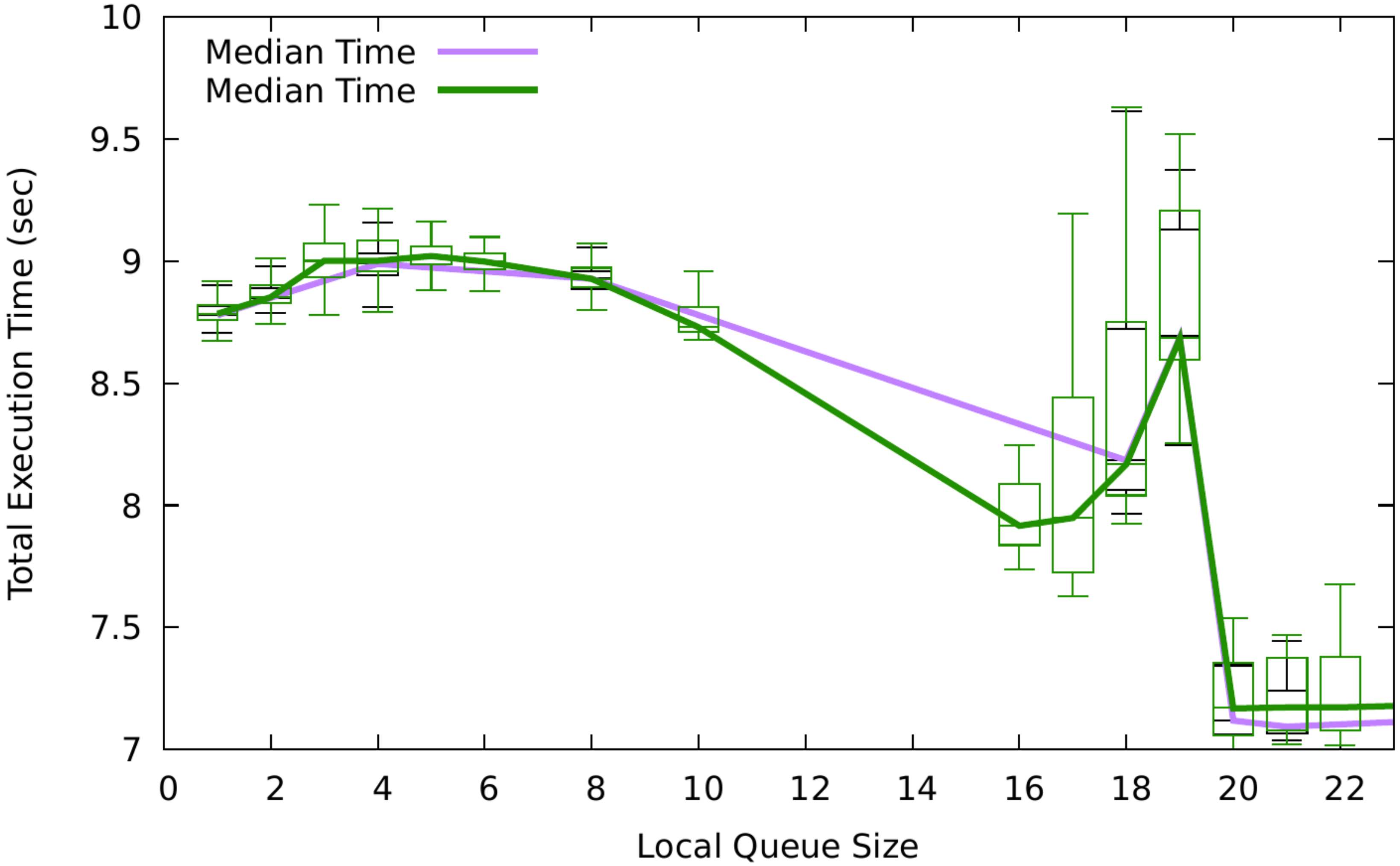
Execution time vs Local Queue Length



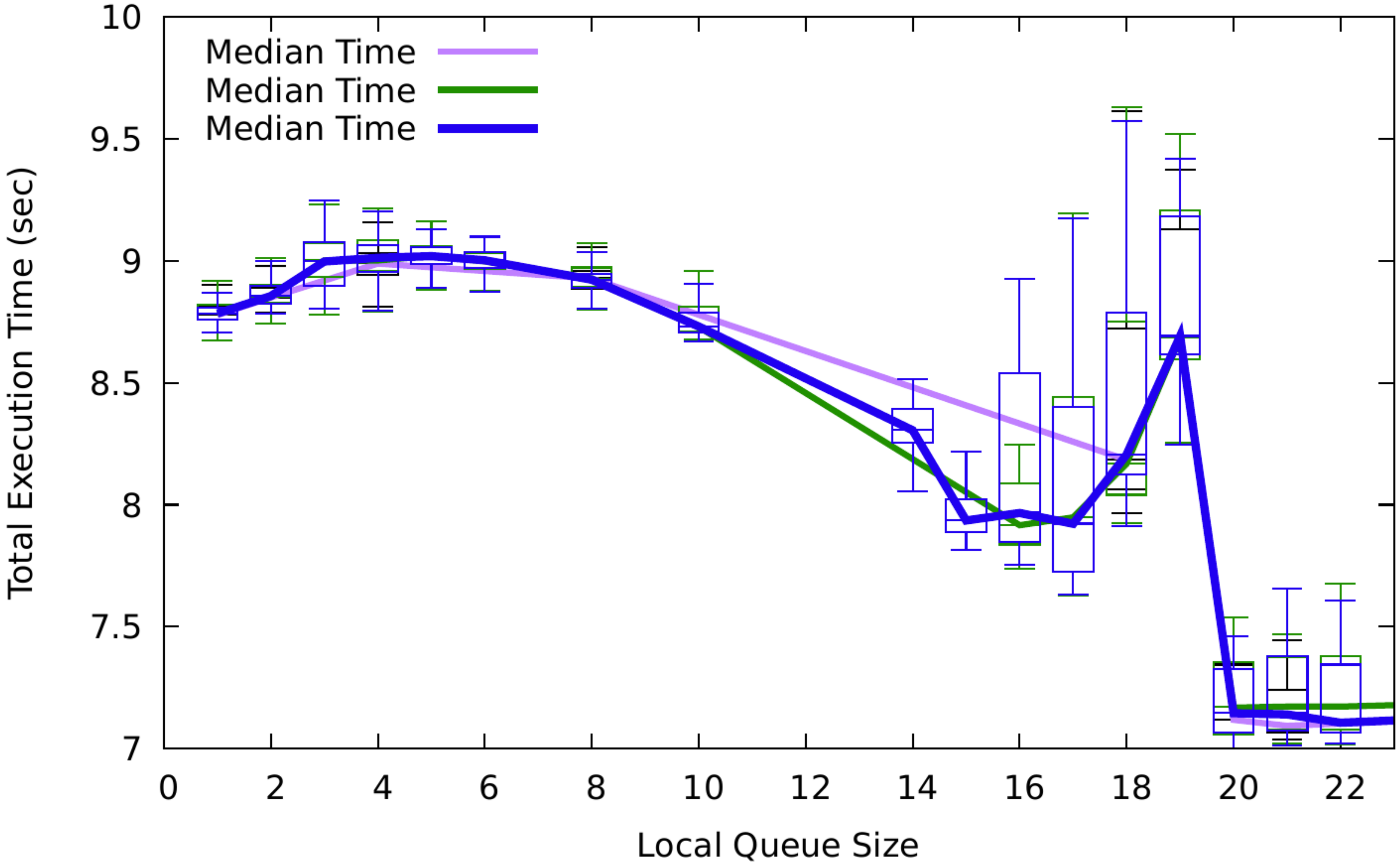
Execution time vs Local Queue Length (zoom)



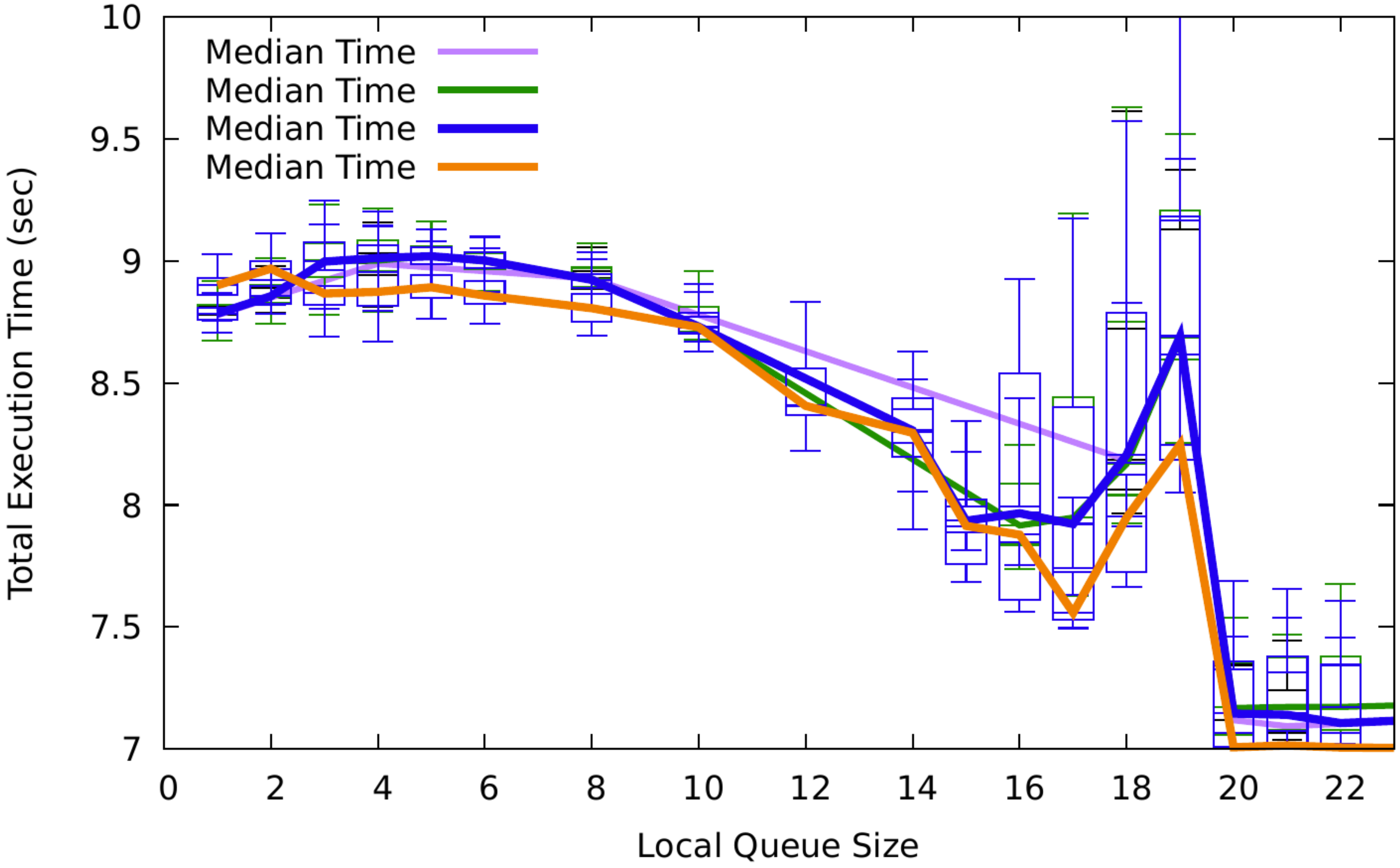
Execution time vs Local Queue Length (zoom 2)



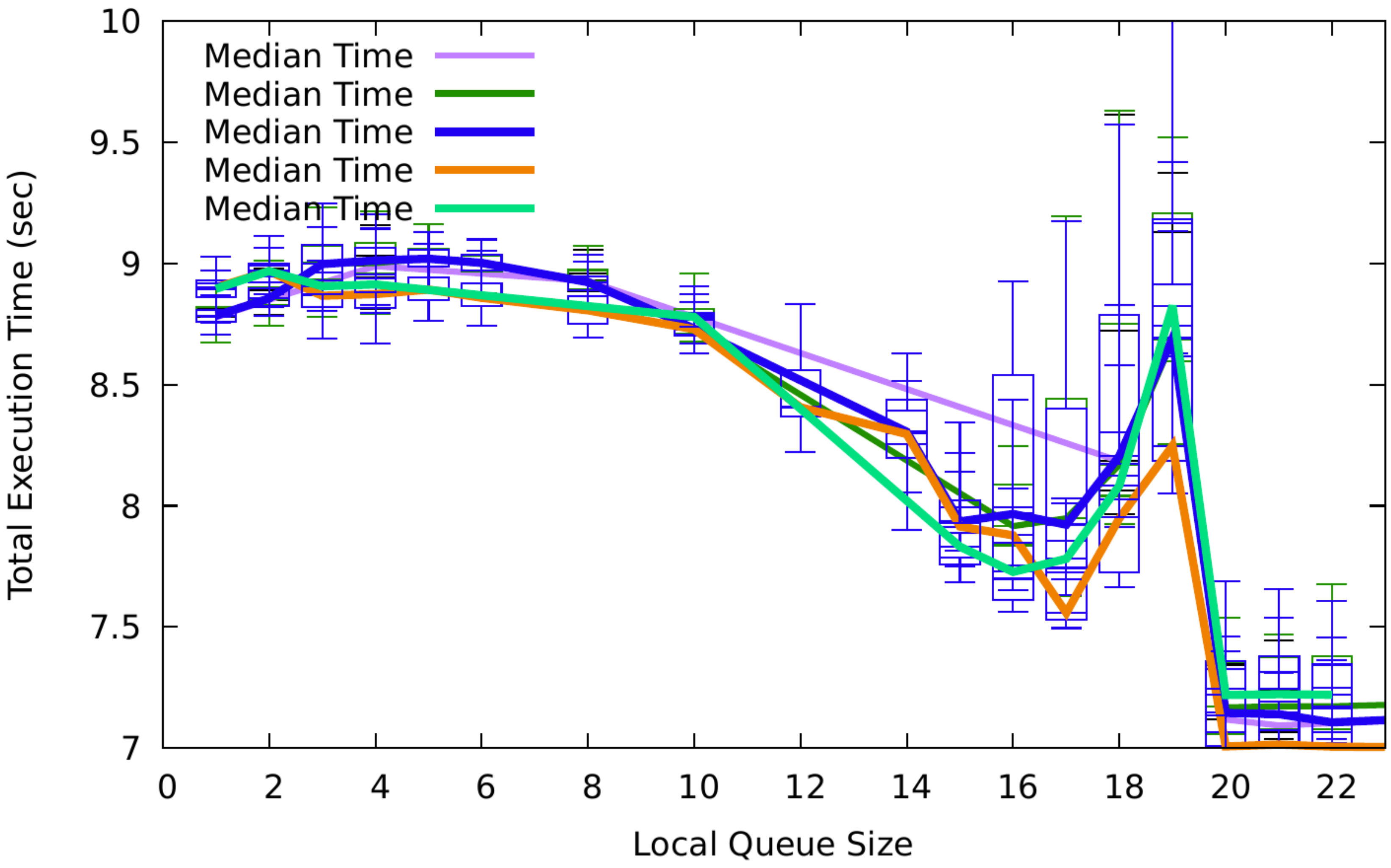
Execution time vs Local Queue Length (zoom 3)



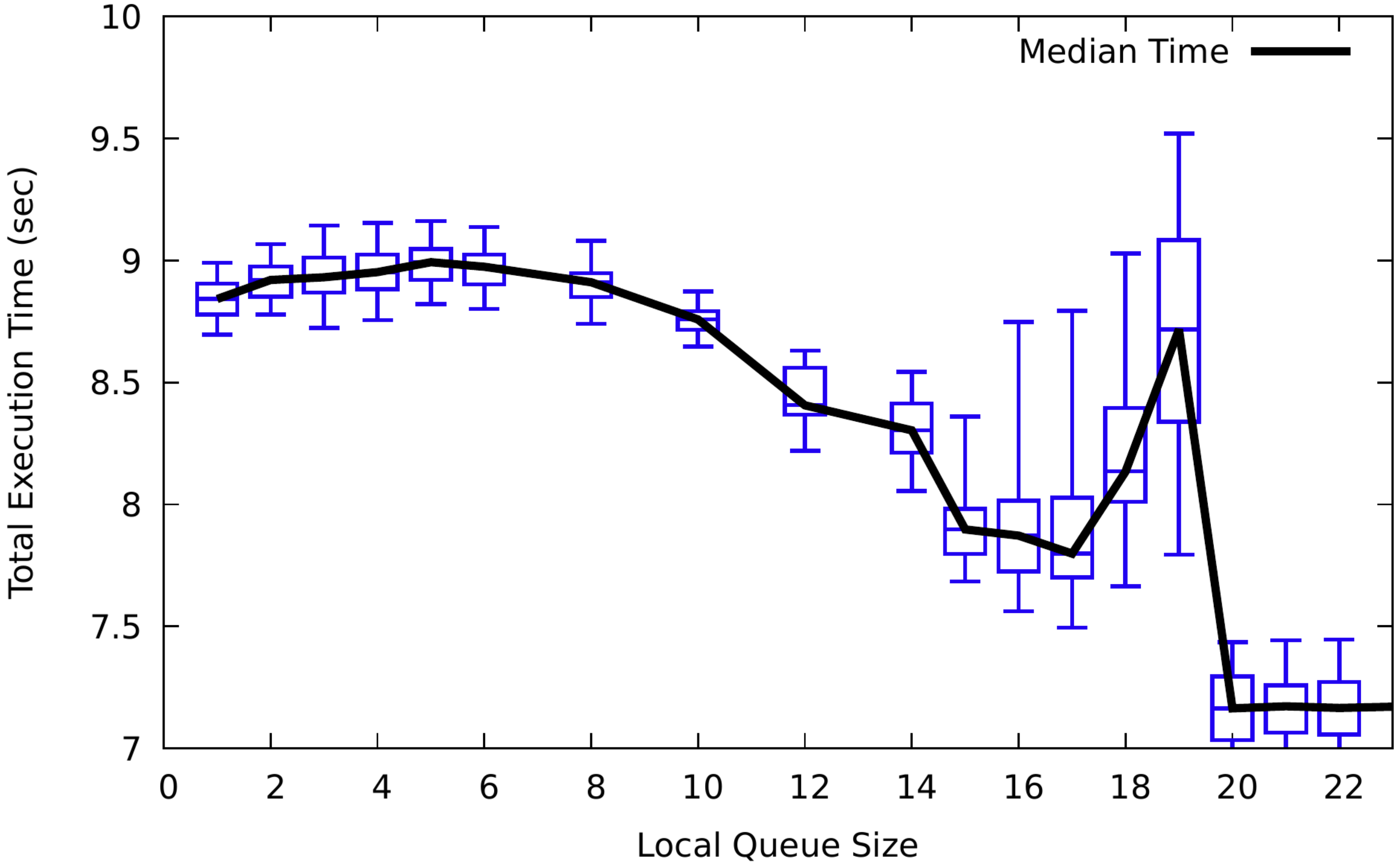
Execution time vs Local Queue Length (zoom 4)



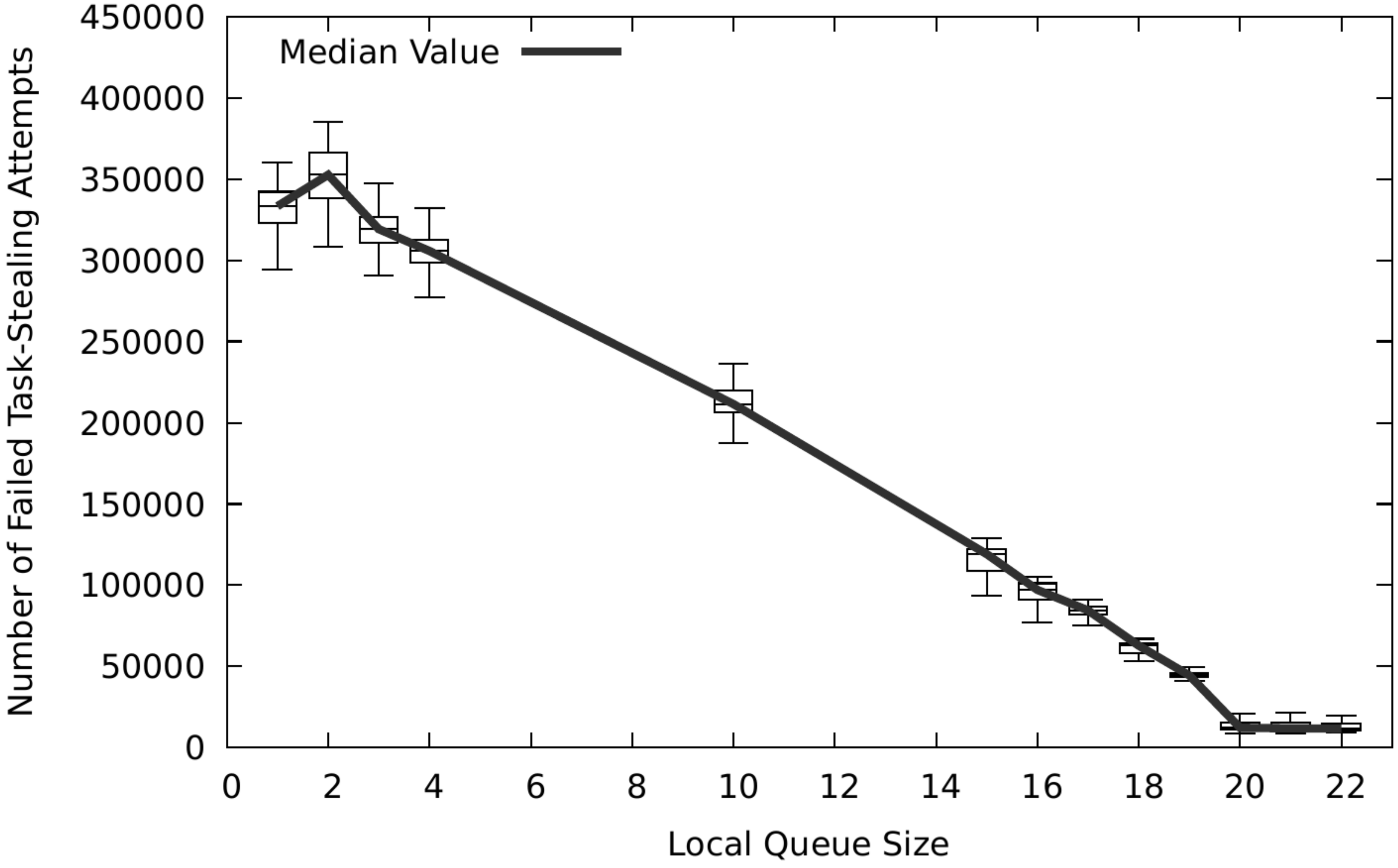
Execution time vs Local Queue Length (zoom 5)



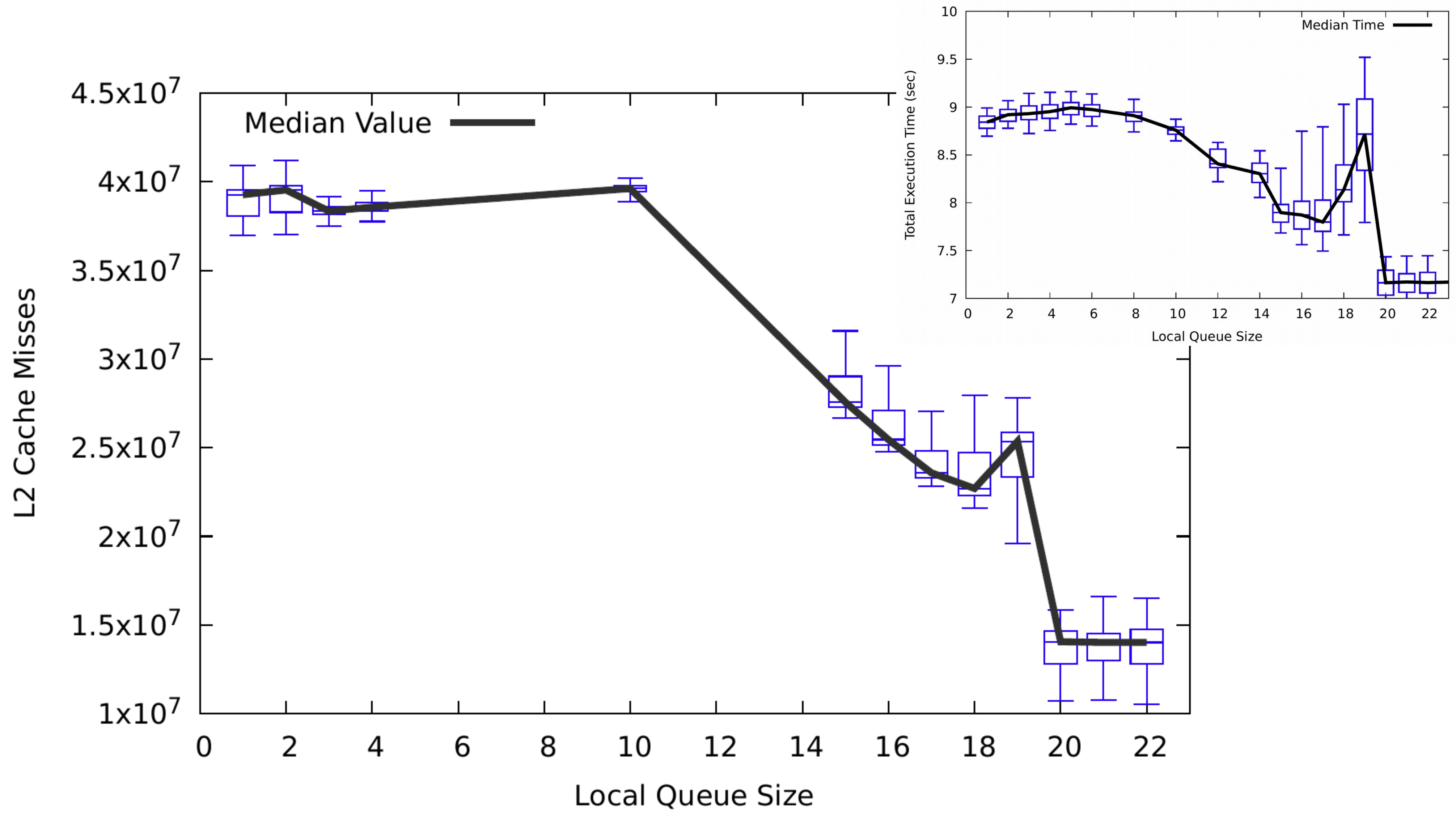
Execution time vs Local Queue Length (combined)



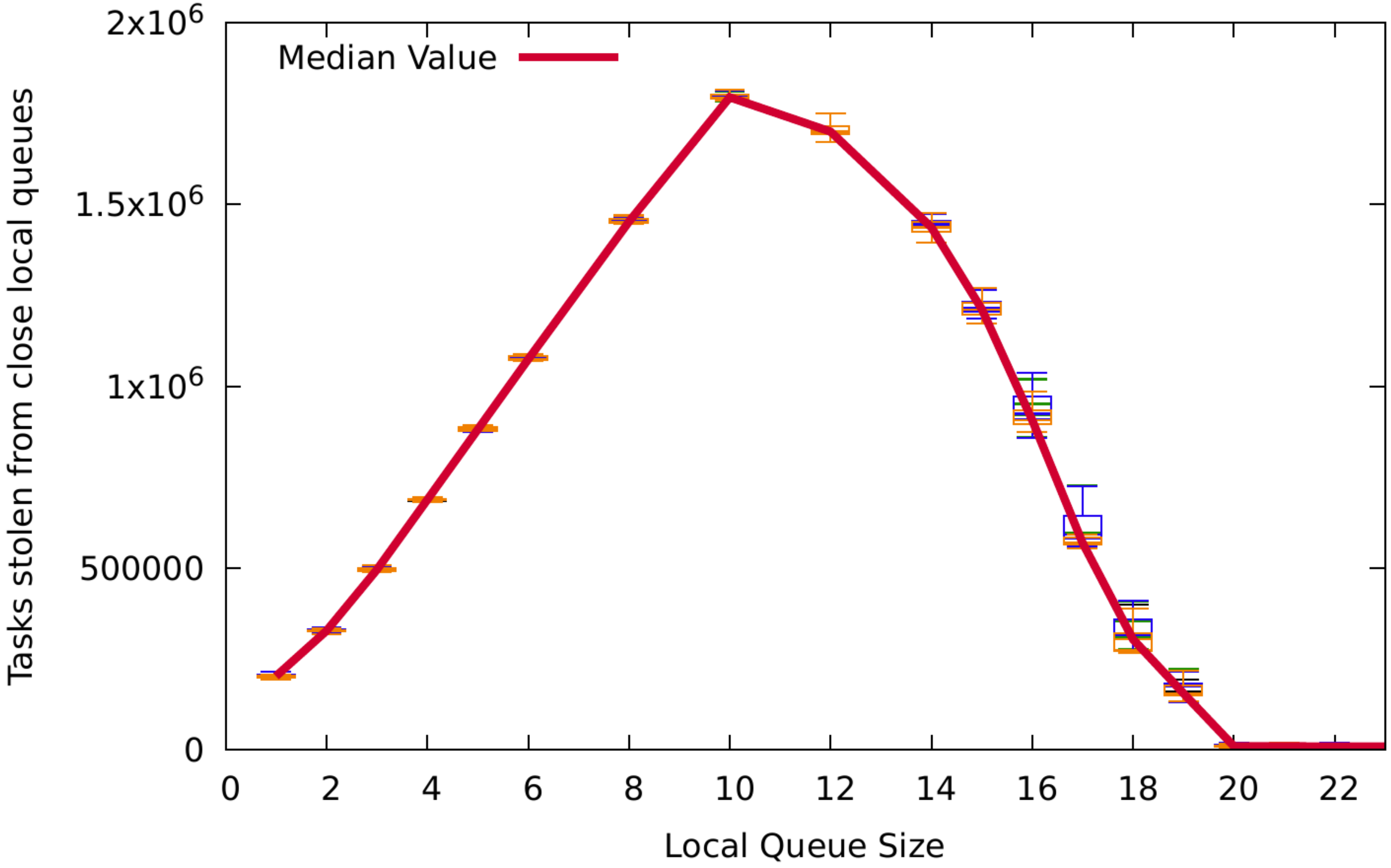
Failed Stealing Attempts



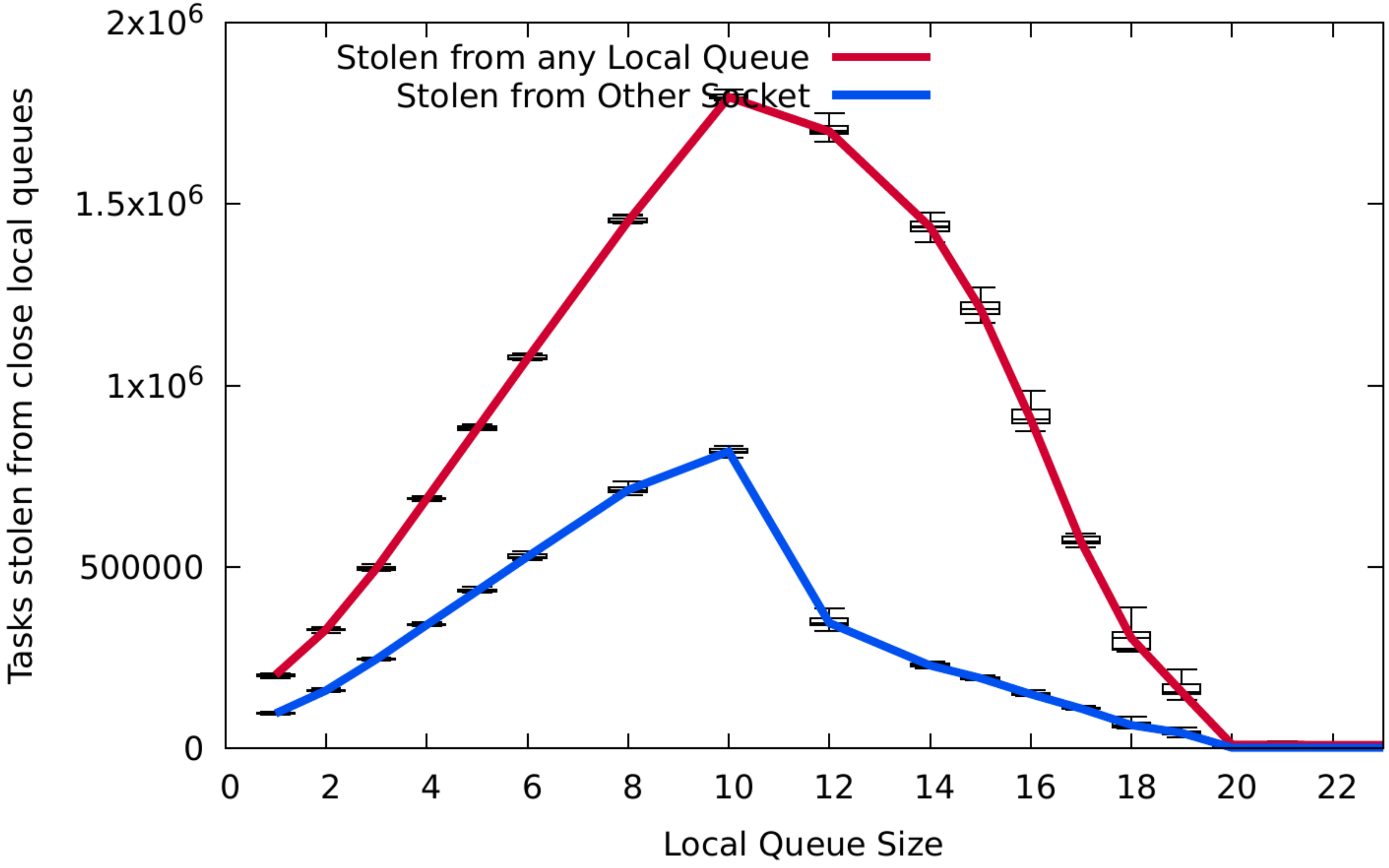
L2 Cache Misses (L3 show same pattern)



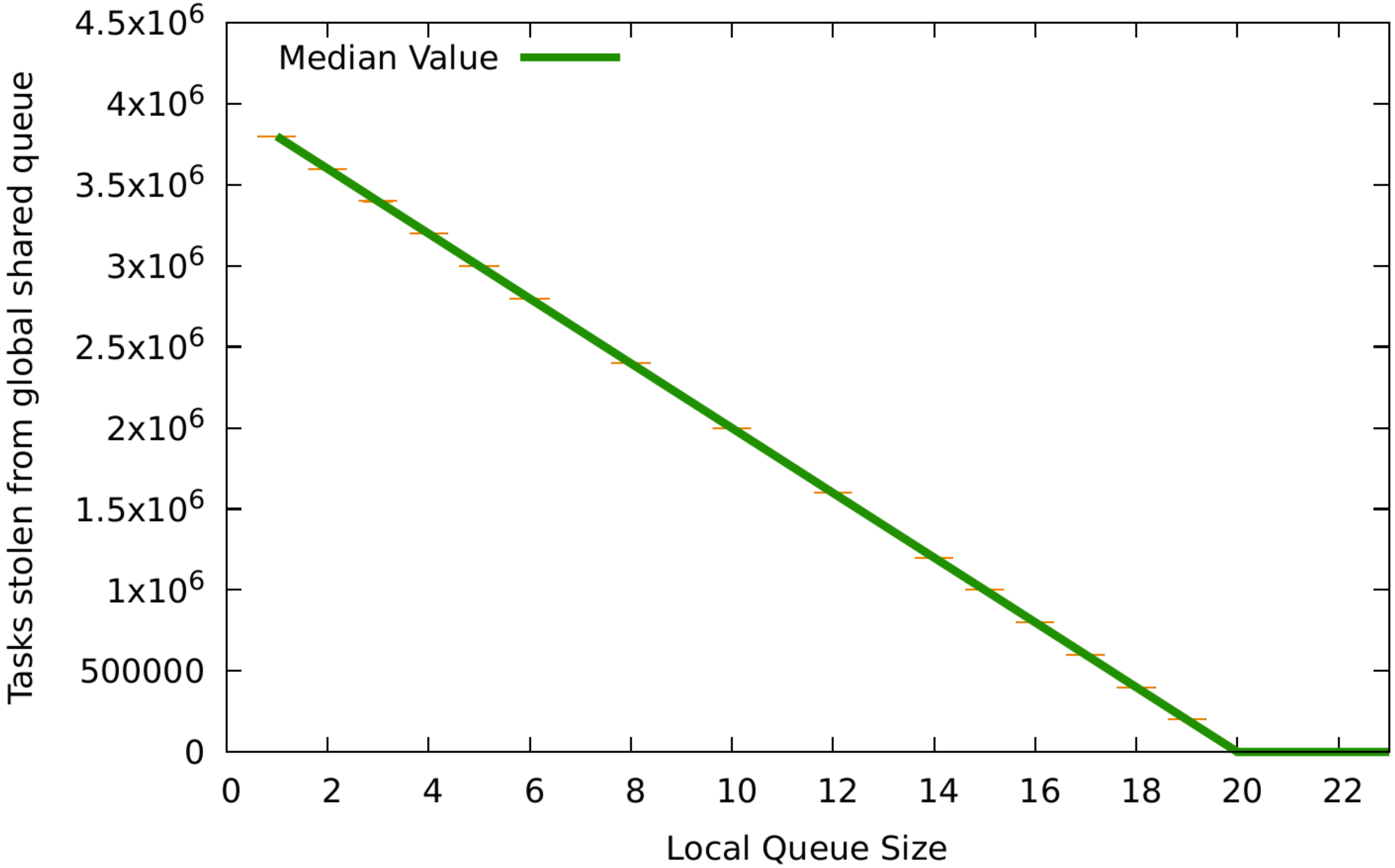
Successful Close Stealing



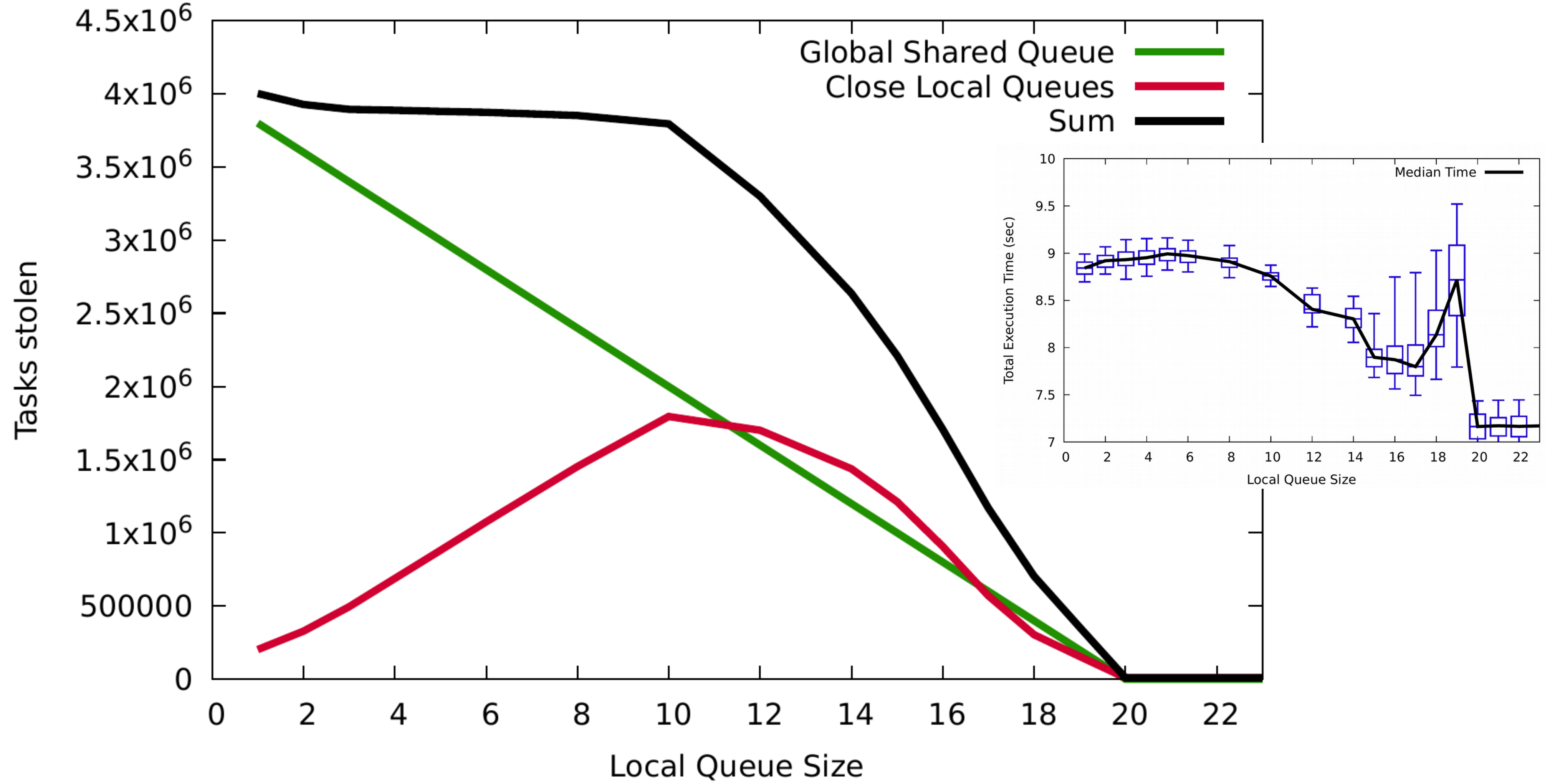
Successful Close & Far Stealing



Successful Shared Queue Stealing



Successful Local + Shared Queue Stealing



Your questions

Q: So, what causes the bump?

Q: How did you measure all these things?

Your questions

Q: So, what causes the bump?

A: I don’t know!

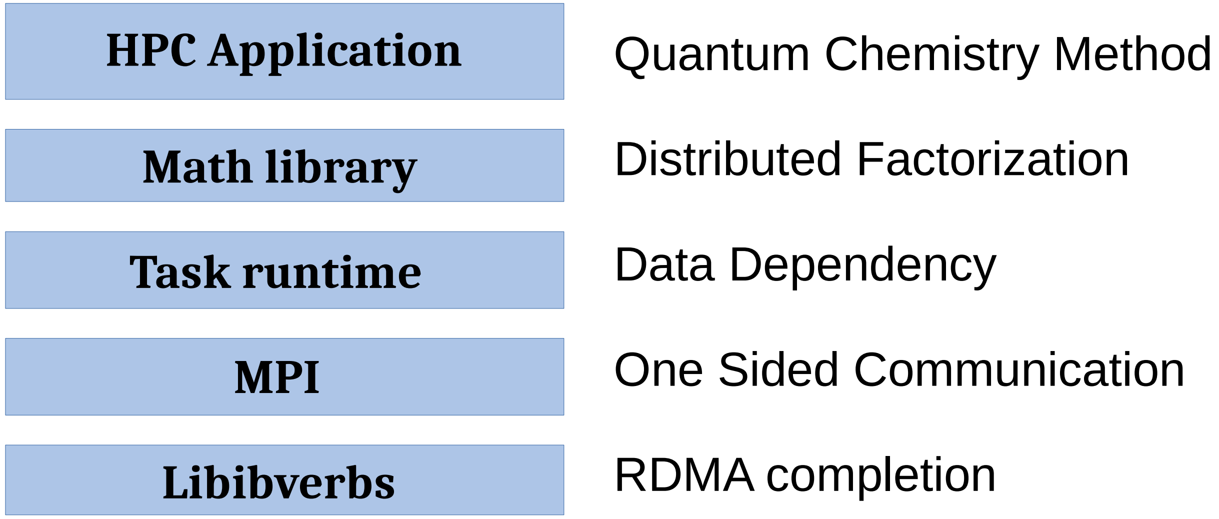
Q: How did you measure all these things?

A: I am glad you asked.

What is missing from current infrastructure?

Events that occurred inside the software stack

There is no standardized way for a software layer to export information about its behavior such that other, independently developed, software layers can read it.



HPC Application	Quantum Chemistry Method
Math library	Distributed Factorization
Task runtime	Data Dependency
MPI	One Sided Communication
Libibverbs	RDMA completion

PAPI Software Defined Events

- De facto standard:

  SDEs from your library can be read using the standard PAPI_start()/PAPI_stop().

- Low overhead:

  Performance critical codes can implement SDEs with zero overhead by exporting existing code variables without adding any new instructions in the fast path.

- Rich feature set:

  PAPI SDE supports counters, groups, recordings, simple statistics, thread safety, custom callbacks.

Simplest SDE code

```
static long long local_var;

void small_test_init( void ){
    local_var = 0;
    papi_handle_t *handle = papi_sde_init(″TEST");
    papi_sde_register_counter( handle, ″Evnt",
                               PAPI_SDE_RO|PAPI_SDE_DELTA,
                               PAPI_SDE_long_long,
                               &local_var );
   ...
}
```

SDE code for registering a callback function

```
sometype_t *data;

void small_test_init( void ){
    data = ...
    papi_handle_t *handle = papi_sde_init(”TEST");
    papi_sde_register_fp_counter(handle, "Evnt",
                                 PAPI_SDE_RO|PAPI_SDE_DELTA,
                                 PAPI_SDE_long_long,
                                 accessor, data);
   ...
}
```

SDE code for creating a counter (push mode)

```
void *counter_handle;

void small_test_init( void ){
    papi_handle_t *handle = papi_sde_init(”TEST");
    papi_sde_create_counter(handle, "Evnt",
                            PAPI_SDE_long_long,
                            &counter_handle);
   ...
}
```

SDE code for creating a recorder (push mode)

```
void *recorder_handle;

void small_test_init( void ){
    papi_handle_t *handle = papi_sde_init(”TEST");
    papi_sde_create_recorder(handle, "RCRDR",
                             sizeof(double),
                             cmpr_func_ptr,
                             &recorder_handle);
   ...
}
```

SDE code for creating a recorder (push mode)

```
void *recorder_handle;
```

```
sde:::TEST::RCRDR
```

```
void small_test_init( void ){
    papi_handle_t *handle = papi_sde_init(″TEST");
    papi_sde_create_recorder(handle, "RCRDR",
                             sizeof(double),
                             cmpr_func_ptr,
                             &recorder_handle);

   ...
}
```

SDE code for creating a recorder (push mode)

```
void *recorder_handle;

void small_test_init(void){
    papi_handle_t *handle = papi_sde_init(”TEST");
    papi_sde_create_recorder(handle, "RCRDR",
                             sizeof(double),
                             cmpr_func_ptr,
                             &recorder_handle);

   ...
}
```

```
sde:::TEST::RCRDR
sde:::TEST::RCRDR:CNT
```

SDE code for creating a recorder (push mode)

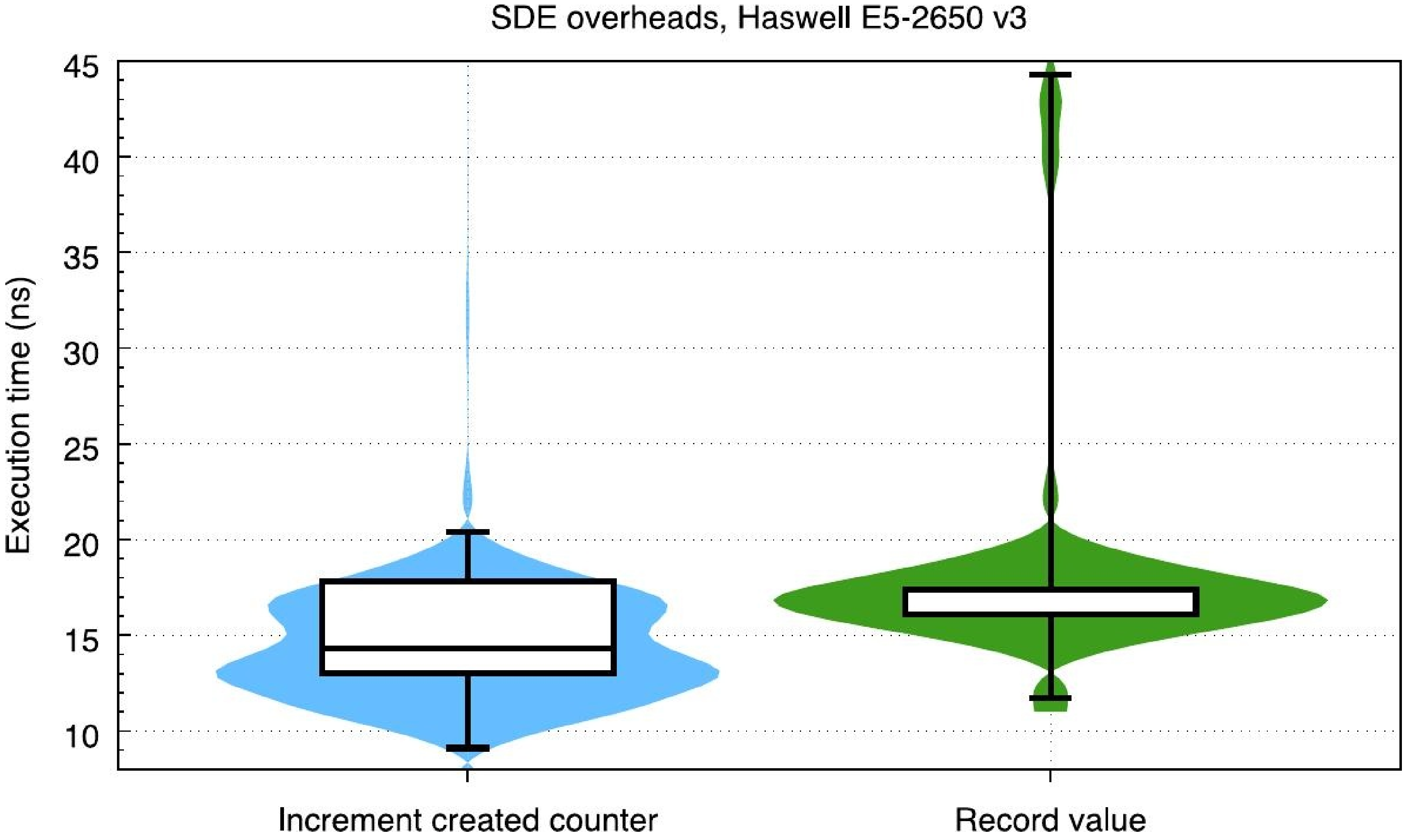
```
void *recorder_handle;

void small_test_init(void){
    papi_handle_t *handle = papi_sde_init(”TEST");
    papi_sde_create_recorder(handle, "RCRDR",
                             sizeof(double),
                             cmpr_func_ptr,
                             &recorder_handle);

   ...
}
```

```
sde:::TEST::RCRDR
sde:::TEST::RCRDR:CNT
sde:::TEST::RCRDR:MIN
sde:::TEST::RCRDR:Q1
sde:::TEST::RCRDR:MED
sde:::TEST::RCRDR:Q3
sde:::TEST::RCRDR:MAX
```

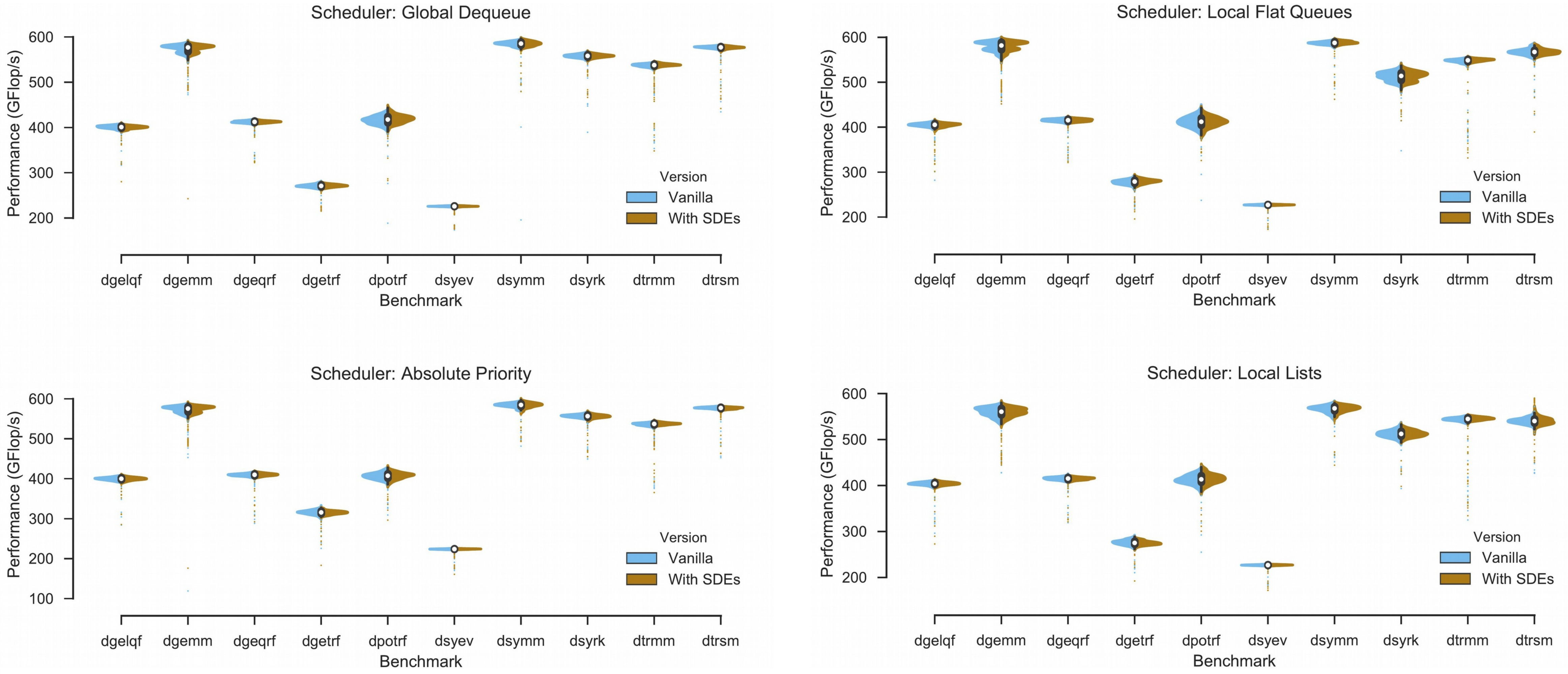
SDE code for updating created counters/recorders

```
void *counter_handle;
void *recorder_handle;

void push_test_dowork(void){
    double val;
    long long increment = 3;

    val = perform_useful_work();
    papi_sde_inc_counter(counter_handle, increment);
    papi_sde_record(recorder_handle, sizeof(val), &val);
}
```

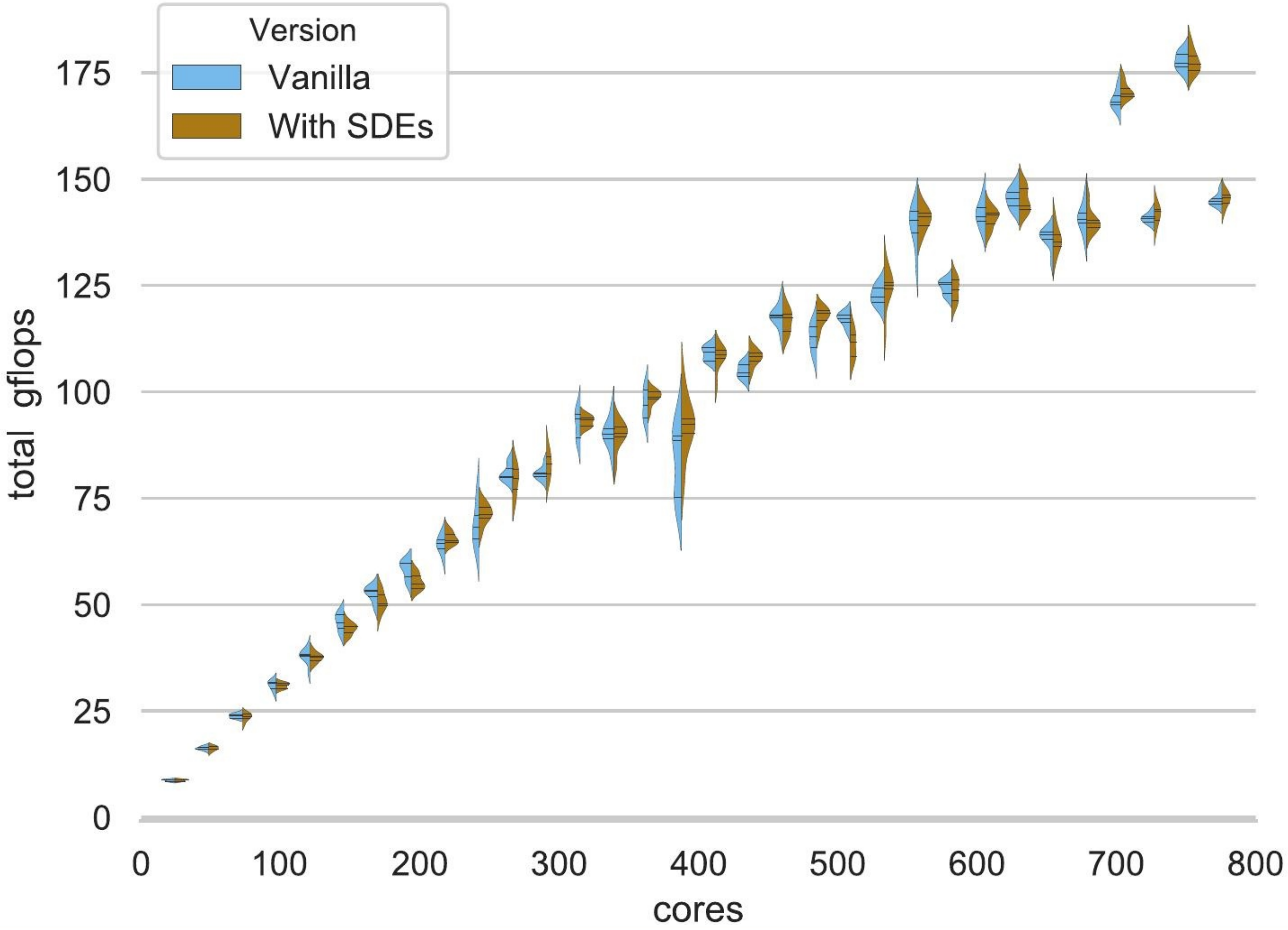
Performance overheads in simple benchmark



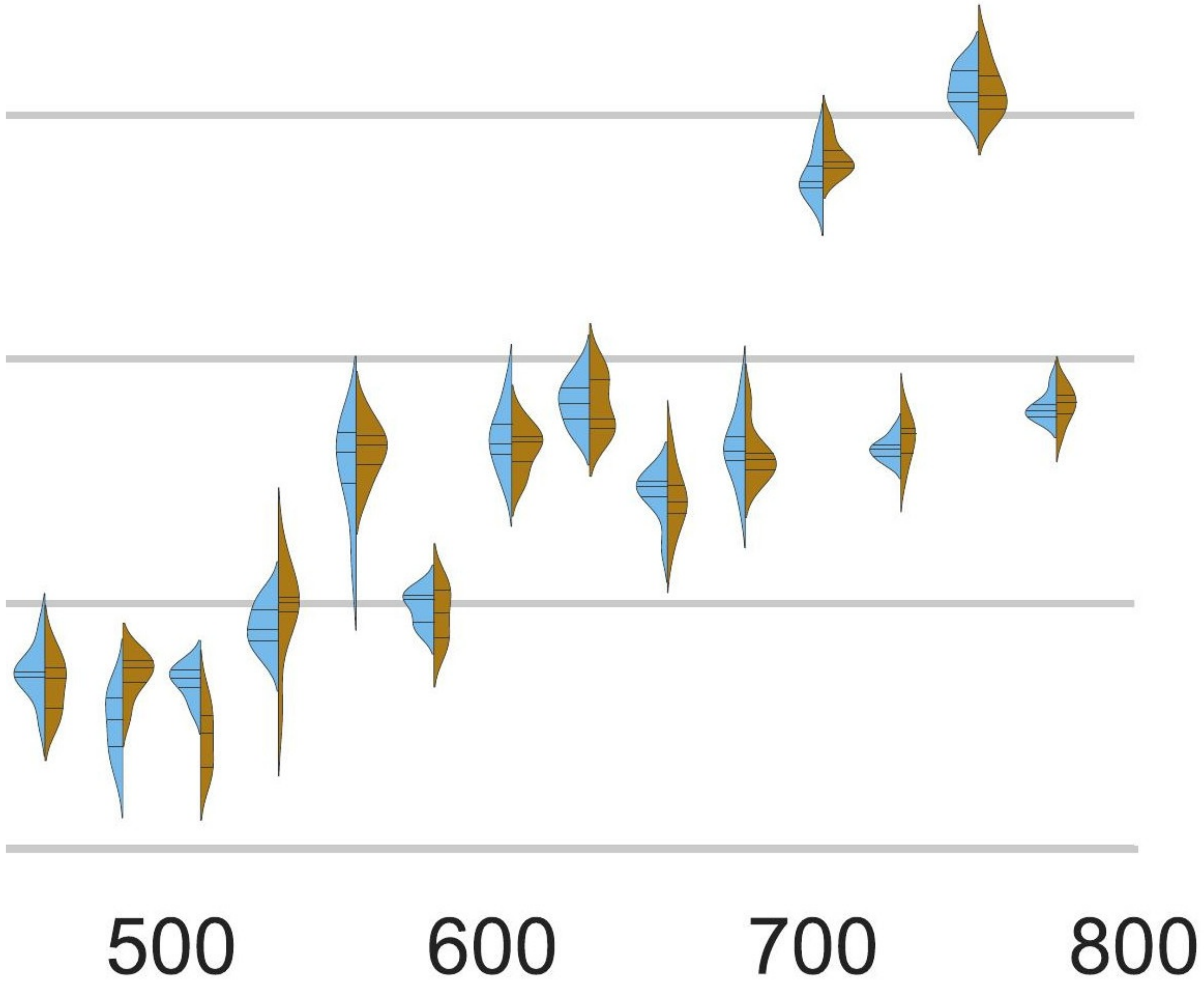
Performance overhead in PaRSEC



Performance overhead in HPCG



Performance overhead in HPCG (zoom)



Conclusions

- High quality scheduling algo. design needs more than heuristics.
- Runtime systems generate multiple useful software “events”.
- PAPI SDE allows any software layer to export events.
- SDEs can be read using the standard PAPI functionality.
- Inserting SDEs to a library is simple and easy.
- SDEs have minimal to zero performance overhead.