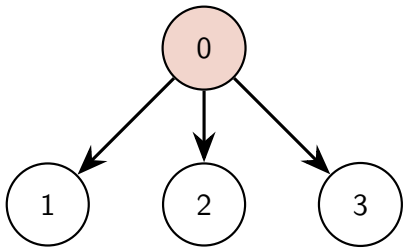
Modelling of MPI Collective Operations: Broadcast and Reduce

Marco Tallone

May 13, 2024

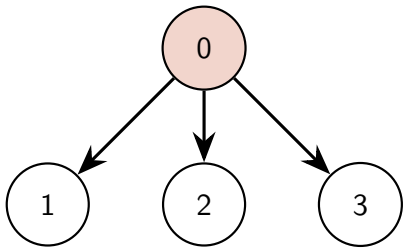
Problem Statement and Objective

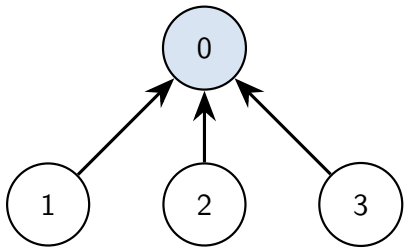
Collective operations studied: **broadcast**



Problem Statement and Objective

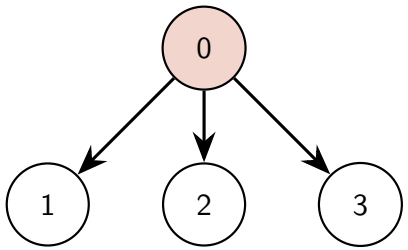
Collective operations studied: **broadcast** and **reduce**

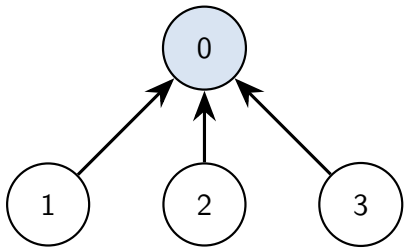




Problem Statement and Objective

Collective operations studied: **broadcast** and **reduce**



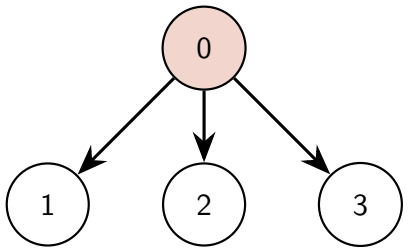


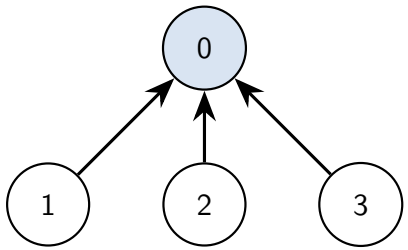
Different algorithms modelled with 2 approaches:

- model based on **point-to-point communications**
- **linear fit** model

Problem Statement and Objective

Collective operations studied: **broadcast** and **reduce**





Different algorithms modelled with 2 approaches:

- model based on **point-to-point communications**
- **linear fit** model

Goal

Predict the latency T of collective operations based on P and m

Broadcast and Reduce Algorithms

Analyzed algorithms:

1. **Default**

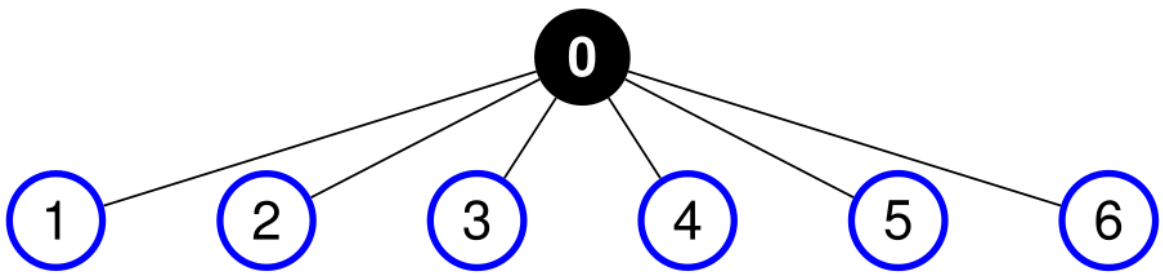
```
1  int bcast_intra_dec_fixed(void *buff,int count,MPI_Datatype
2  *datatype,int root,MPI_comm *comm)
3  {
4  const size_t small_message_size = 2048;
5  const size_t intermediate_message_size = 370728;
6  const double a_p16 = 3.2118e-6;
7  const double b_p16 = 8.7936;
8  const double a_p64 = 2.3679e-6;
9  const double b_p64 = 1.1787;
10 const double a_p128 = 1.6134e-6;
11 const double b_p128 = 2.1102;
12
13 int communicator_size;
14 size_t message_size, dsize;
15
16 communicator_size = MPI_comm_size(comm);
17 MPI_Type_size(datatype, &dsize);
18 message_size = dsize * (unsigned long)count;
19
20 if ((message_size < small_message_size) || (count <= 1)) {
21 return binomial_tree_bcast(. . .);
22 } else if (message_size < intermediate_message_size) {
23 return split_binary_tree_bcast(. . .);
24 } else if (communicator_size < (a_p128 * message_size + b_p128))
       {
25 return chain_bcast(. . . );
26 } else if (communicator_size < 13) {
27 return split_binary_tree_bcast(. . .);
28 } else if (communicator_size < (a_p64 * message_size + b_p64)) {
29 return chain_bcast(. . .);
30 } else if (communicator_size < (a_p16 * message_size + b_p16)) {
31 return chain_bcast(. . .);
32 }
33 return chain_bcast(. . .);
34 }
```

Broadcast and Reduce Algorithms

Analyzed algorithms:

2. **Linear**

$$T^c_{p2p}(m) \leq T^c_{NBFT}(P, m) \leq T^c_{BFT}(P, m) = (P-1) \cdot T^c_{p2p}(m)$$



Broadcast and Reduce Algorithms

Analyzed algorithms:

2. **Linear**

$$T^c_{p2p}(m) \leq T^c_{NBFT}(P,m) \leq T^c_{BFT}(P,m) = (P-1) \cdot T^c_{p2p}(m)$$

$$\Downarrow$$

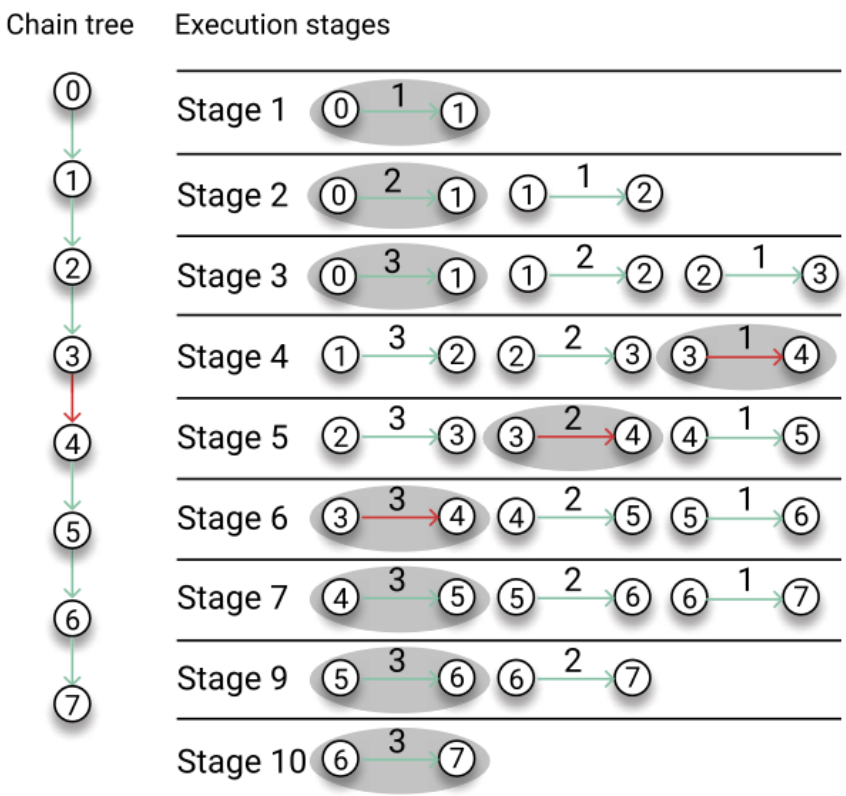
$$T^c_{NBFT}(P,m) \approx \gamma^c(P,m) \cdot T^c_{p2p}(m)$$

where $\gamma^c(P,m)$ is a **parallelization factor**

Broadcast and Reduce Algorithms

Analyzed algorithms:

3 **Chain** $T_{chain}(P, m, n_s) = \sum_{i=1}^{P+n_s-2} \max_{j_i} T_{NBFT}^{c_{j_i}}(2, m_s)$

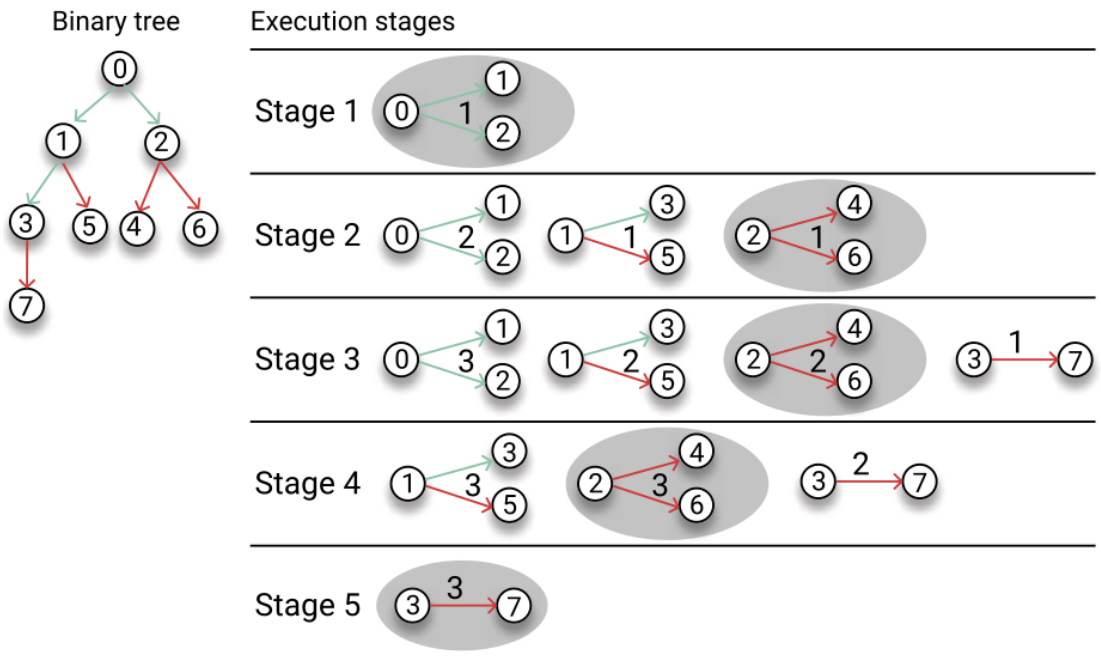


Broadcast and Reduce Algorithms

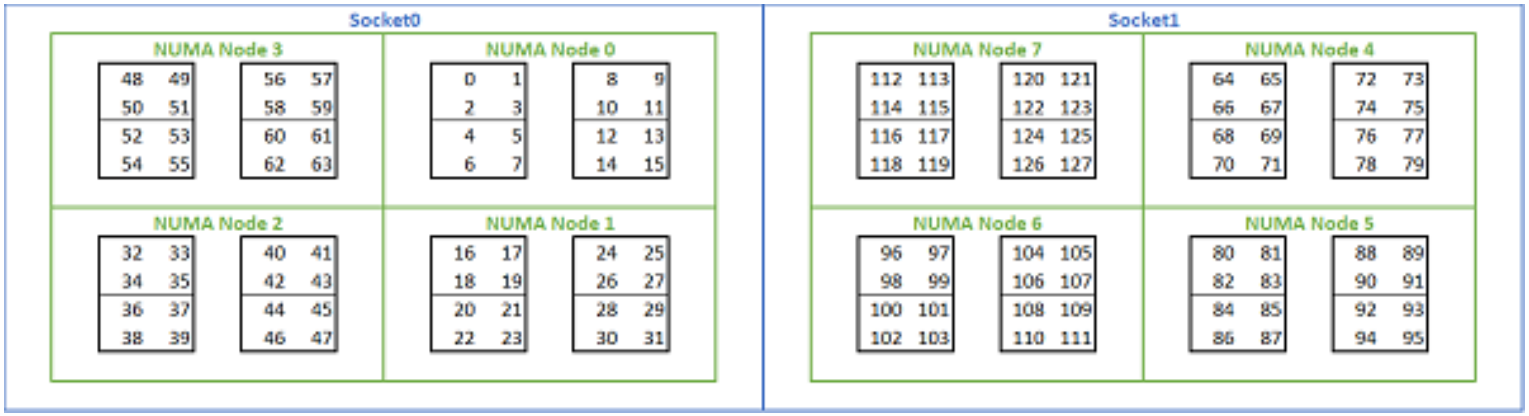
Analyzed algorithms:

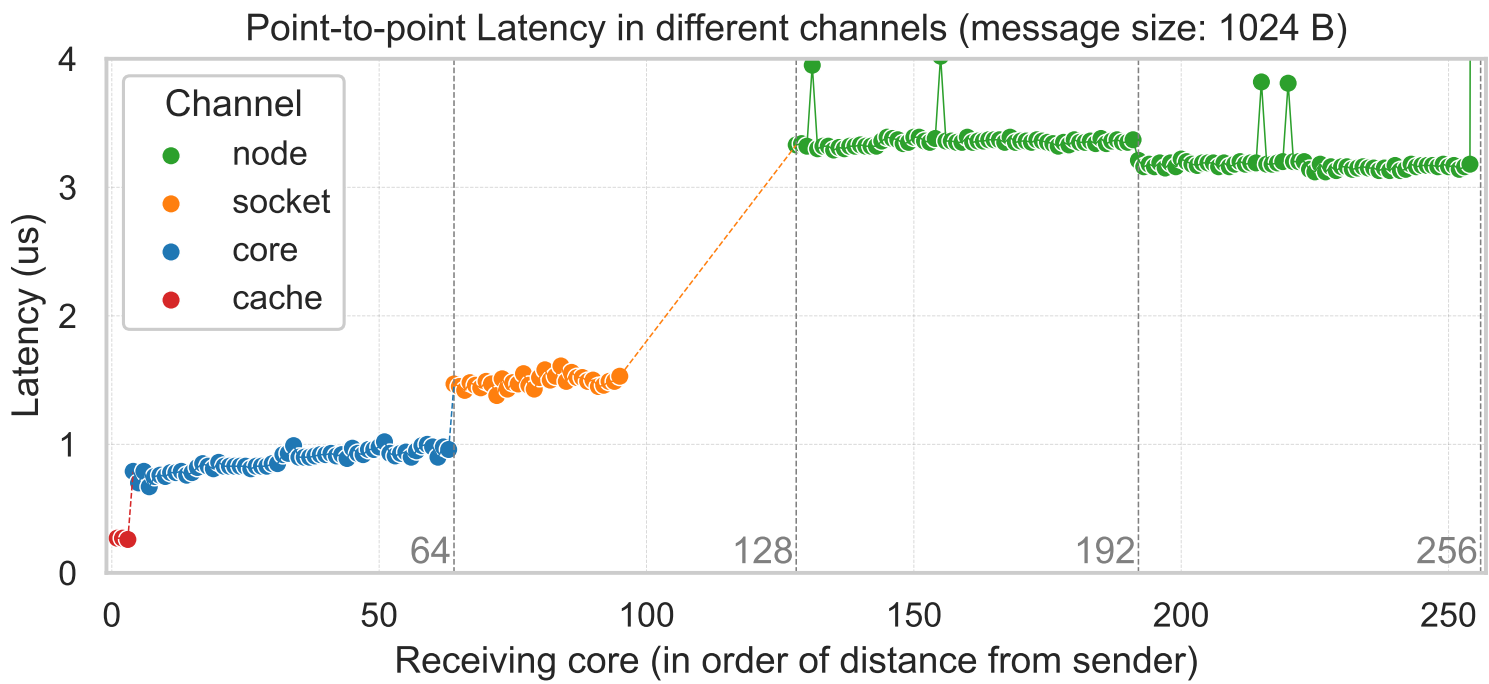
4 **Binary Tree**

$$T_{binary}(P, m, n_s) = \sum_{i=1}^{\lfloor \log_2(P) \rfloor + n_s - 1} \max_{j_i} T_{NBFT}^{c_{j_i}}(P_i, m_s)$$



EPYC Architecture and P2P Communications





Parameters Estimation

1 Hockney Model Fit for P2P Communications

$$\widehat{T}^{c}_{p2p}(m) = \alpha^{c} + \beta^{c} \cdot m$$

Parameters Estimation

1 Hockney Model Fit for P2P Communications

$$\widehat{T}^{c}_{p2p}(m) = \alpha^{c} + \beta^{c} \cdot m$$

2 Measure and Fit NBFT for $P \in [2, P^{\max(c)}_{NBFT}]$

$$\widehat{T}^{c,m}_{NBFT}(P) = \alpha^{c,m} + \beta^{c,m} \cdot (P - 1)$$

Parameters Estimation

1 Hockney Model Fit for P2P Communications

$$\widehat{T}^{c}_{p2p}(m) = \alpha^{c} + \beta^{c} \cdot m$$

2 Measure and Fit NBFT for $P \in [2, P^{\max(c)}_{NBFT}]$

$$\widehat{T}^{c,m}_{NBFT}(P) = \alpha^{c,m} + \beta^{c,m} \cdot (P-1)$$

3 Compute Parallelization Factor

$$\widehat{\gamma}^{c}(P,m) = \widehat{T}^{c,m}_{NBFT}(P) / \widehat{T}^{c}_{p2p}(m) \quad \forall P \in [2, P^{\max(c)}_{NBFT}] \ , \ \forall m$$

Parameters Estimation

1 Hockney Model Fit for P2P Communications

$$\widehat{T}^{c}_{p2p}(m) = \alpha^{c} + \beta^{c} \cdot m$$

2 Measure and Fit NBFT for $P \in [2, P^{\max(c)}_{NBFT}]$

$$\widehat{T}^{c,m}_{NBFT}(P) = \alpha^{c,m} + \beta^{c,m} \cdot (P-1)$$

3 Compute Parallelization Factor

$$\widehat{\gamma}^{c}(P,m) = \widehat{T}^{c,m}_{NBFT}(P) / \widehat{T}^{c}_{p2p}(m) \quad \forall P \in [2, P^{\max(c)}_{NBFT}] \ , \ \forall m$$

4 $\Rightarrow$ Latency Prediction for NBFT using **only** channel c

$$T^{c}_{NBFT}(P,m) \approx \widehat{\gamma}^{c}(P,m) \cdot \widehat{T}^{c}_{p2p}(m)$$

Generalization to more than one channel

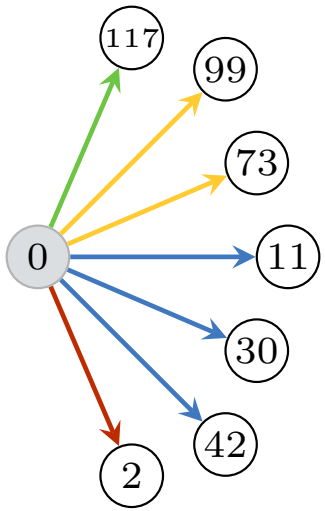
5 Ratio of Delays

$$Q_{i,j}(m) = \widehat{T}^{c_i}_{p2p}(m) / \widehat{T}^{c_j}_{p2p}(m) \quad , \forall m, \forall i,j \in \{0,1,2,3\} |\ i > j$$

Generalization to more than one channel

5 Ratio of Delays

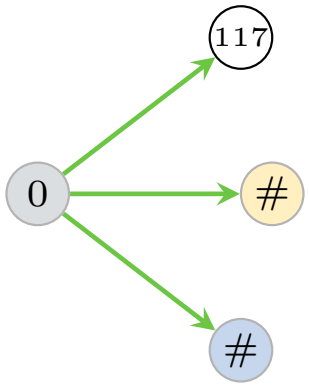
$$Q_{i,j}(m) = \widehat{T}^{c_i}_{p2p}(m) / \widehat{T}^{c_j}_{p2p}(m) \quad , \forall m, \ \forall i,j \in \{0,1,2,3\} |\ i > j$$



$T^{\bullet,\bullet,\bullet,\bullet}_{NBFT}(8,m)$

becomes

$Q_{\bullet,\bullet}(m) = 2$
$Q_{\bullet,\bullet}(m) = 3$
$Q_{\bullet,\bullet}(m) = 7$



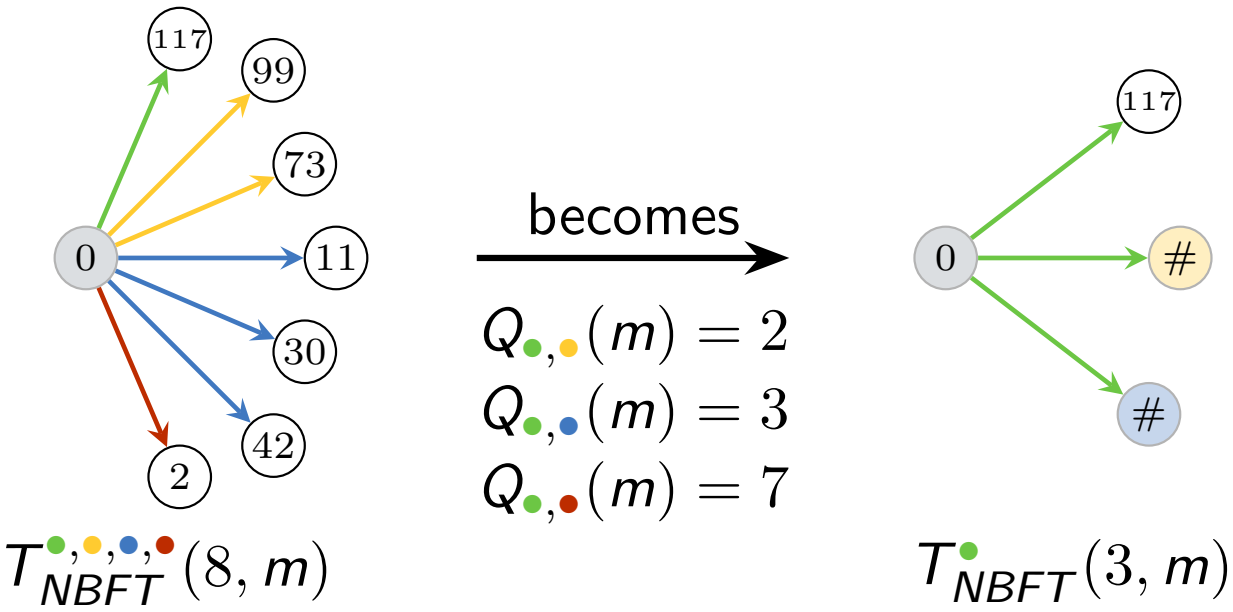
$T^{\bullet}_{NBFT}(3,m)$

where: • : **cache** < • : **core**, < • : **socket**, < • : **node**

Generalization to more than one channel

5 Ratio of Delays

$$Q_{i,j}(m) = \widehat{T}^{c_i}_{p2p}(m) / \widehat{T}^{c_j}_{p2p}(m) \quad , \forall m, \ \forall i,j \in \{0,1,2,3\} |\ i > j$$



where: • : **cache** < • : **core**, < • : **socket**, < • : **node**

⇒ **platform-dependent** but **algorithm-independent** model

Linear Fit Model

$$T^m_{\text{latency}}(P) = \beta_0 + \beta_1 P + \sum_{i=1}^{3} (\beta_{2,i} z_i + \beta_{3,i}(P \cdot z_i))$$

- `map-by core`: binary dummy variable $\mathbf{z}$ identifies which sockets are used
- `map-by socket`: binary dummy variable $\mathbf{z}$ identifies which nodes are used
- `map-by node`: no dummy variable needed, simple fit

⚠ For **binary tree** fit w.r.t $\log_2(P)$

$\Rightarrow$ **platform-dependent** + **algorithm-dependent** model

Broadcast and Reduce Results

Broadcast (`map-by`)

-  Default `core`
-  Default `socket`
-  Default `node`

-  Linear `core`
-  Linear `socket`
-  Linear `node`

-  Chain `core`
-  Chain `socket`
-  Chain `node`

-  Binary `core`
-  Binary `socket`
-  Binary `node`

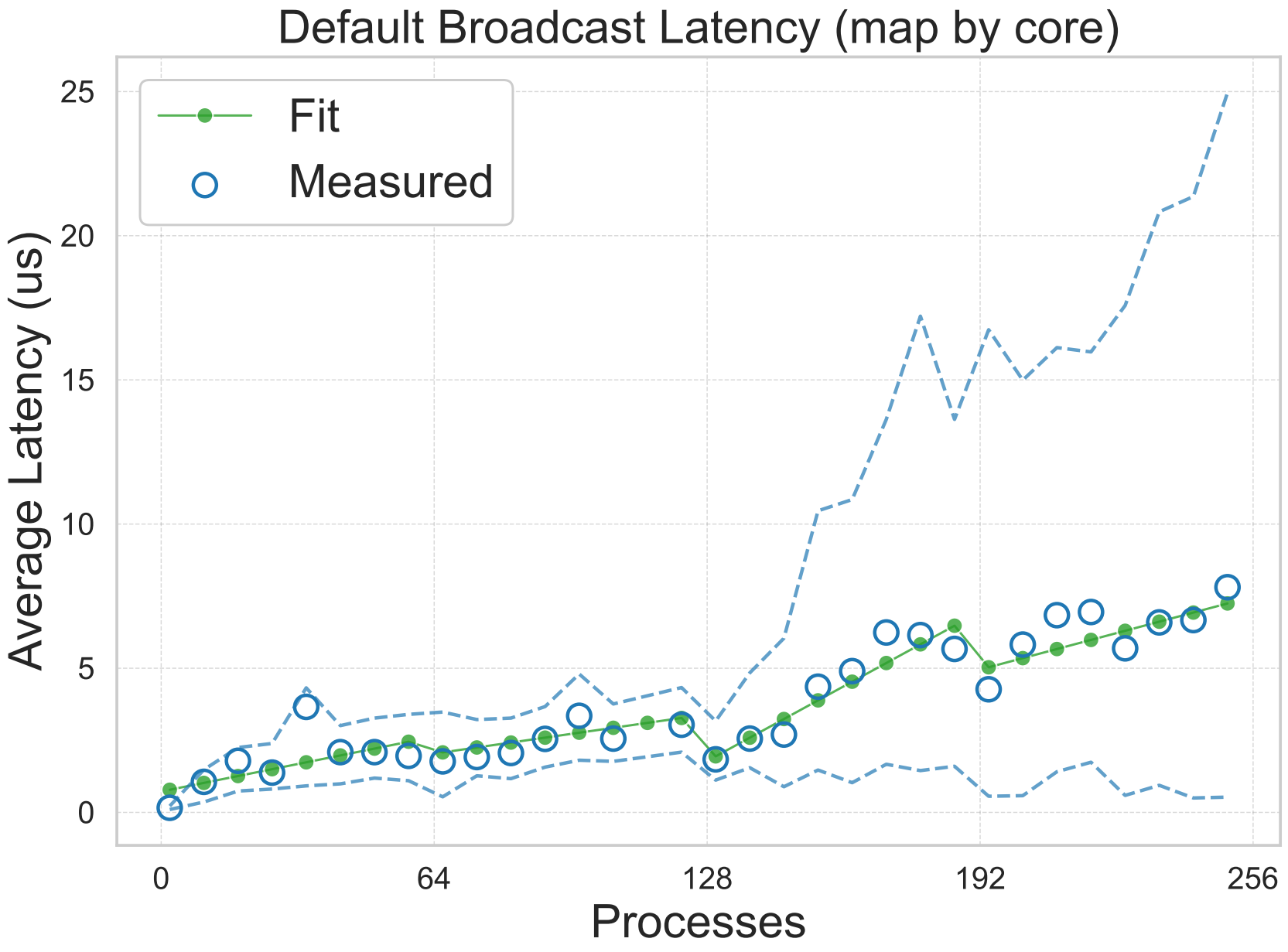
Reduce (`map-by`)

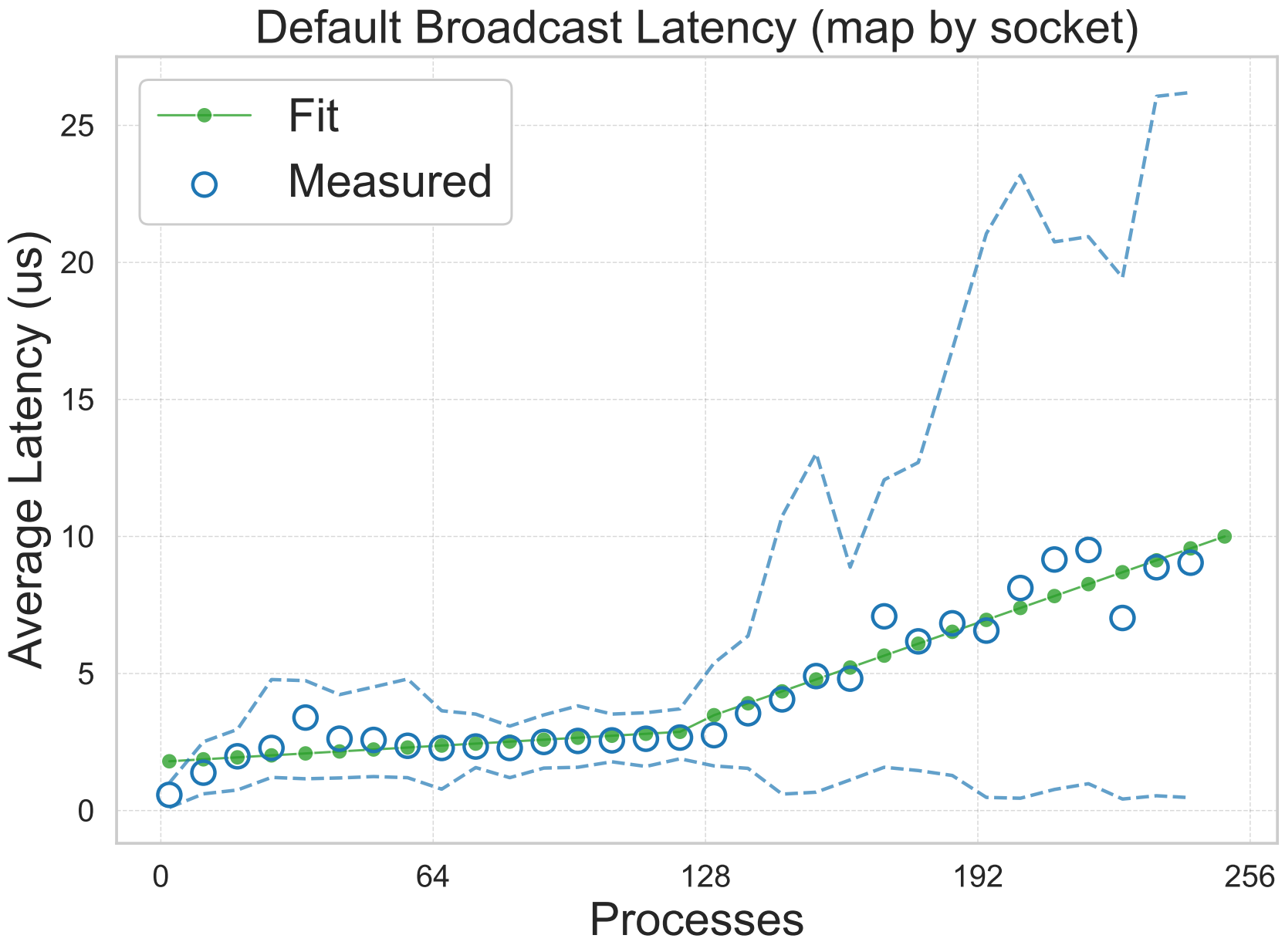
-  Default `core`
-  Default `socket`
-  Default `node`

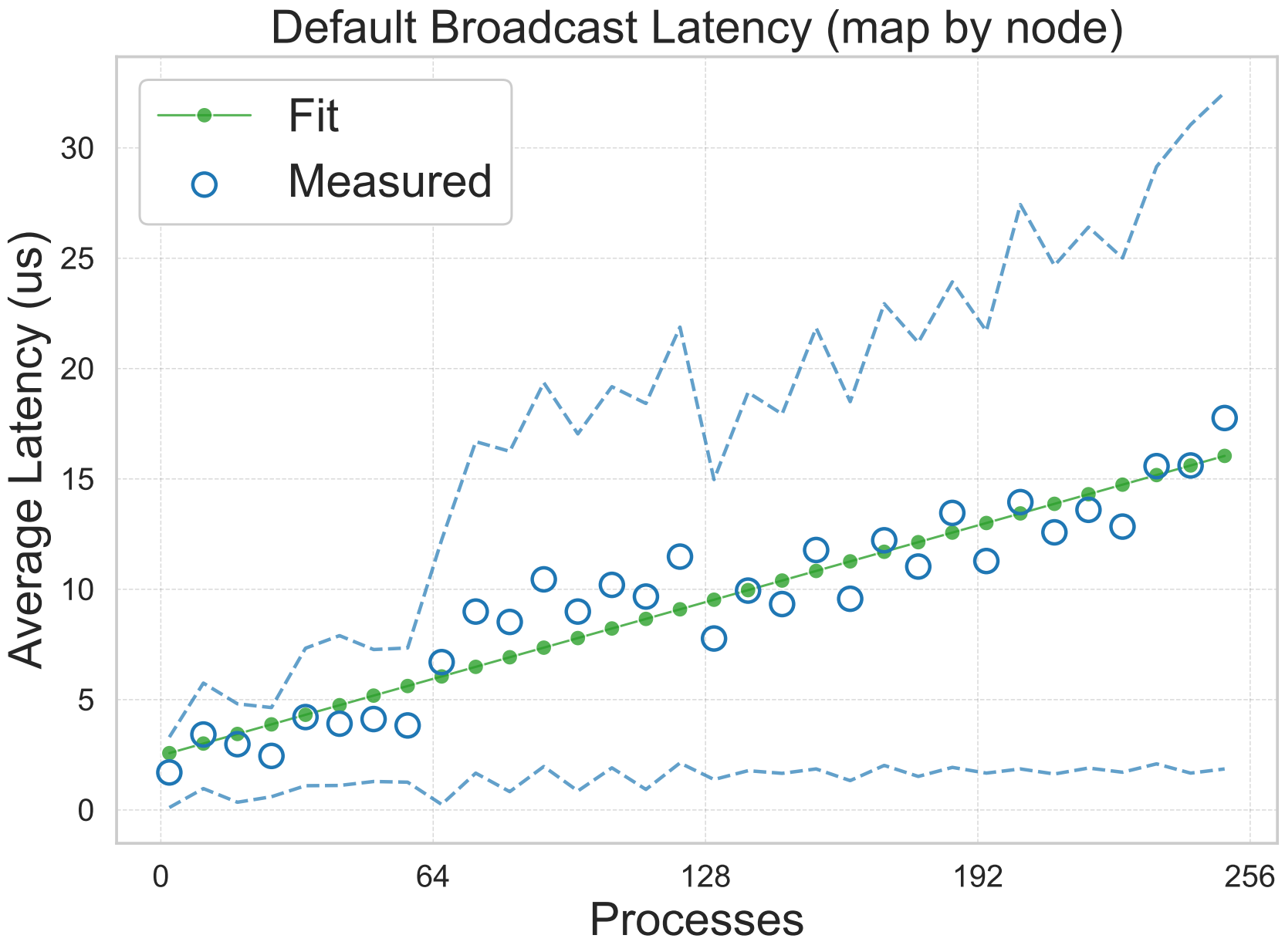
-  Linear `core`
-  Linear `socket`
-  Linear `node`

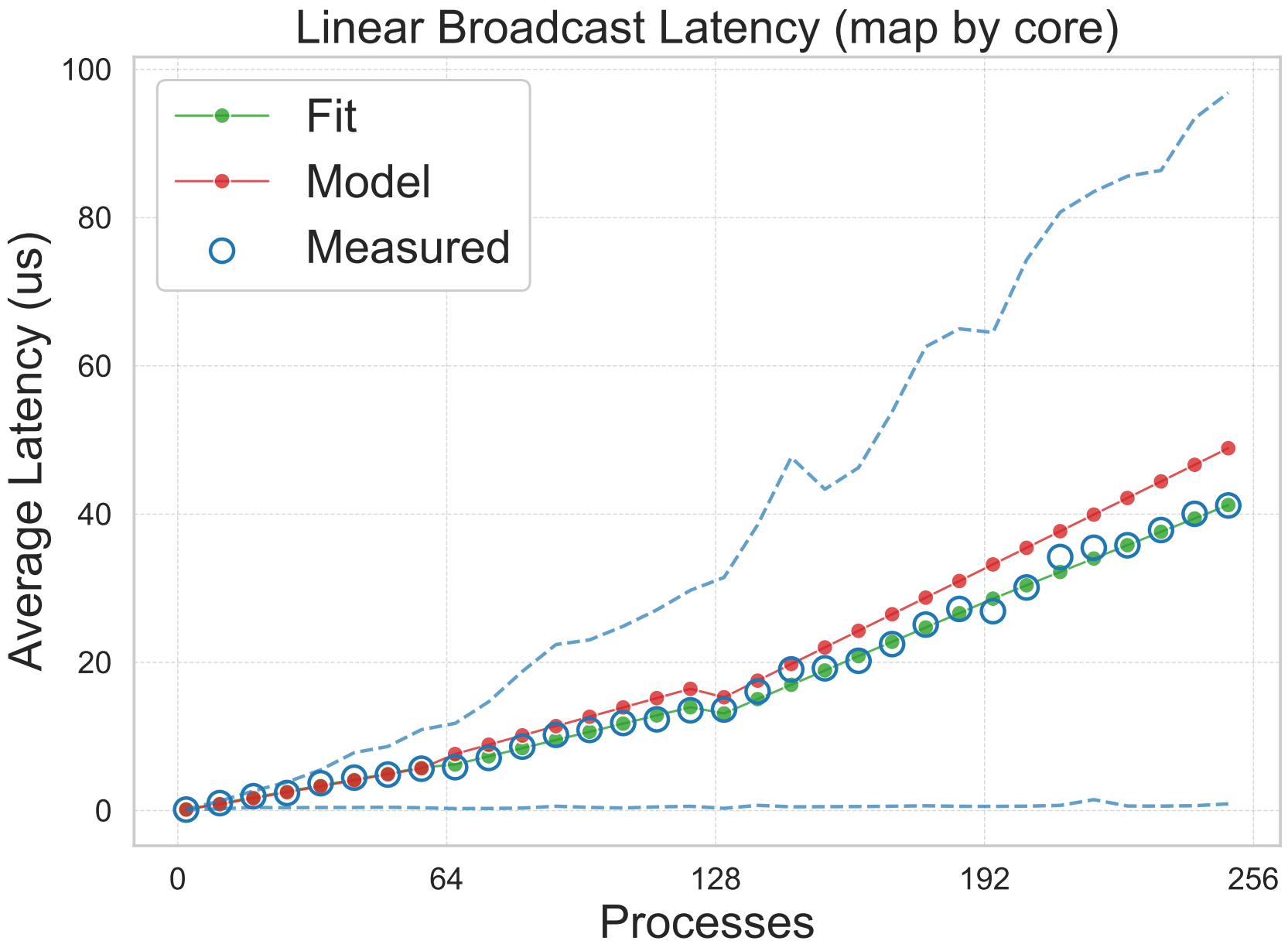
-  Chain `core`
-  Chain `socket`
-  Chain `node`

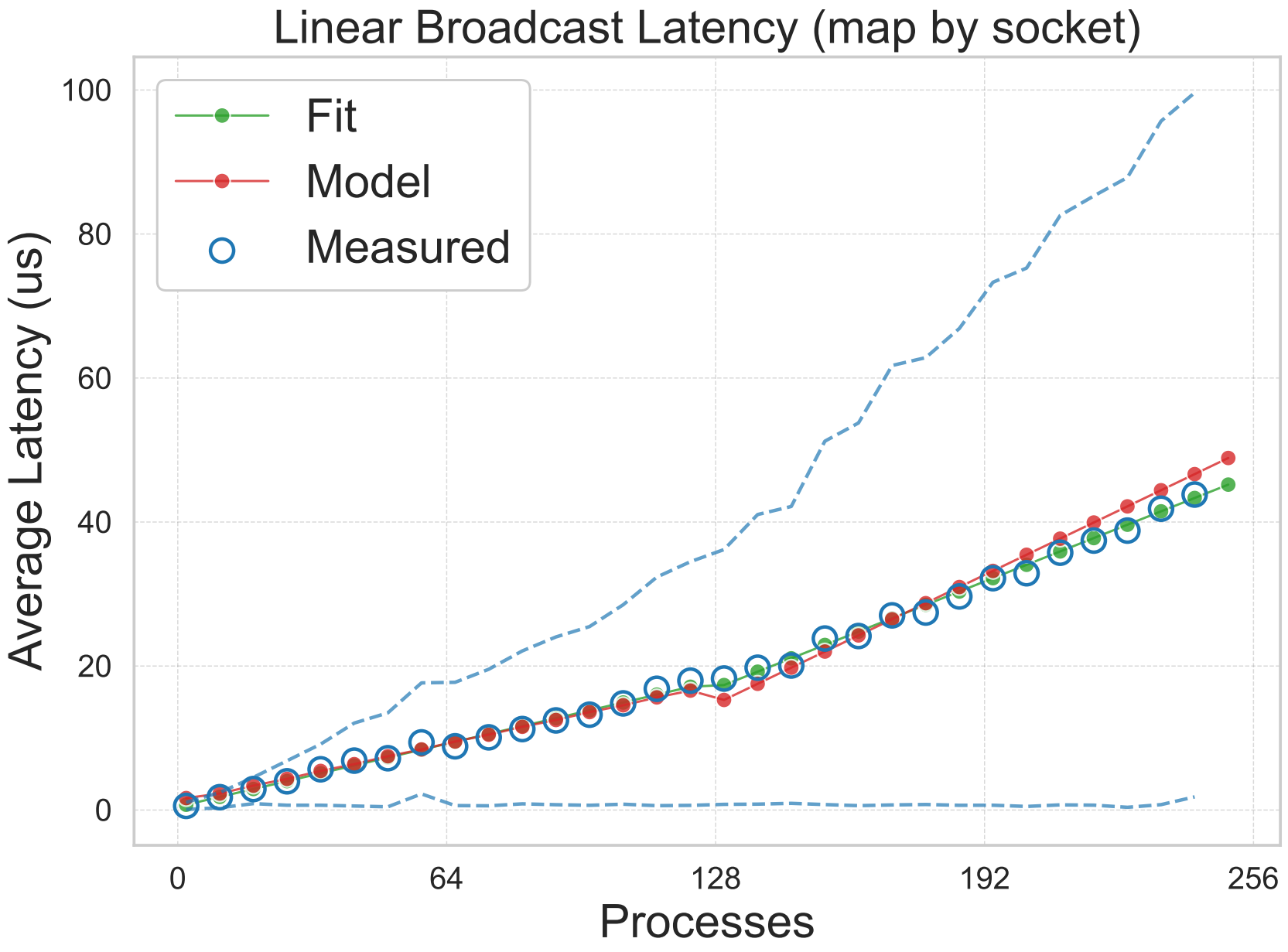
-  Binary `core`
-  Binary `socket`
-  Binary `node`

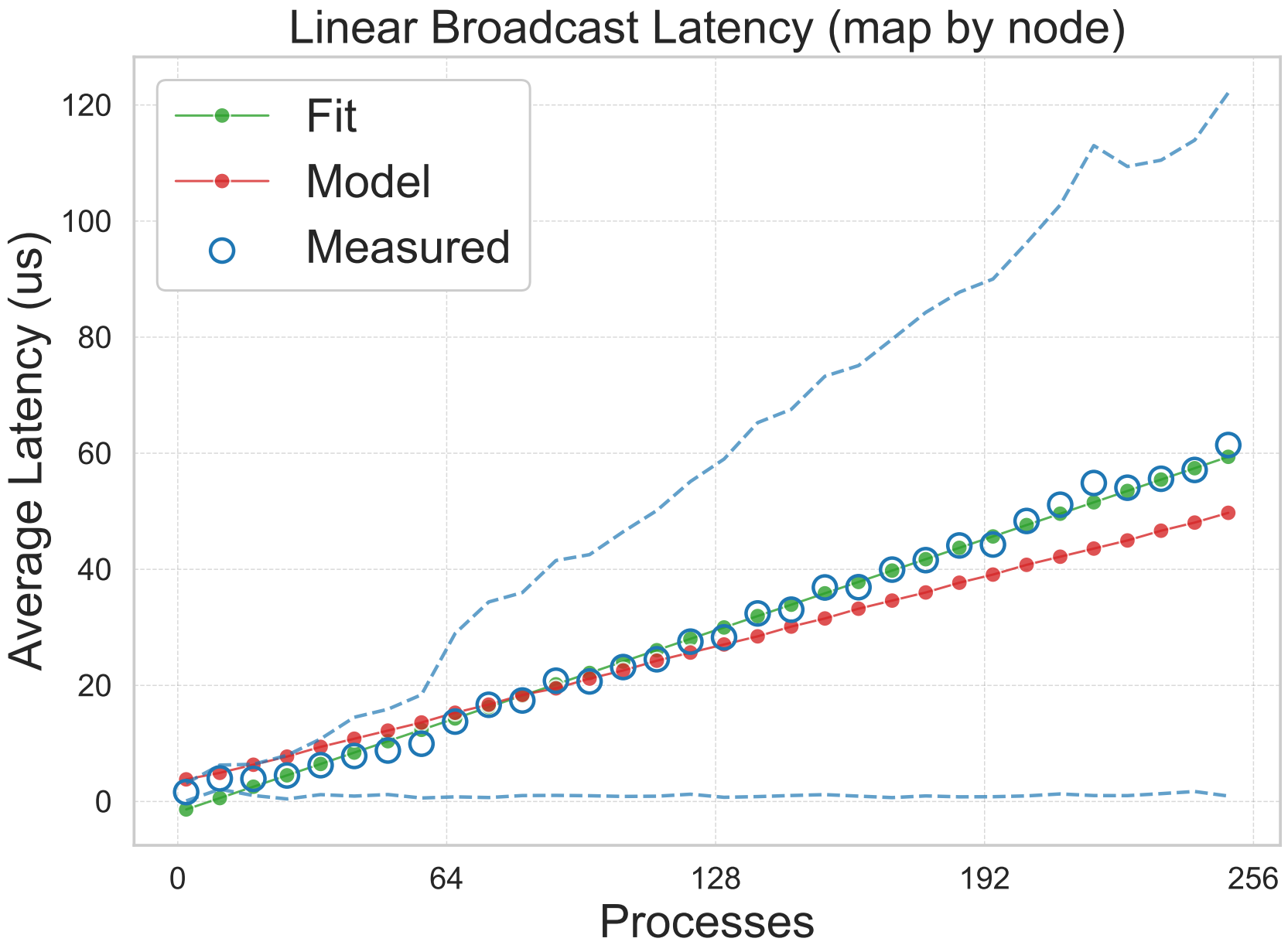

Default Broadcast Latency (map by core)
Fit
Measured
Average Latency (us)
0
5
10
15
20
25
0
64
128
192
256
Processes

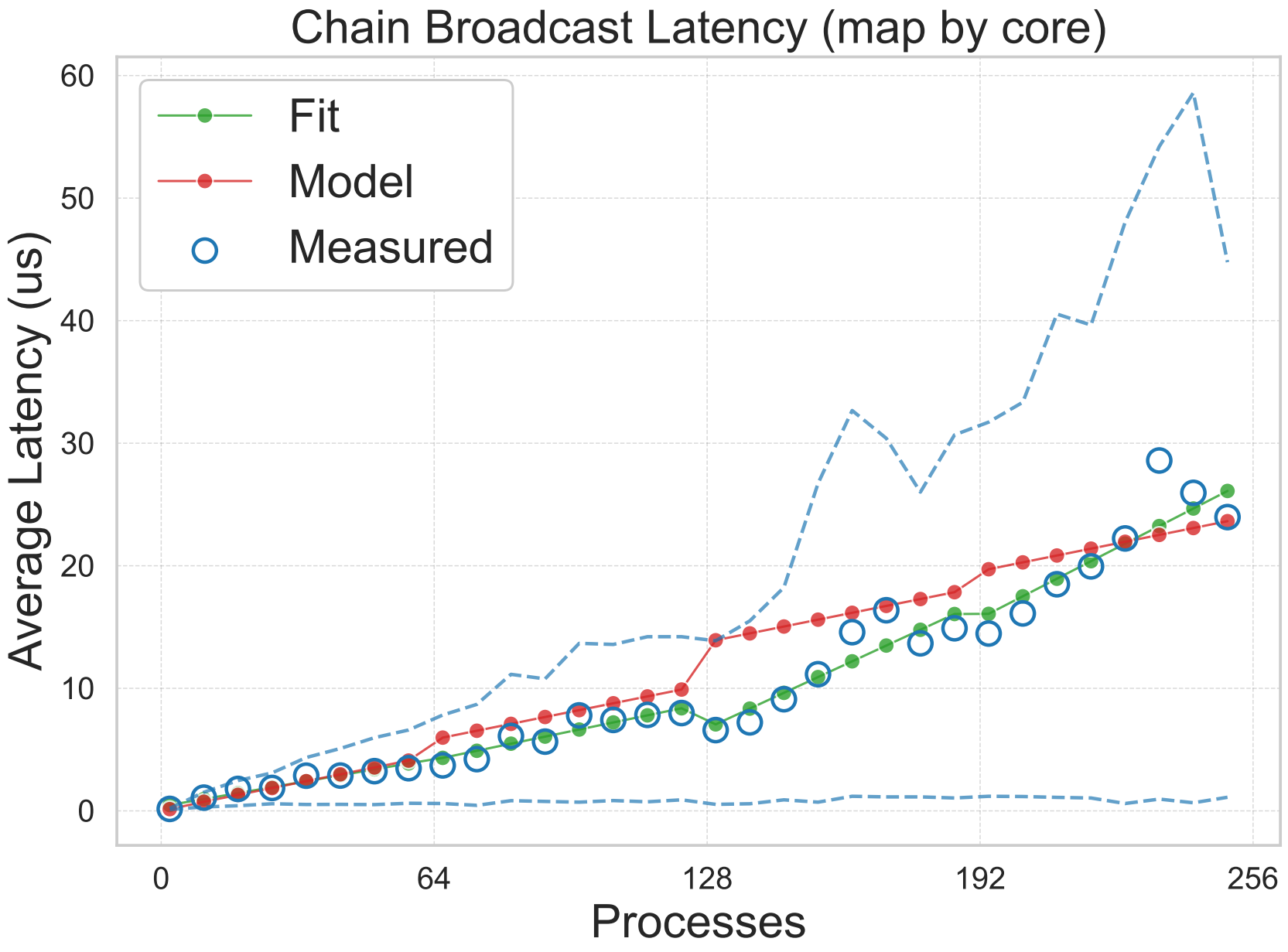

Default Broadcast Latency (map by socket)
Fit
Measured
Average Latency (us)
0
5
10
15
20
25
0
64
128
192
256
Processes

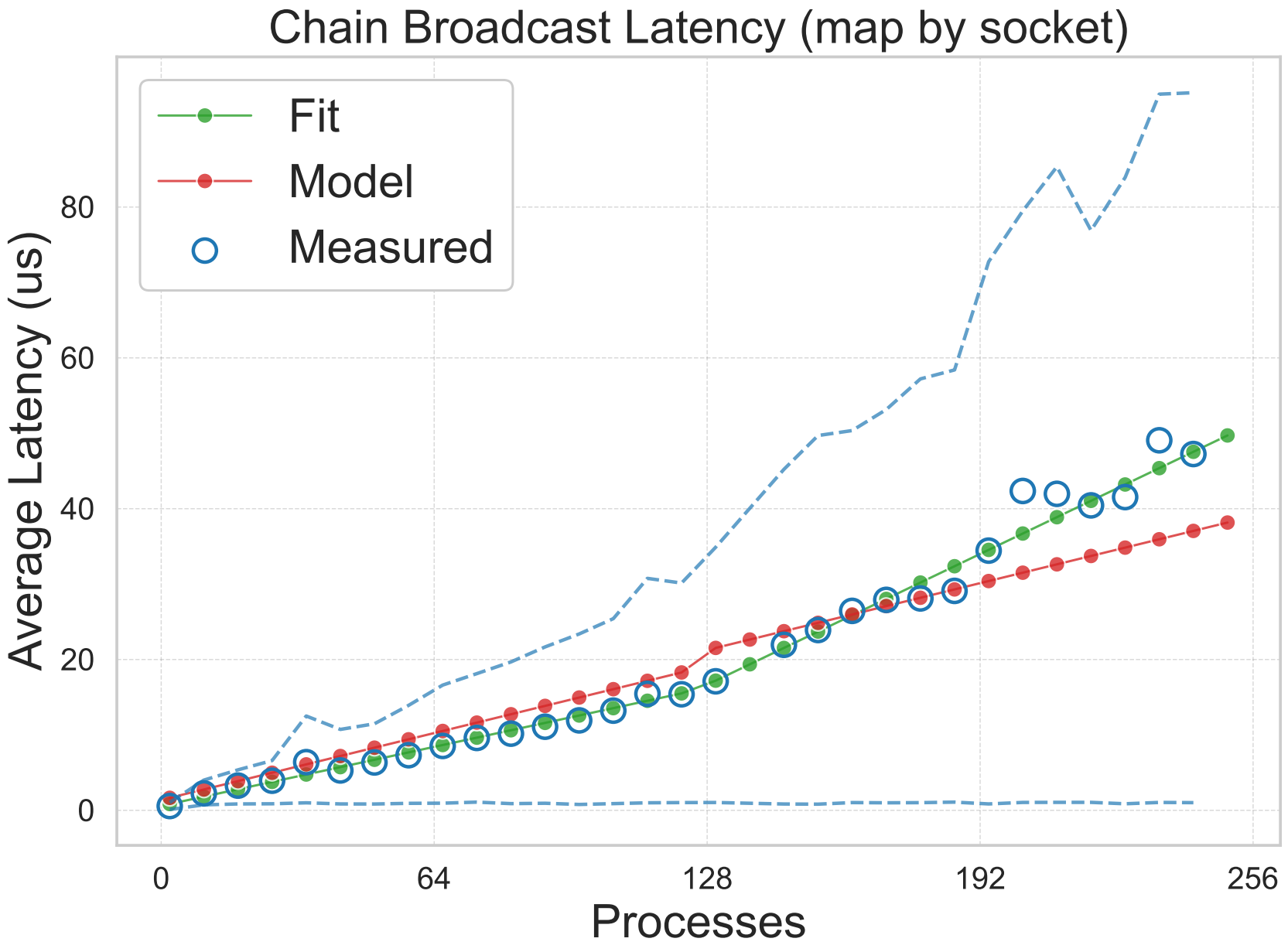

Default Broadcast Latency (map by node)
Fit
Measured
Average Latency (us)
0
5
10
15
20
25
30
0
64
128
192
256
Processes

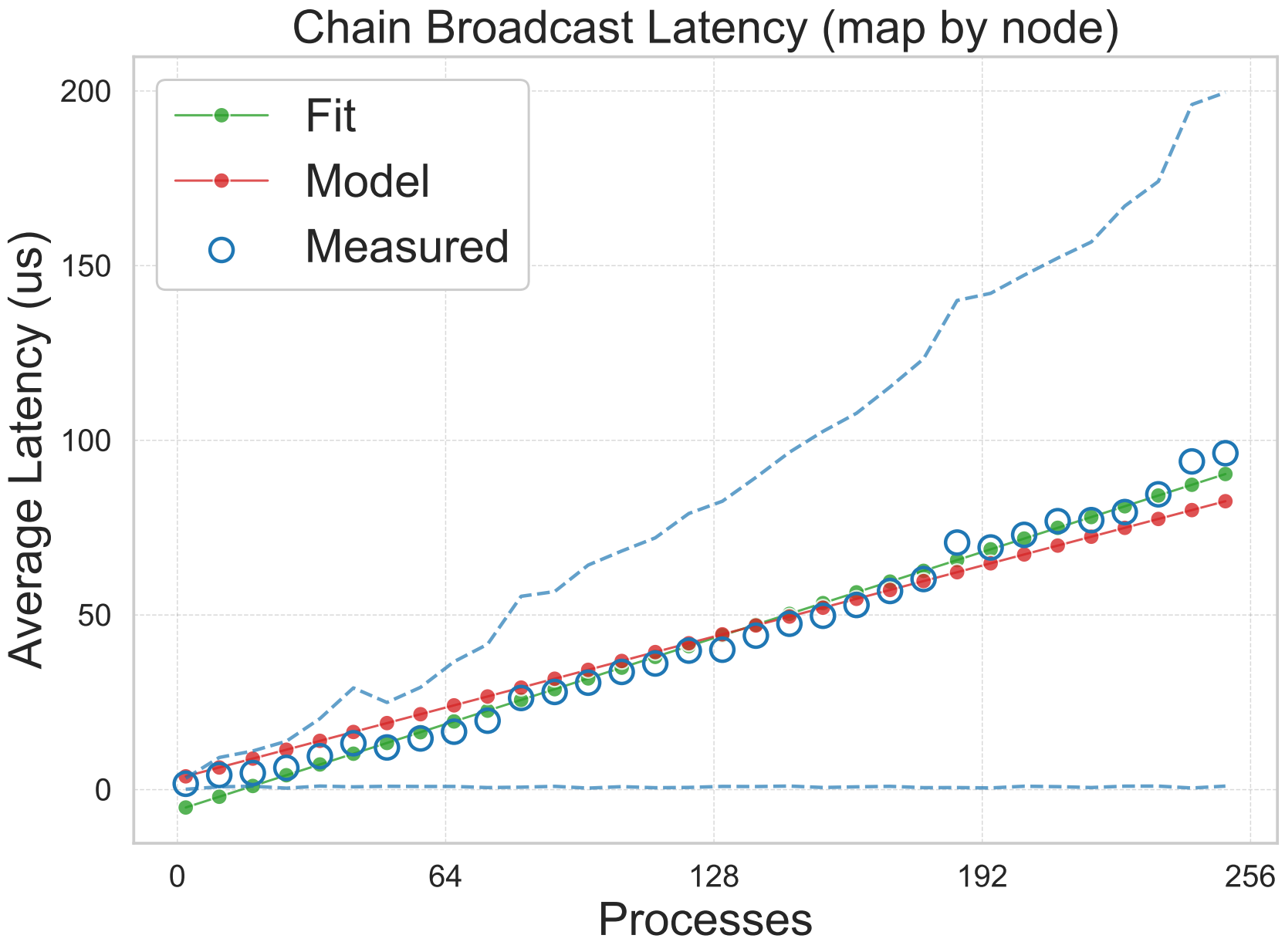

Linear Broadcast Latency (map by core)
Fit
Model
Measured
Average Latency (us)
100
80
60
40
20
0
0
64
128
192
256
Processes

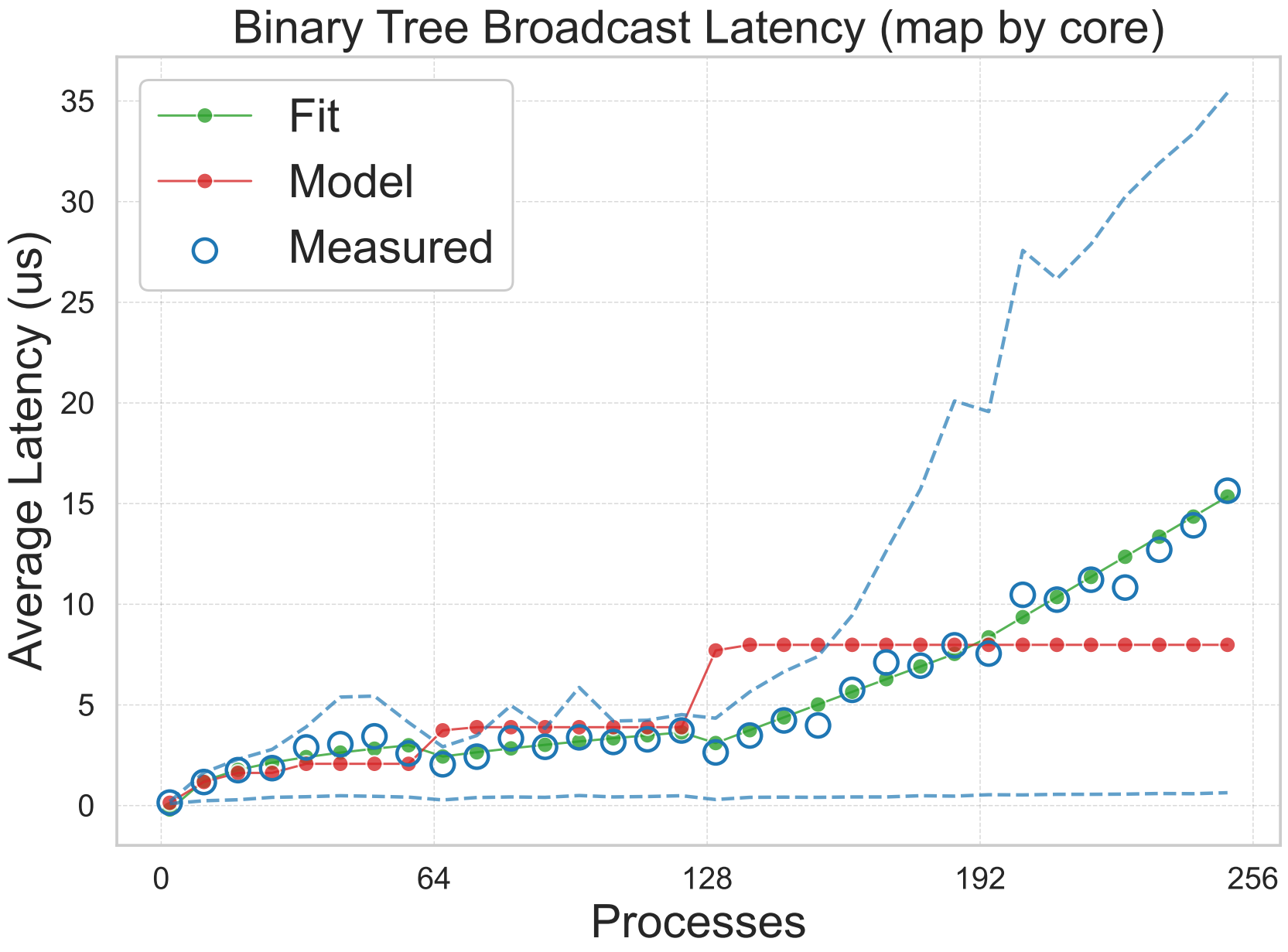

Linear Broadcast Latency (map by socket)
Fit
Model
Measured
Average Latency (us)
100
80
60
40
20
0
0
64
128
192
256
Processes

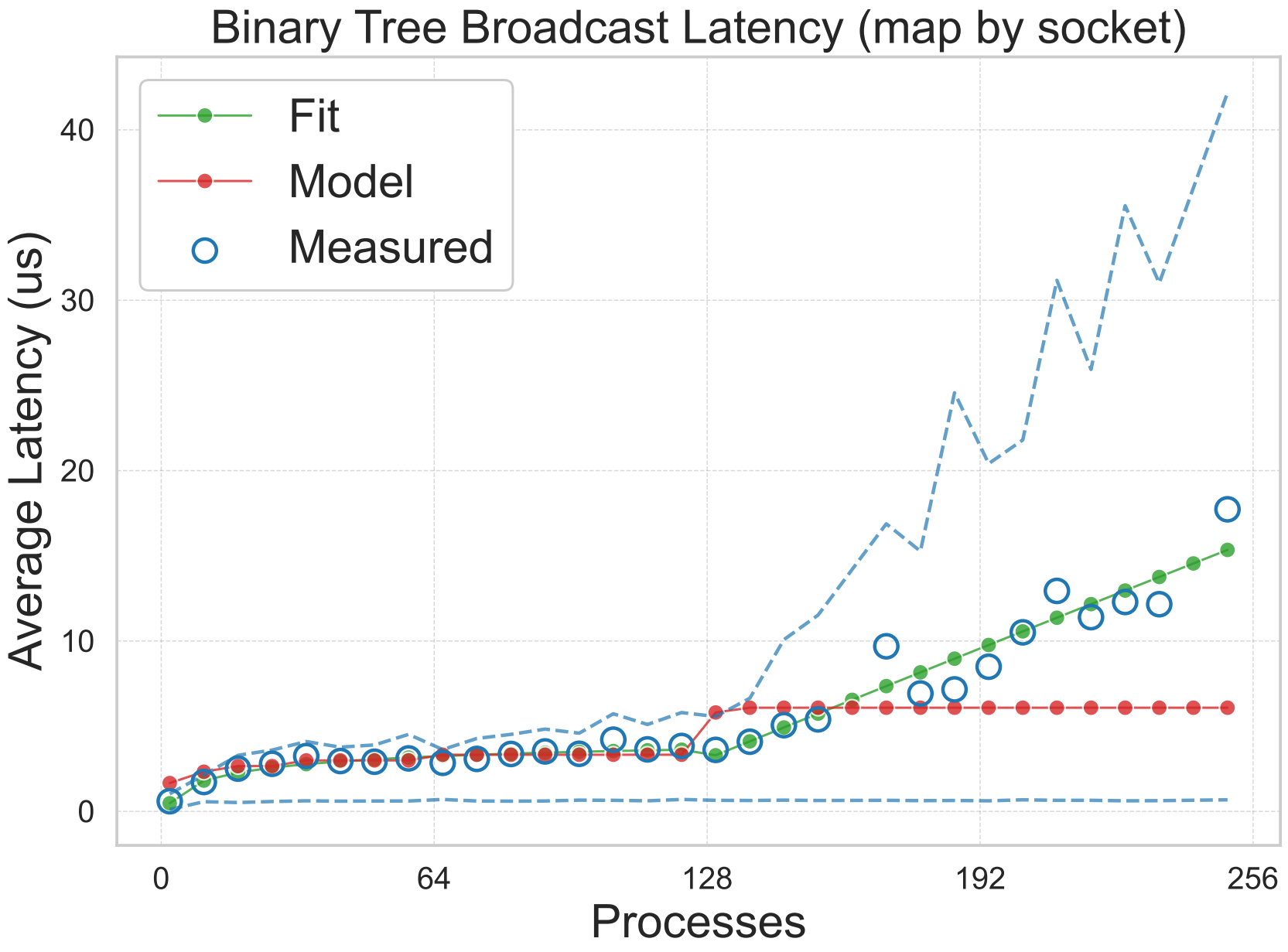

Linear Broadcast Latency (map by node)
Fit
Model
Measured
Average Latency (us)
120
100
80
60
40
20
0
0
64
128
192
256
Processes

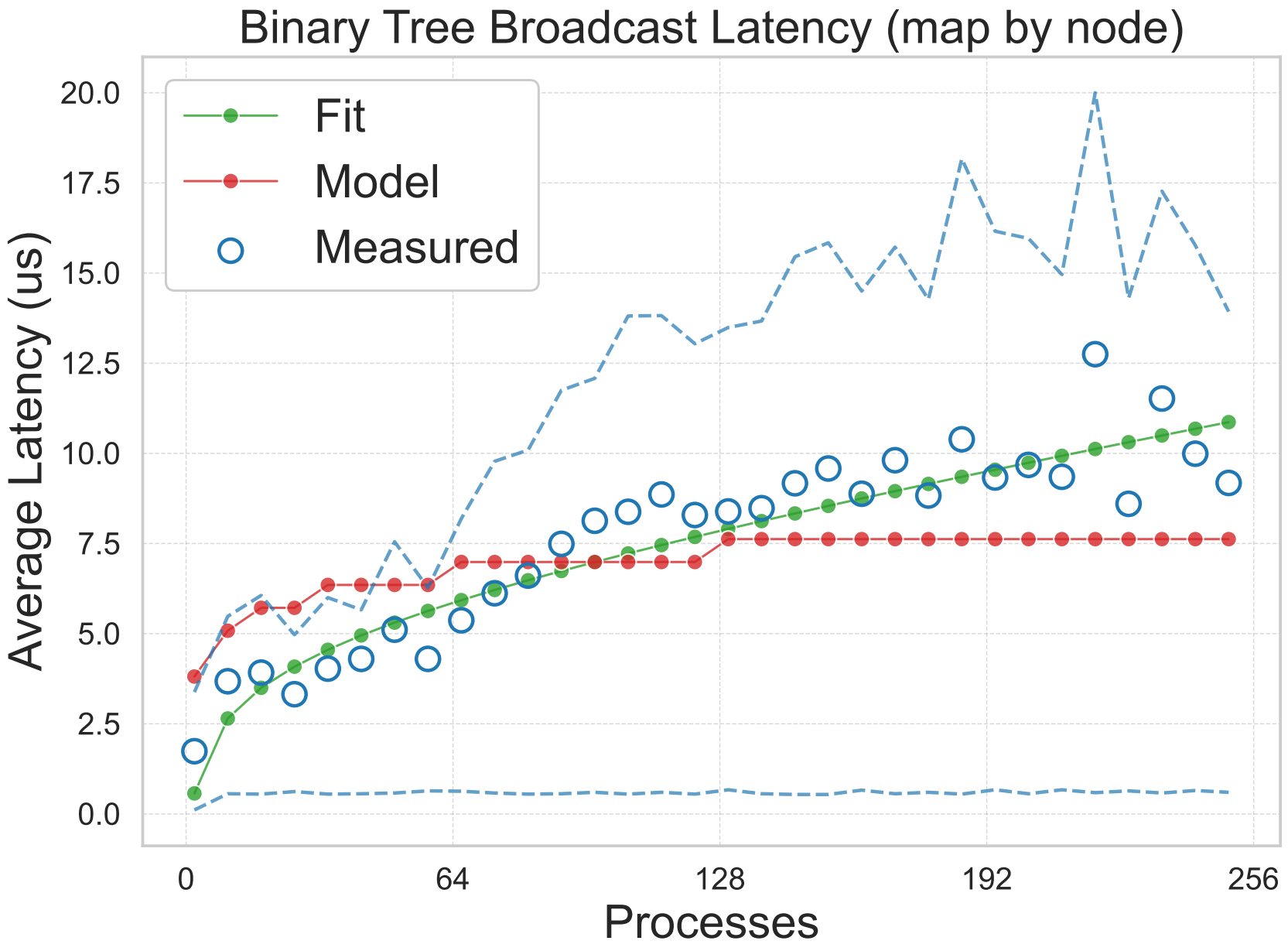

Chain Broadcast Latency (map by core)
Fit
Model
Measured
Average Latency (us)
0
10
20
30
40
50
60
0
64
128
192
256
Processes

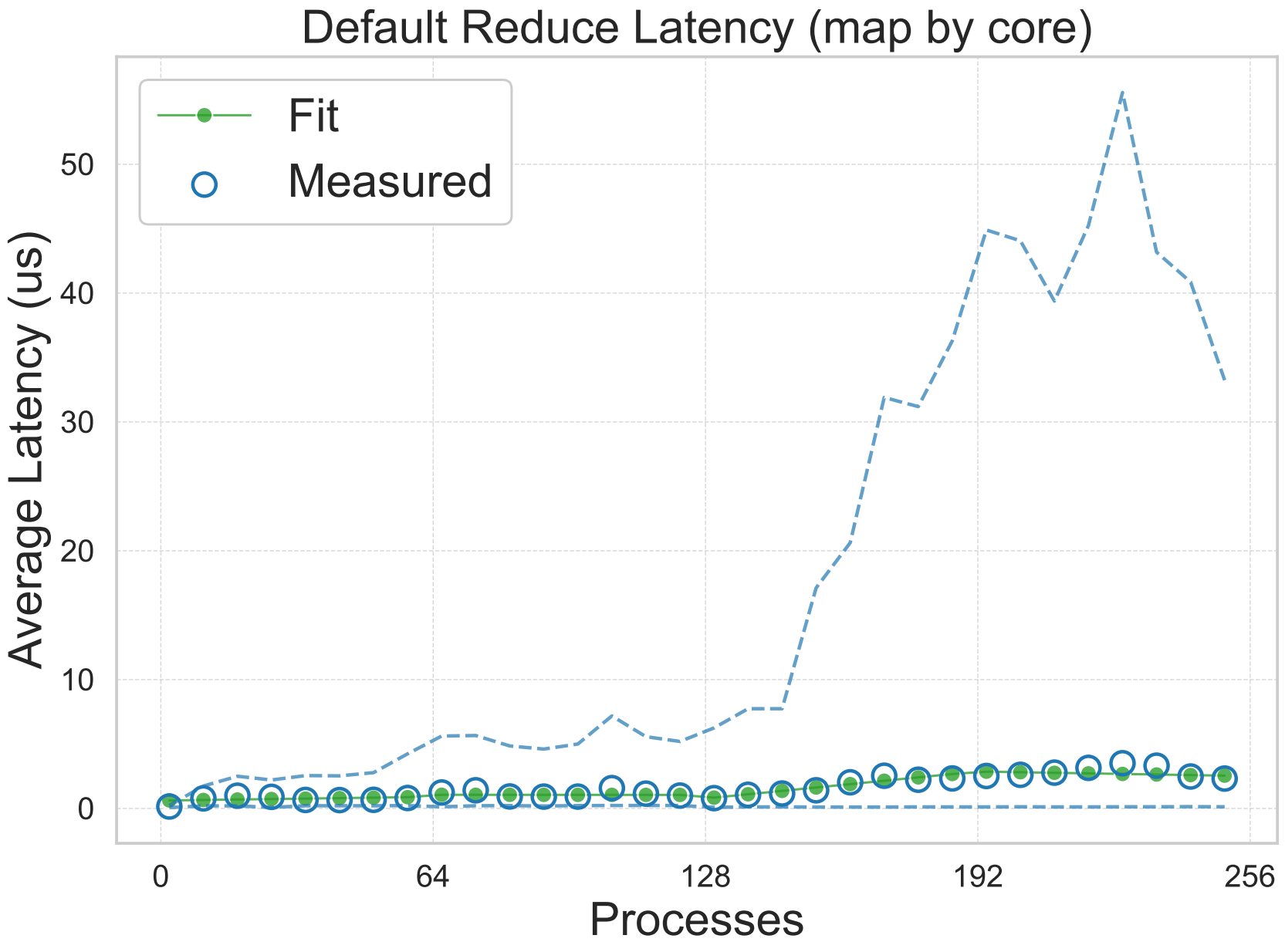

Chain Broadcast Latency (map by socket)
Fit
Model
Measured
Average Latency (us)
0
20
40
60
80
0
64
128
192
256
Processes

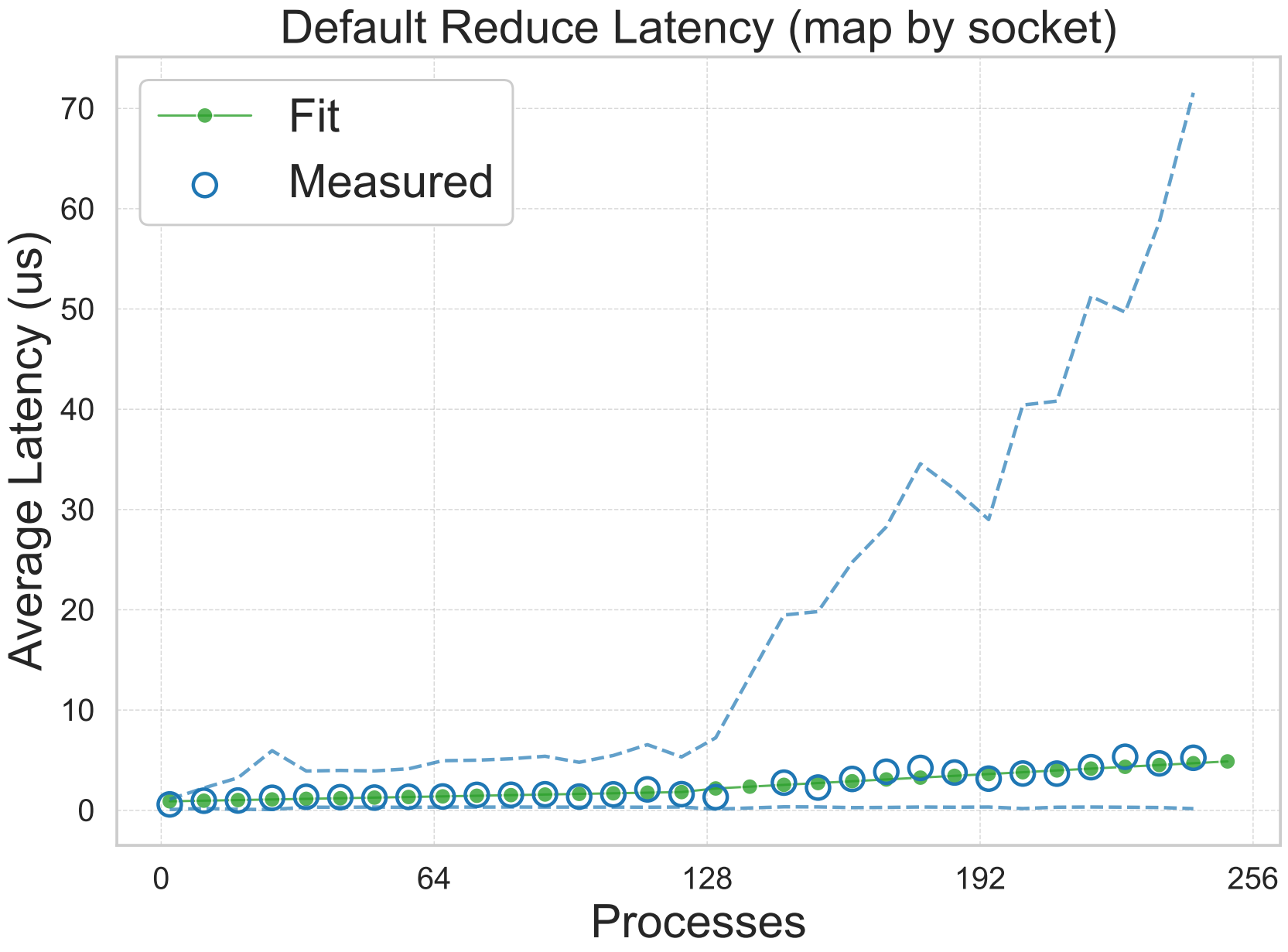

Chain Broadcast Latency (map by node)
Fit
Model
Measured
Average Latency (us)
200
150
100
50
0
0
64
128
192
256
Processes

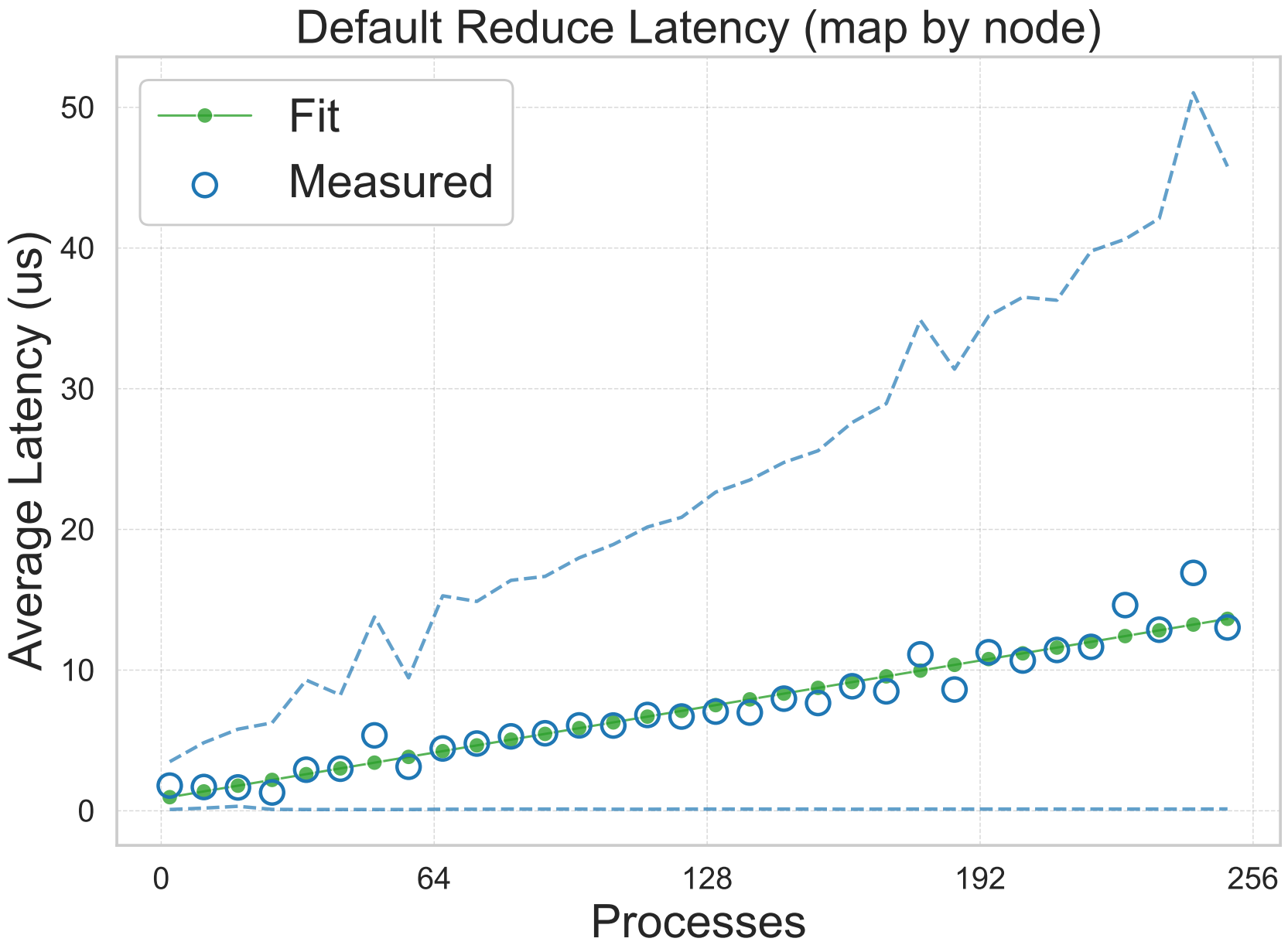

Binary Tree Broadcast Latency (map by core)
Fit
Model
Measured
Average Latency (us)
0
5
10
15
20
25
30
35
0
64
128
192
256
Processes

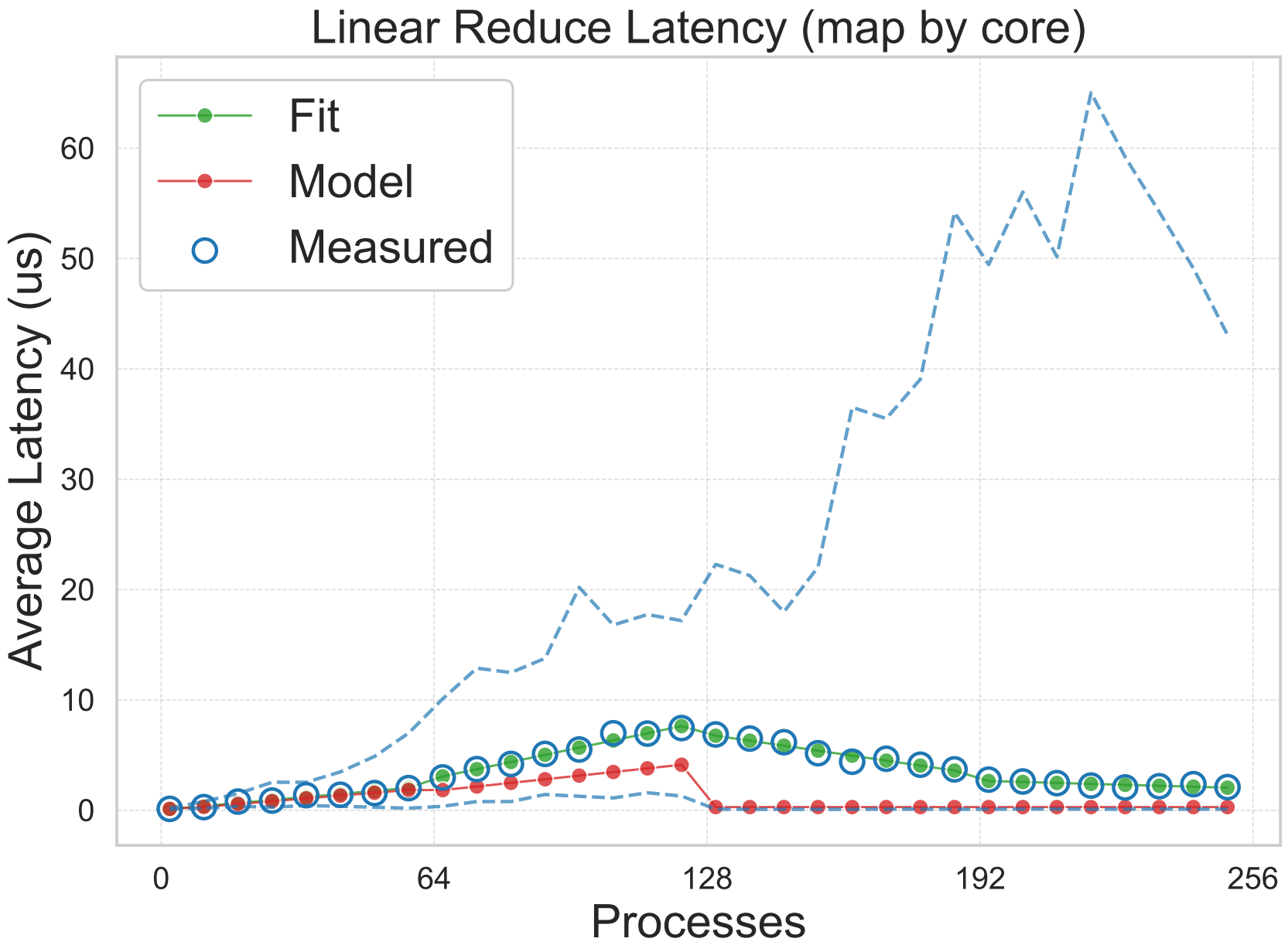

Binary Tree Broadcast Latency (map by socket)
Fit
Model
Measured
Average Latency (us)
0
10
20
30
40
0
64
128
192
256
Processes

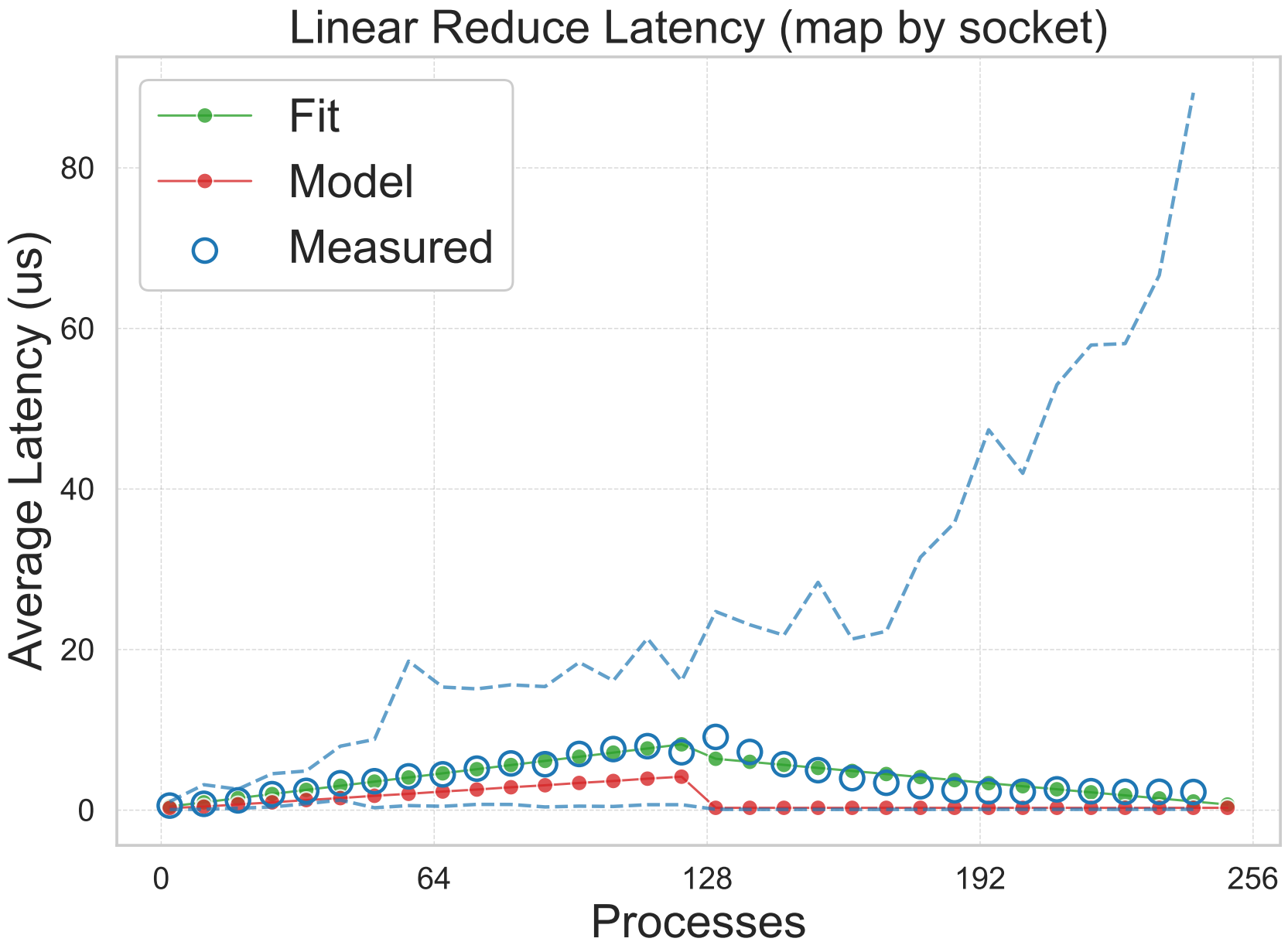

Binary Tree Broadcast Latency (map by node)
Fit
Model
Measured
Average Latency (us)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
64
128
192
256
Processes

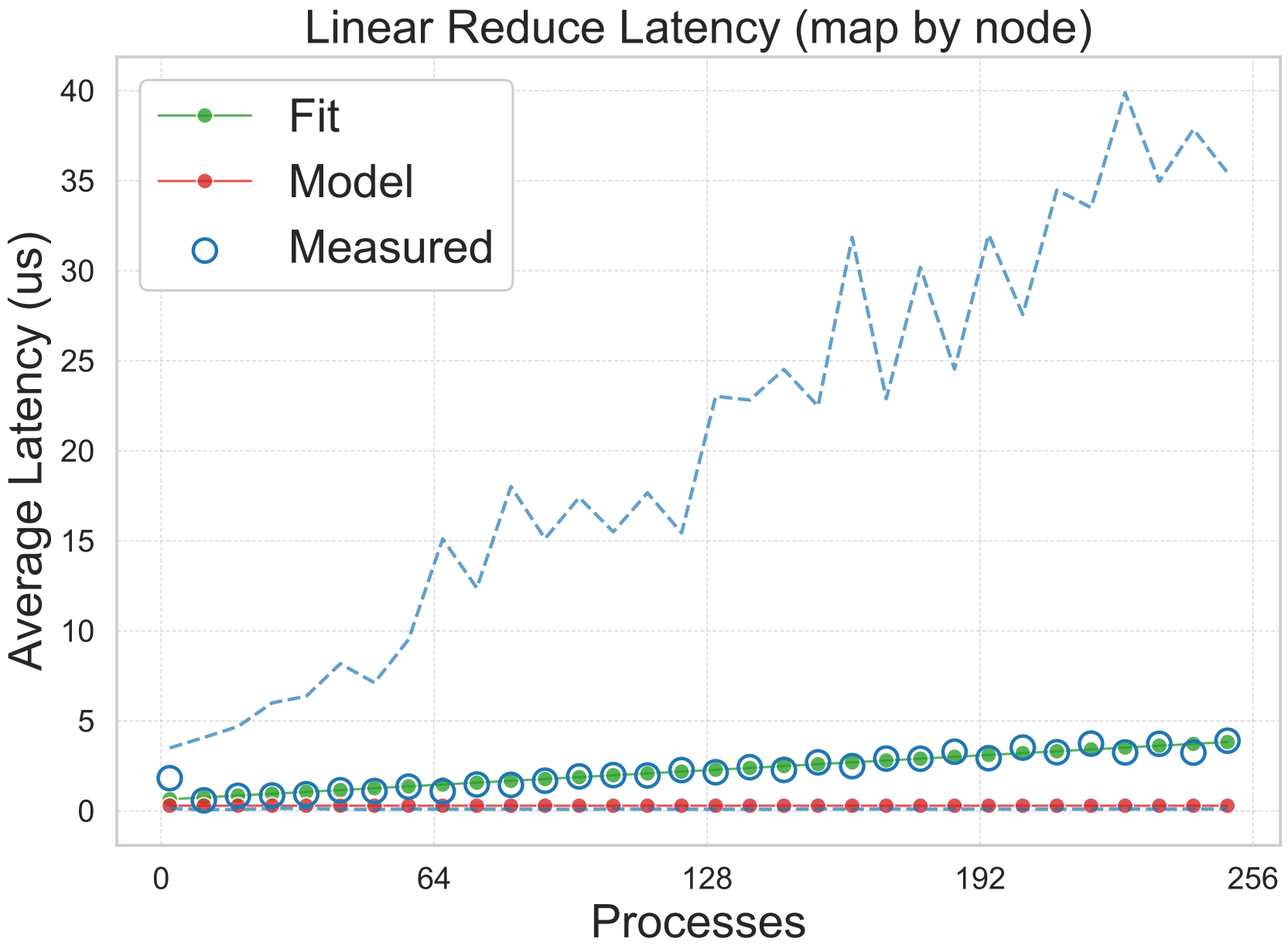

Default Reduce Latency (map by core)
Fit
Measured
Average Latency (us)
0
10
20
30
40
50
0
64
128
192
256
Processes

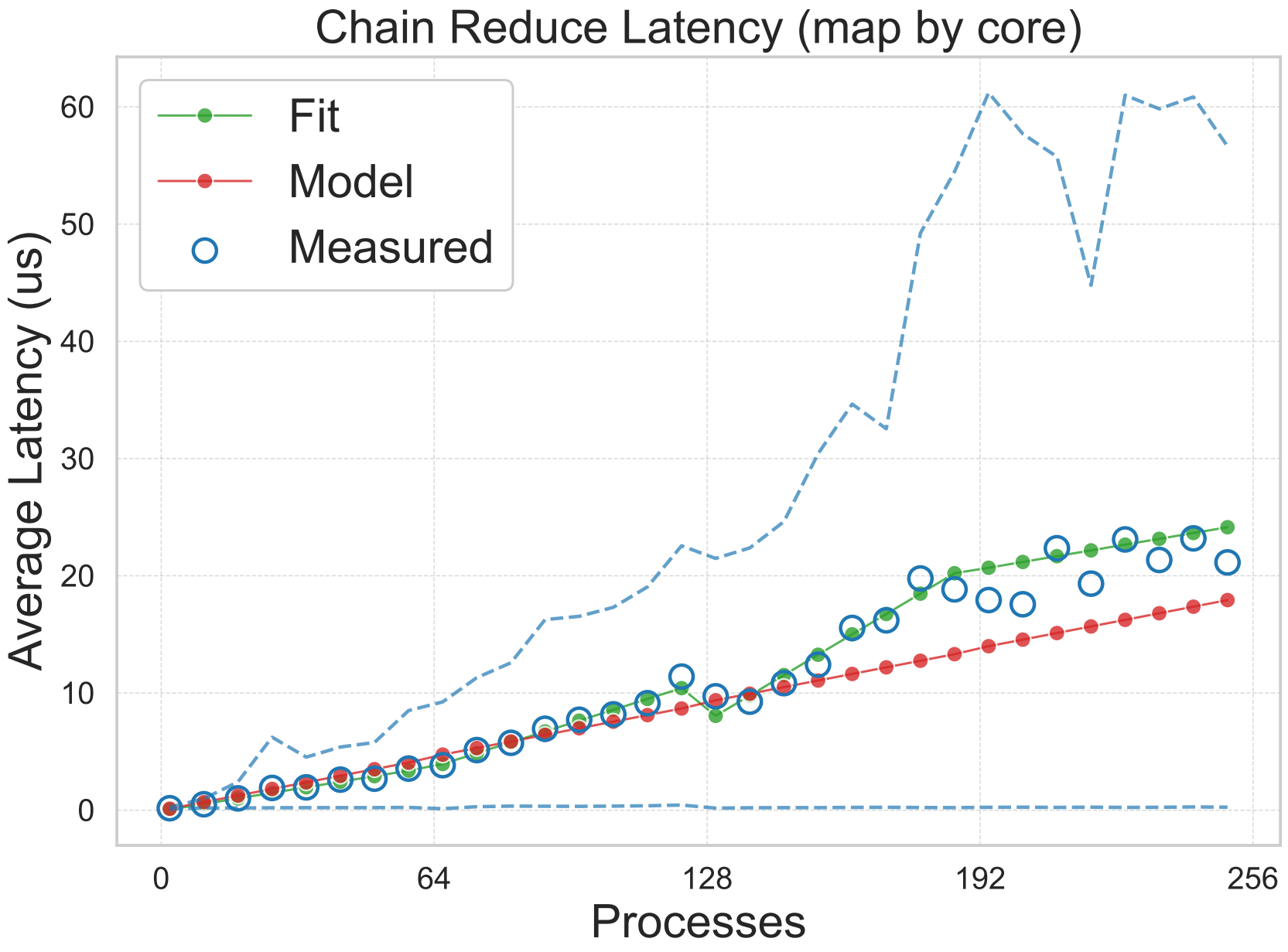

Default Reduce Latency (map by socket)
Fit
Measured
Average Latency (us)
0
10
20
30
40
50
60
70
0
64
128
192
256
Processes

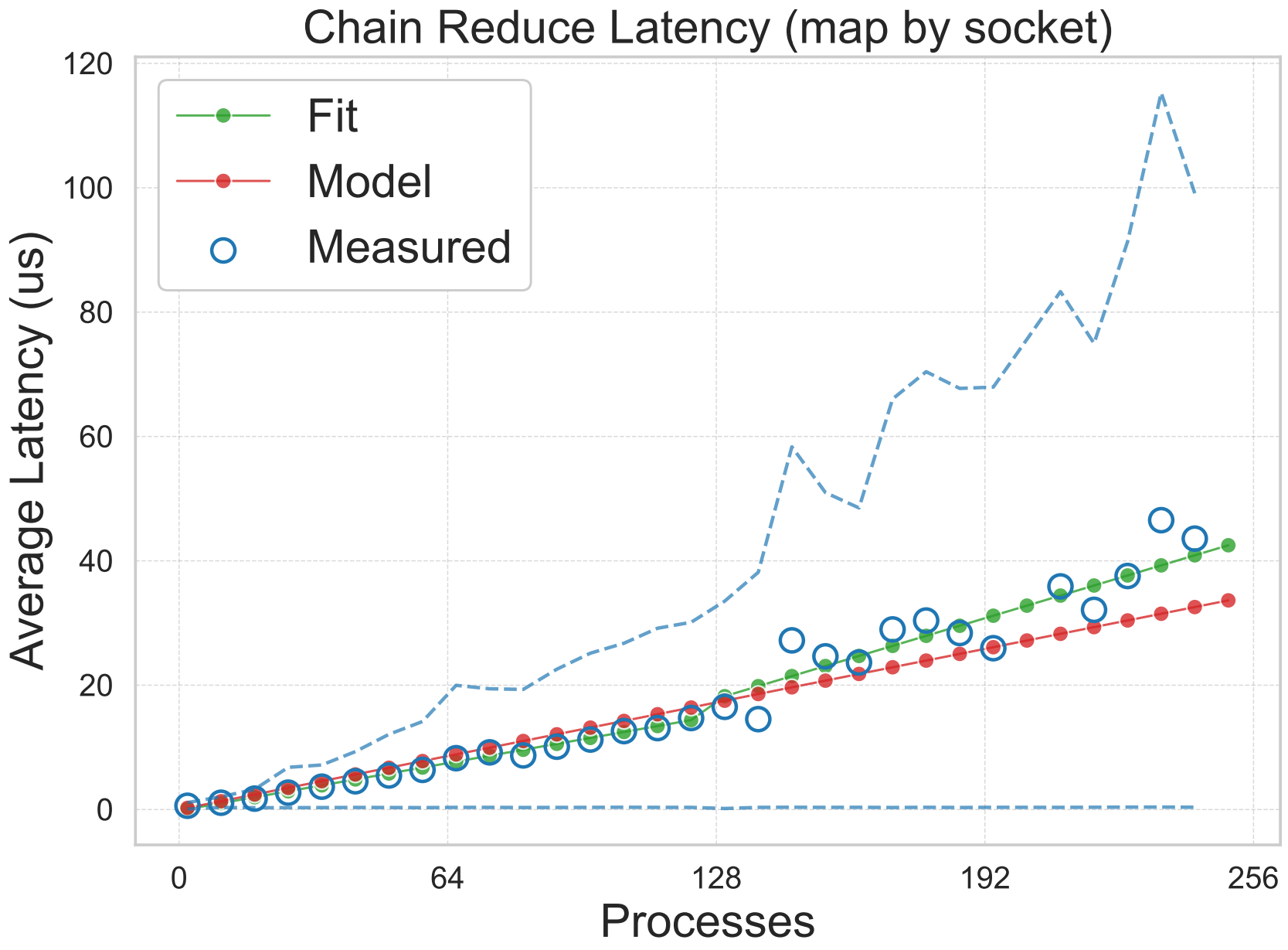

Default Reduce Latency (map by node)
Fit
Measured
Average Latency (us)
0
10
20
30
40
50
0
64
128
192
256
Processes

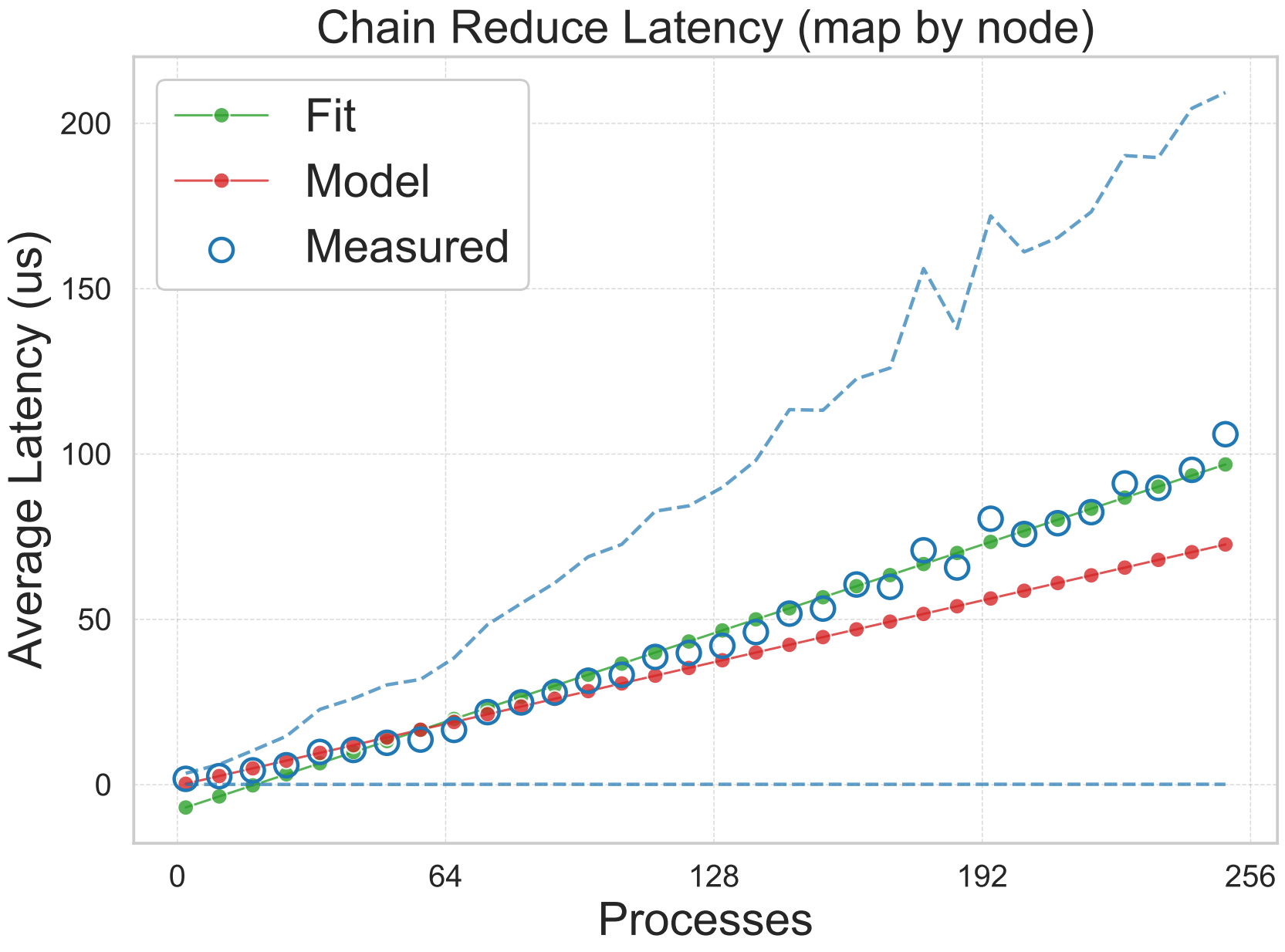

Linear Reduce Latency (map by core)
Fit
Model
Measured
Average Latency (us)
0
10
20
30
40
50
60
0
64
128
192
256
Processes

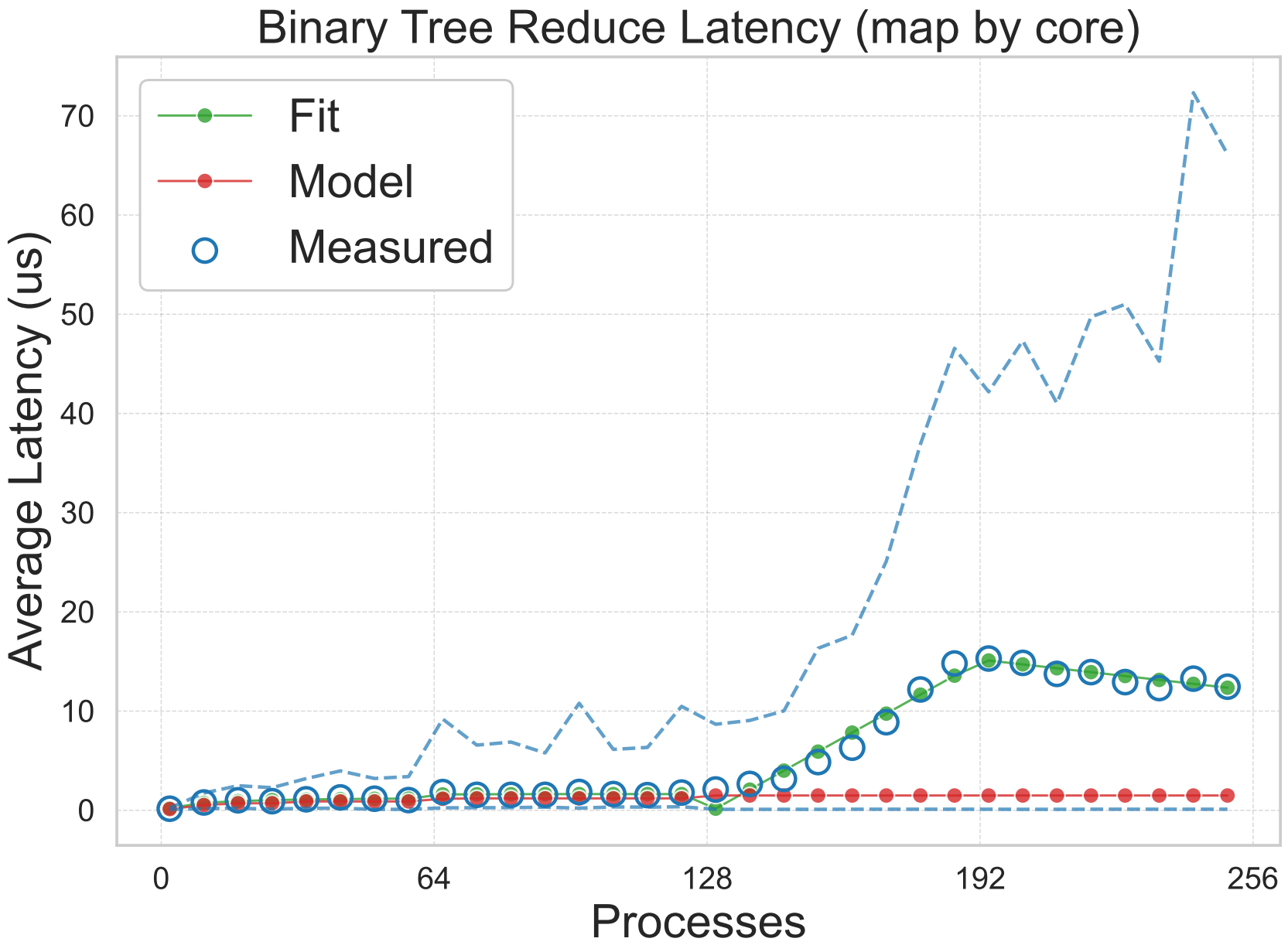

Linear Reduce Latency (map by socket)
Fit
Model
Measured
Average Latency (us)
0
20
40
60
80
0
64
128
192
256
Processes

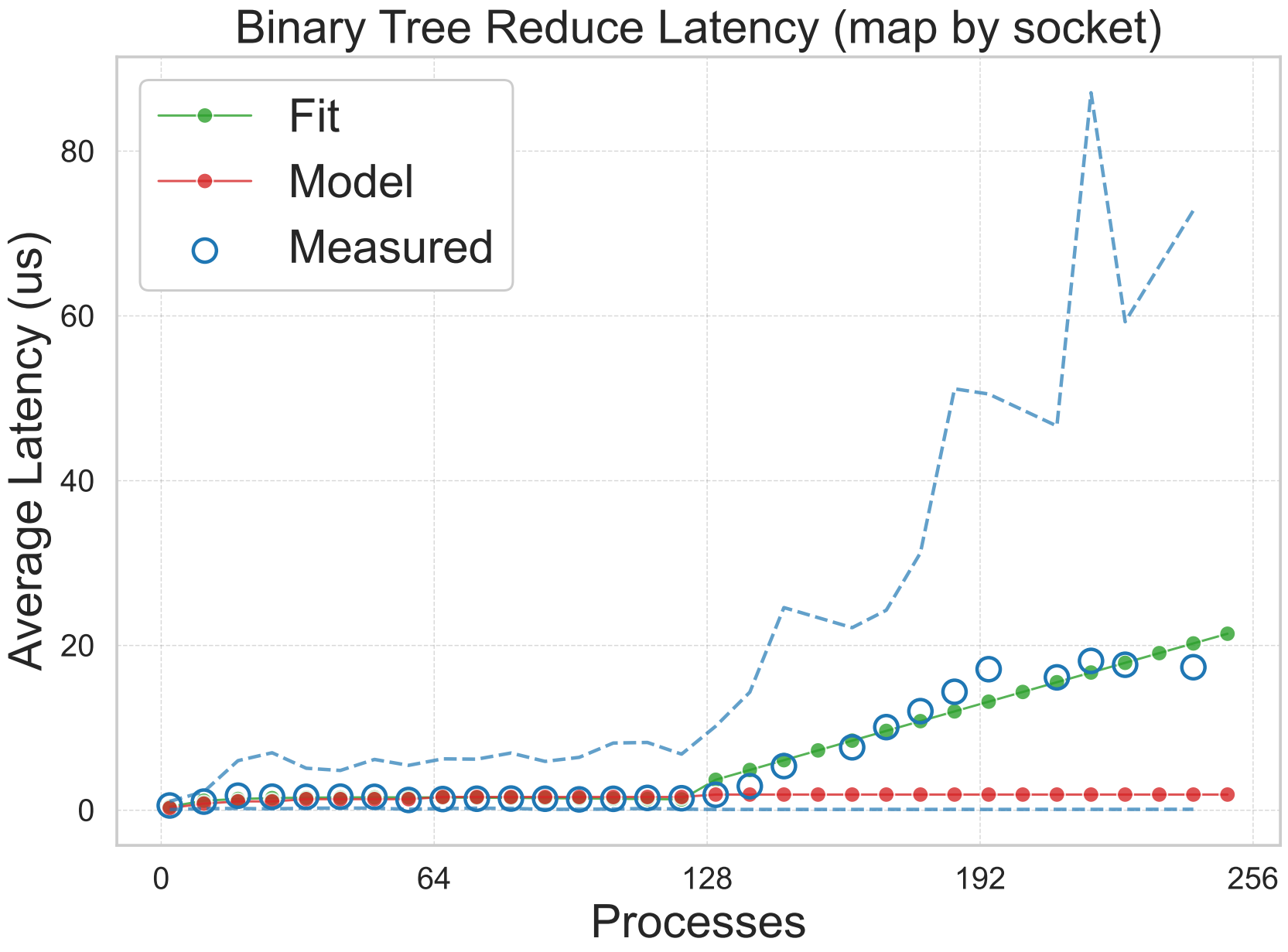

Linear Reduce Latency (map by node)
Fit
Model
Measured
Average Latency (us)
0
5
10
15
20
25
30
35
40
0
64
128
192
256
Processes

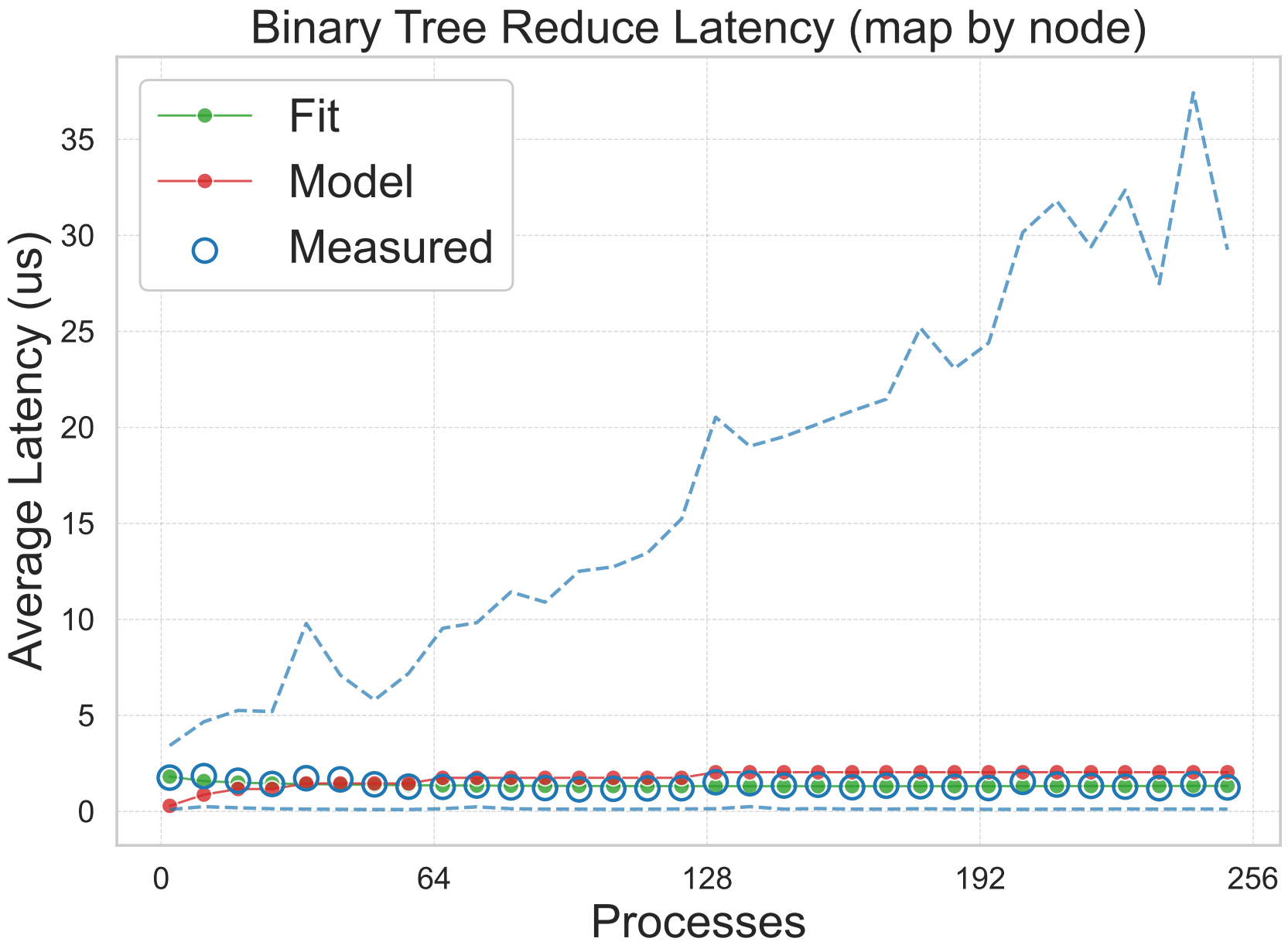

Chain Reduce Latency (map by core)
Fit
Model
Measured
Average Latency (us)
0
10
20
30
40
50
60
0
64
128
192
256
Processes


Chain Reduce Latency (map by socket)
Fit
Model
Measured
Average Latency (us)
120
100
80
60
40
20
0
0
64
128
192
256
Processes


Chain Reduce Latency (map by node)
Fit
Model
Measured
Average Latency (us)
200
150
100
50
0
0
64
128
192
256
Processes


Binary Tree Reduce Latency (map by core)
Fit
Model
Measured
Average Latency (us)
0
10
20
30
40
50
60
70
0
64
128
192
256
Processes


Binary Tree Reduce Latency (map by socket)
Fit
Model
Measured
Average Latency (us)
0
20
40
60
80
0
64
128
192
256
Processes


Binary Tree Reduce Latency (map by node)
Fit
Model
Measured
Average Latency (us)
0
5
10
15
20
25
30
35
0
64
128
192
256
Processes

References


AMD.
Epyc 7h12 specifications, 2023.


MPI Forum.
Mpi: A message-passing interface standard, 2023.


Roger W. Hockney.
The communication challenge for mpp: Intel paragon and meiko cs-2.
Parallel Computing, 20(3):389–398, 1994.


Emin Nuriyev, Juan-Antonio Rico-Gallego, and Alexey Lastovetsky.
Model-based selection of optimal mpi broadcast algorithms for multi-core clusters.
Journal of Parallel and Distributed Computing, 165:1–16, 2022.


AREA Science Park.
Orfeo documentation, 2023.


Markus Velten, Robert Schöne, Thomas Ilsche, and Daniel Hackenberg.
Memory performance of amd epyc rome and intel cascade lake sp server processors.