Project
Algorithm Engineering

Jens K. Mueller

jkm@informatik.uni-jena.de

Department of Mathematics and Computer Science
Friedrich-Schiller-University Jena

Wednesday 28th May, 2014

Profiling

Compiler Flags

- dmd
  `-unittest -noboundscheck -inline -O`
- gdc
  `-funittest -fno-bounds-check -O3 -ffast-math -funroll-loops`
- ldc
  `-unittest -disable-boundscheck -inline -mcpu=native -O3`

Larger Applications

- Profile on real(istic) input
  - Make common case fast
  - Focus on innermost loops
- Optimize identified hot spot(s)

Profiling

Resources

- CPU
- Memory

Tools

- Instrumentation
  gcc's -pg together with gprof
- Statistical
  google-perftools
- Trapping on predefined events
  Java, .NET, ...

Optimizing Hot Spots

1. Algorithms and data structures
2. Basic optimizations
   - Move computation out of loop
   - Eliminate unnecessary memory references
3. Low-level optimizations
   - Unroll loops
   - Increase instruction level parallelism (eliminate data dependencies)
   - Use conditional moves

Amdahl's Law

Fraction x of program optimized by a factor s, then

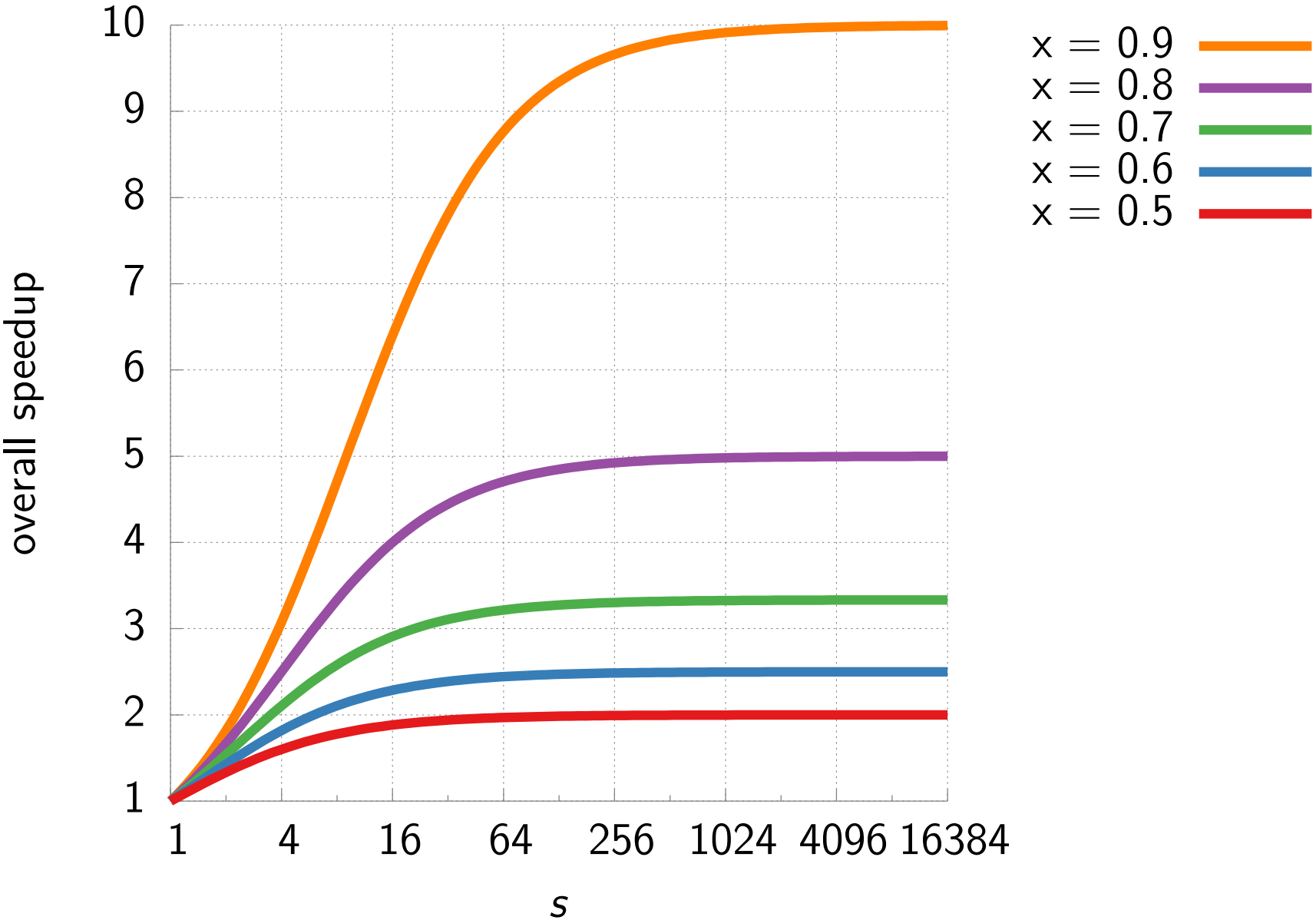
$$T_s = \underbrace{(1-x)}_{\text{unoptimized}} + \underbrace{\frac{x}{s}}_{\text{optimized}}$$

Assume $T = 1$ (unoptimized program's running time), then overall speedup is

$$\frac{T}{T_s} = \frac{1}{(1-x) + \frac{x}{s}}$$

- $x = 1$ yields overall speed up of s
- Strong scaling
  Overall speed up for fixed input size

Amdahl's Law (cont.)



Gustafson's Law

Embarrassingly parallel fraction x of program, then

$$T = (1 - x) + p \cdot x$$

Parallel running time is

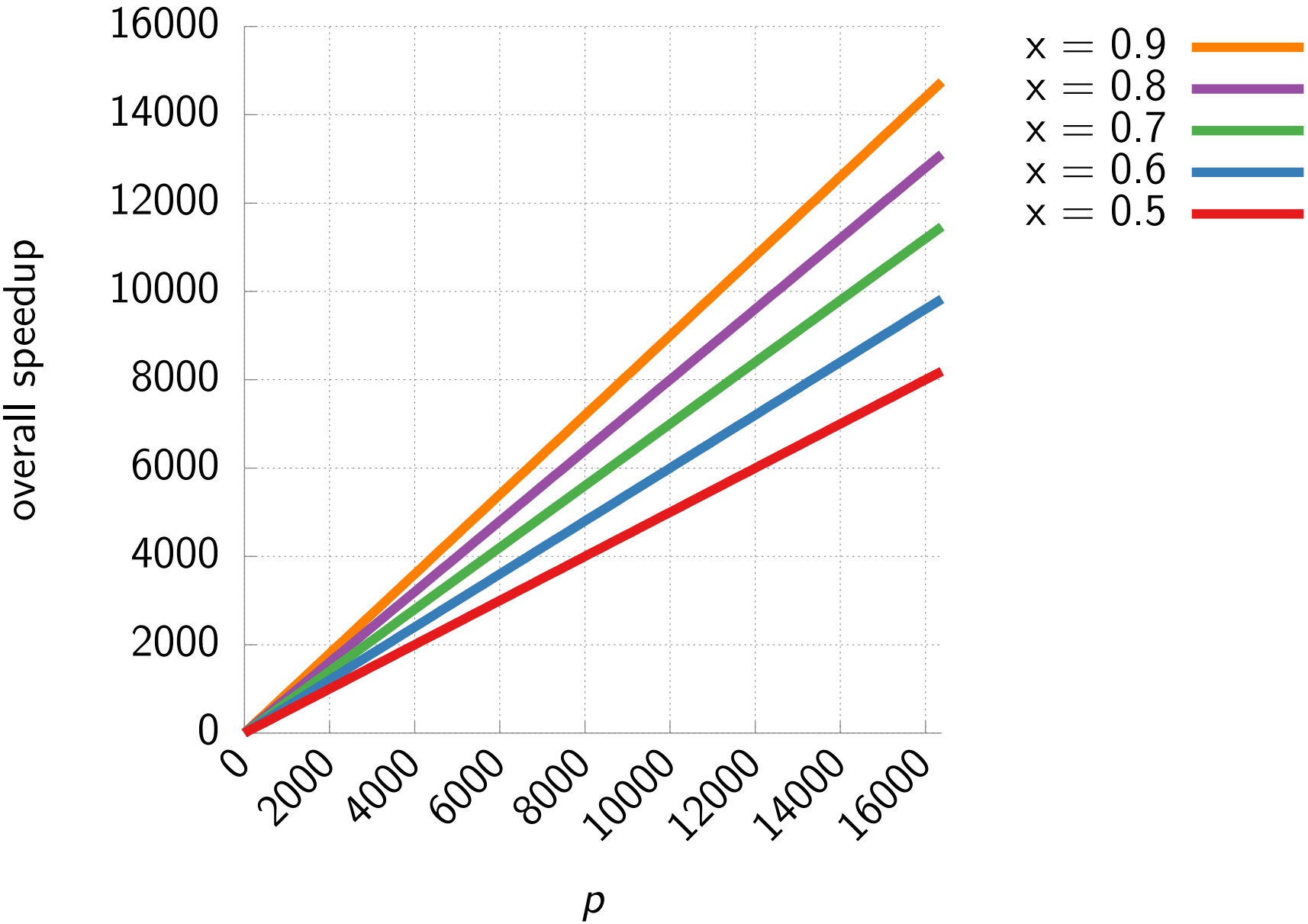
$$T_p = (1 - x) + x = 1$$

Hence, speed up is

$$\frac{T}{T_p} = (1 - x) + p \cdot x = 1 + (p - 1) \cdot x$$

- Parallel part grows linear with the number of processors
- Input sizes increase
- Weak scaling
  Overall speed up for increasing input size

Gustafson's Law (cont.)



Theoretical and Effective Performance

- Theoretical Performance
  - Number of nodes: 1
  - Number of CPUs: 1
  - Number of Cores: 1
  - CPU frequency: 2.66 GHz
  - Number of operations per cycle: 4 FLOP/cycle (single precision)

  Theoretical (peak) performance

$$2.66\,\text{GHz} \cdot 4\,\text{FLOP/cycle} \approx 10.6\,\text{GFLOP/s}$$

- Effective performance determines the number of operations per time

Theoretical and Effective Bandwidth

- Theoretical bandwidth
  - Memory clock: 1066 MHz
  - Type of memory: DDR2 (double data rate)
  - Number of channels: 2
  - Memory bus: 64 bit

  $$1066\,\text{MHz} \cdot 2 \cdot 2 \cdot 64\,\text{bit}/8 = 34\,112\,\text{B/s} \approx 34.1\,\text{GB/s}$$

- Effective bandwidth

  $$(B_r + B_w)/t,$$

  where B_r is the number of bytes read, B_w number of bytes written and t the spent time.

System Information

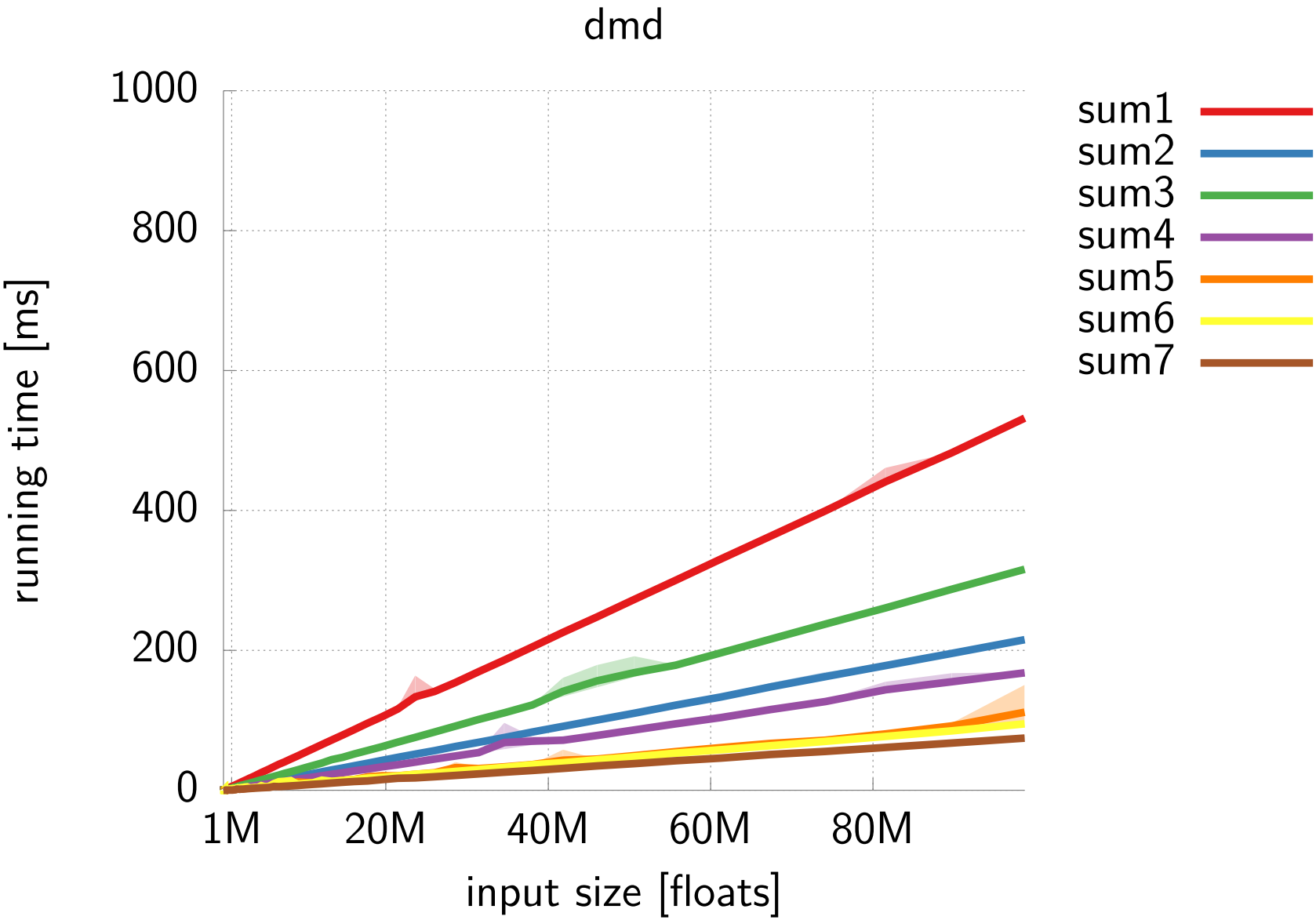
- CPU

  ```
  $ lshw -C cpu
  ```

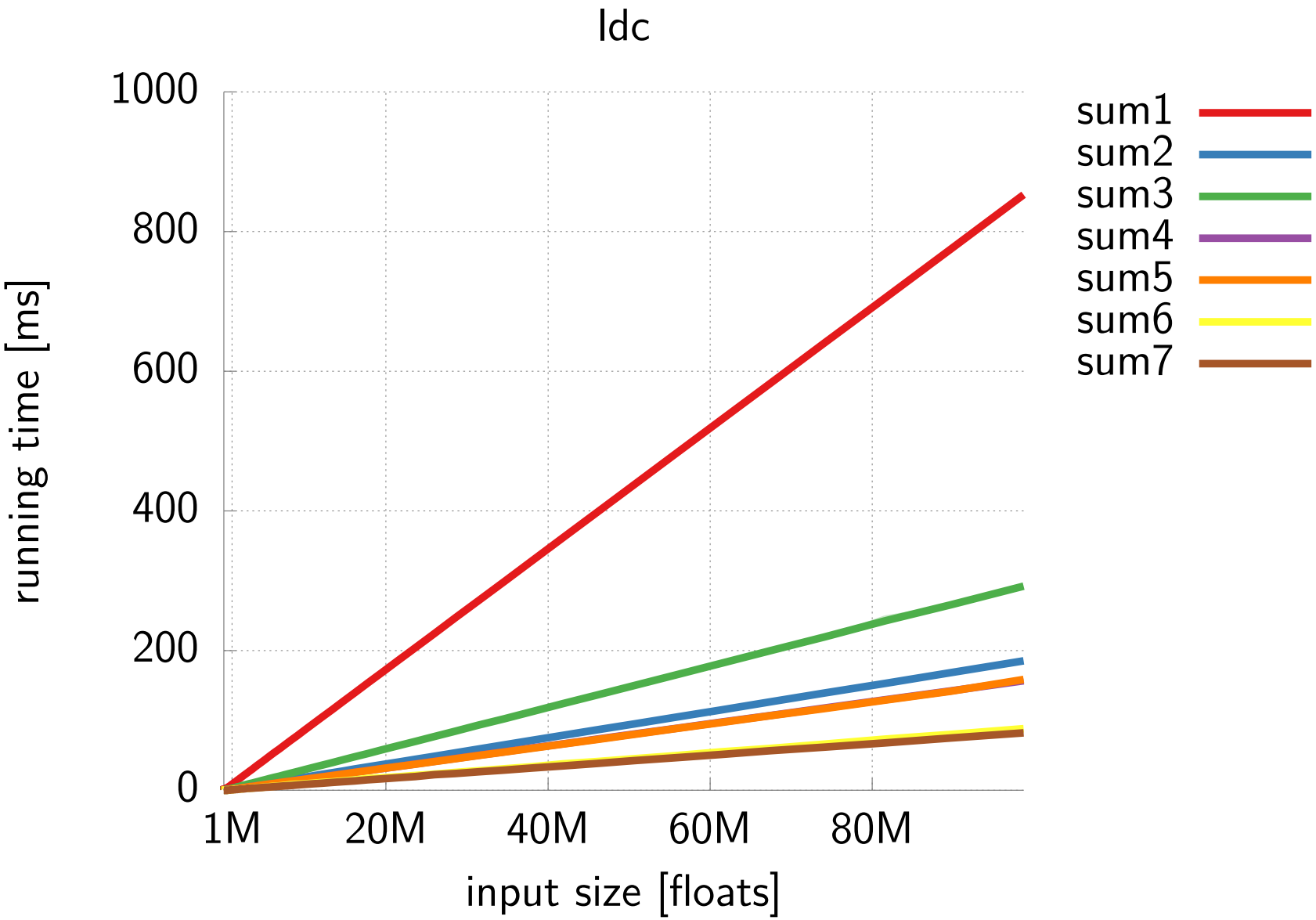
- Memory

  ```
  $ lshw -short -C memory
  ```

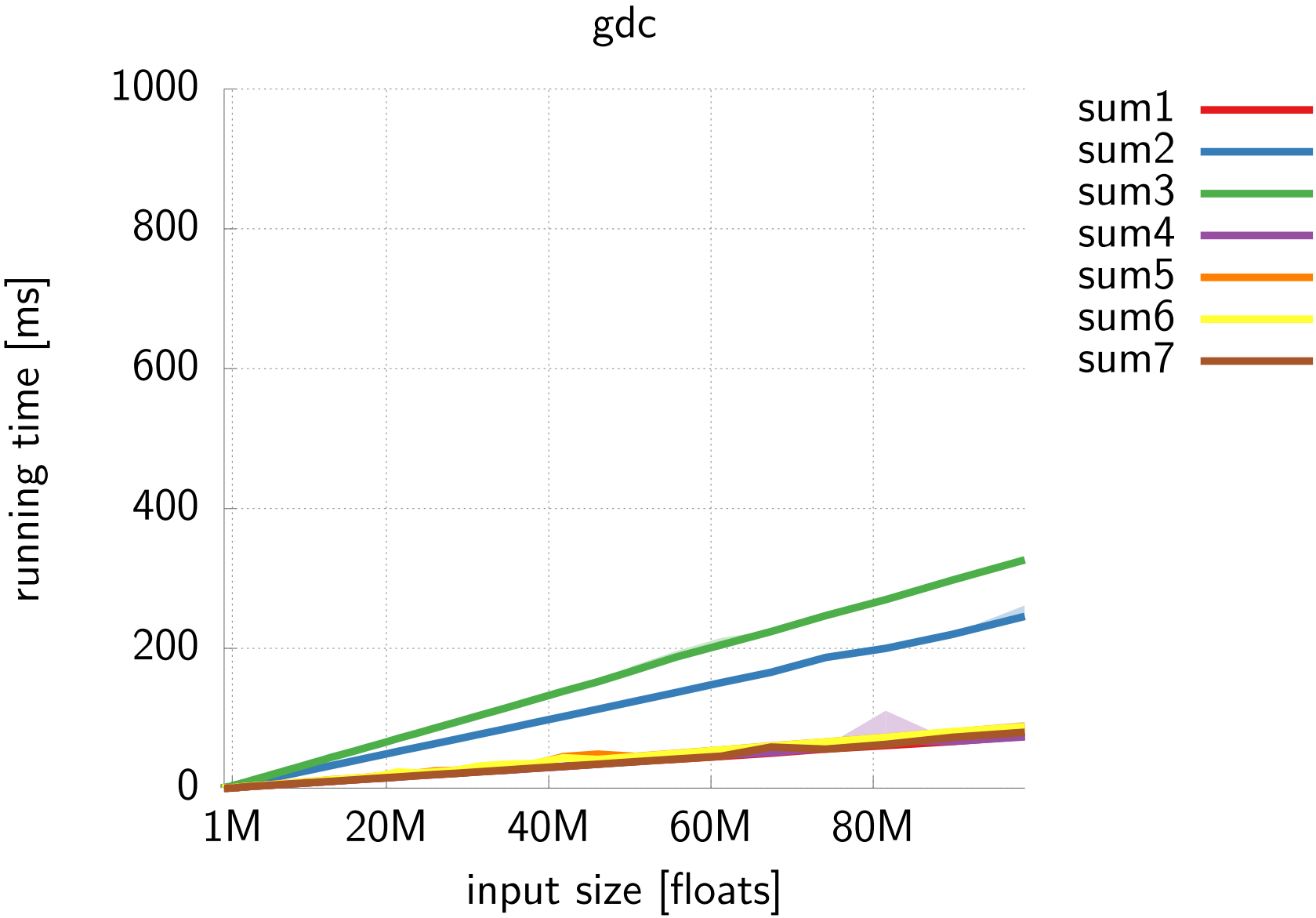
Running Time



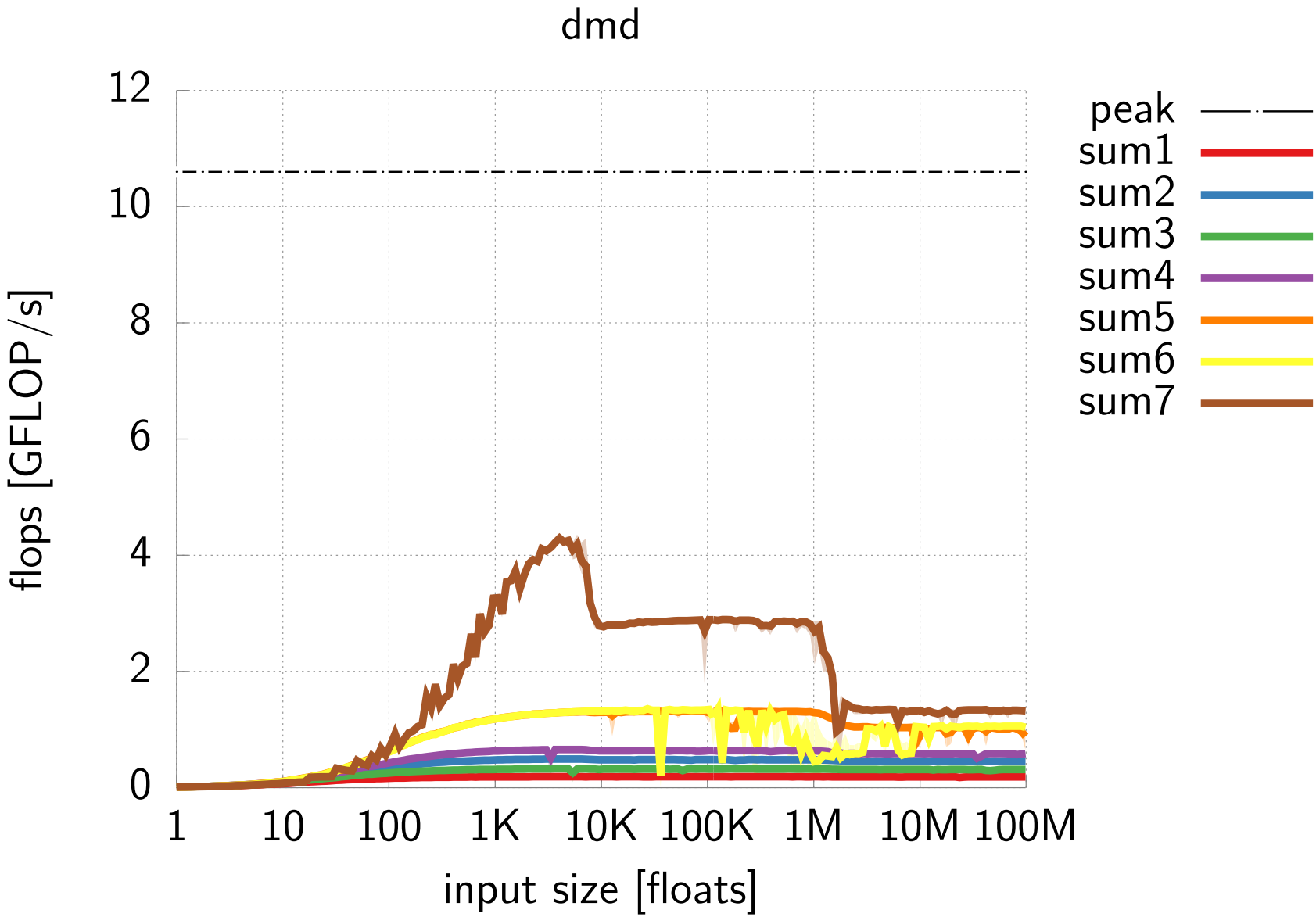
Running Time (cont.)



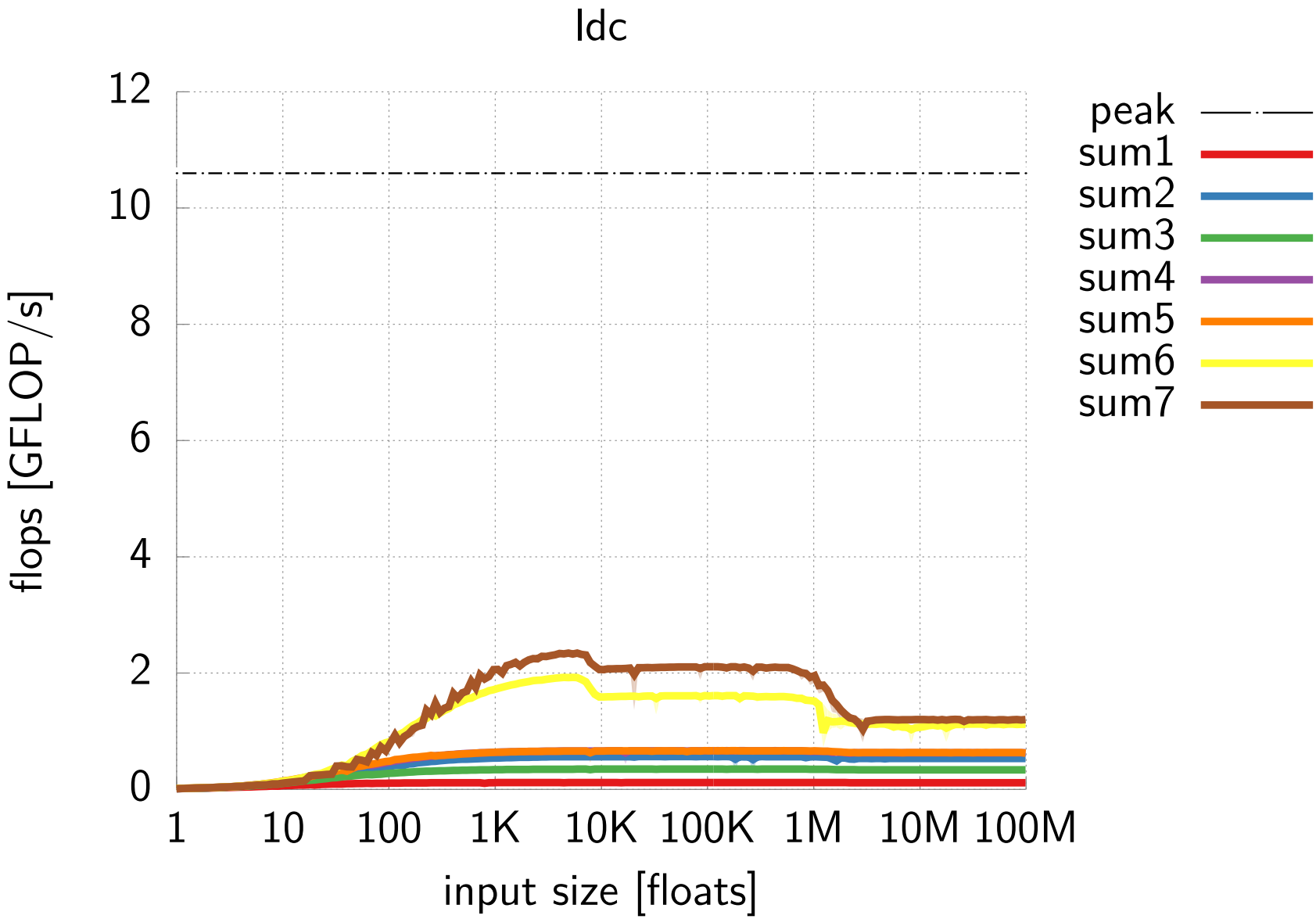
Running Time (cont.)



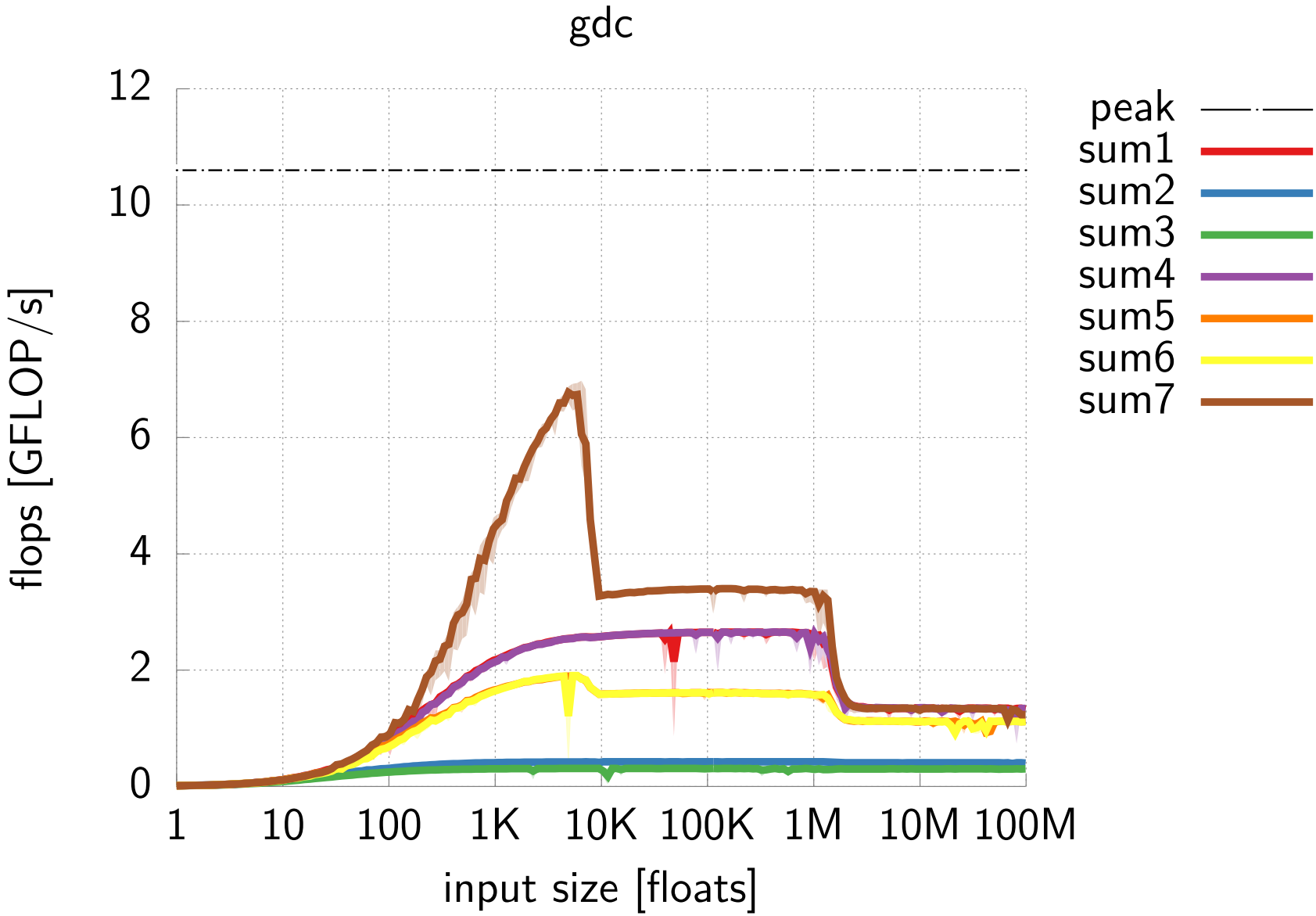
FLOPS


dmd
flops [GFLOP/s]
0
2
4
6
8
10
12
1
10
100
1K
10K
100K
1M
10M
100M
input size [floats]
peak
sum1
sum2
sum3
sum4
sum5
sum6
sum7

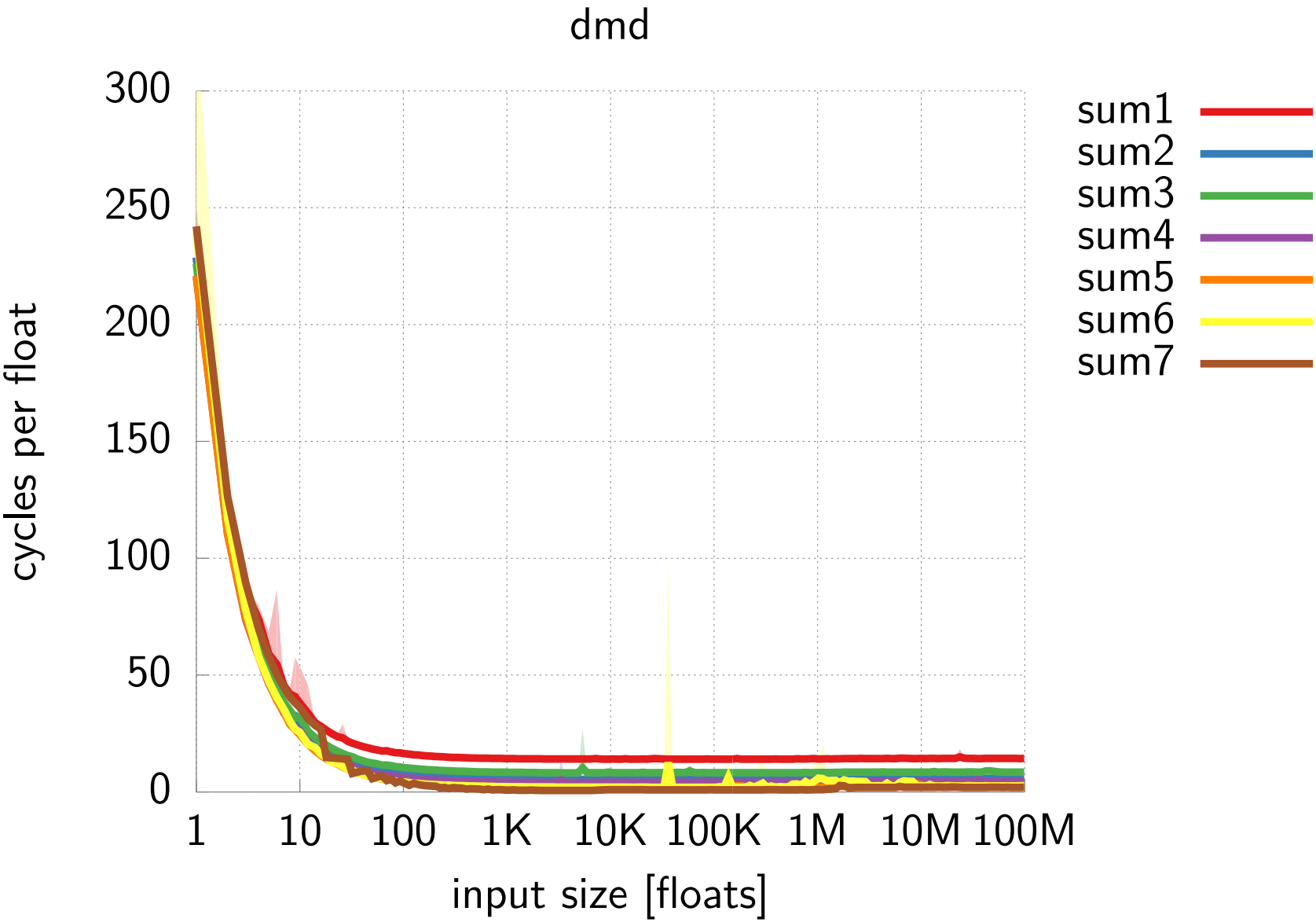
FLOPS (cont.)


ldc
flops [GFLOP/s]
0
2
4
6
8
10
12
1
10
100
1K
10K
100K
1M
10M
100M
input size [floats]
peak
sum1
sum2
sum3
sum4
sum5
sum6
sum7

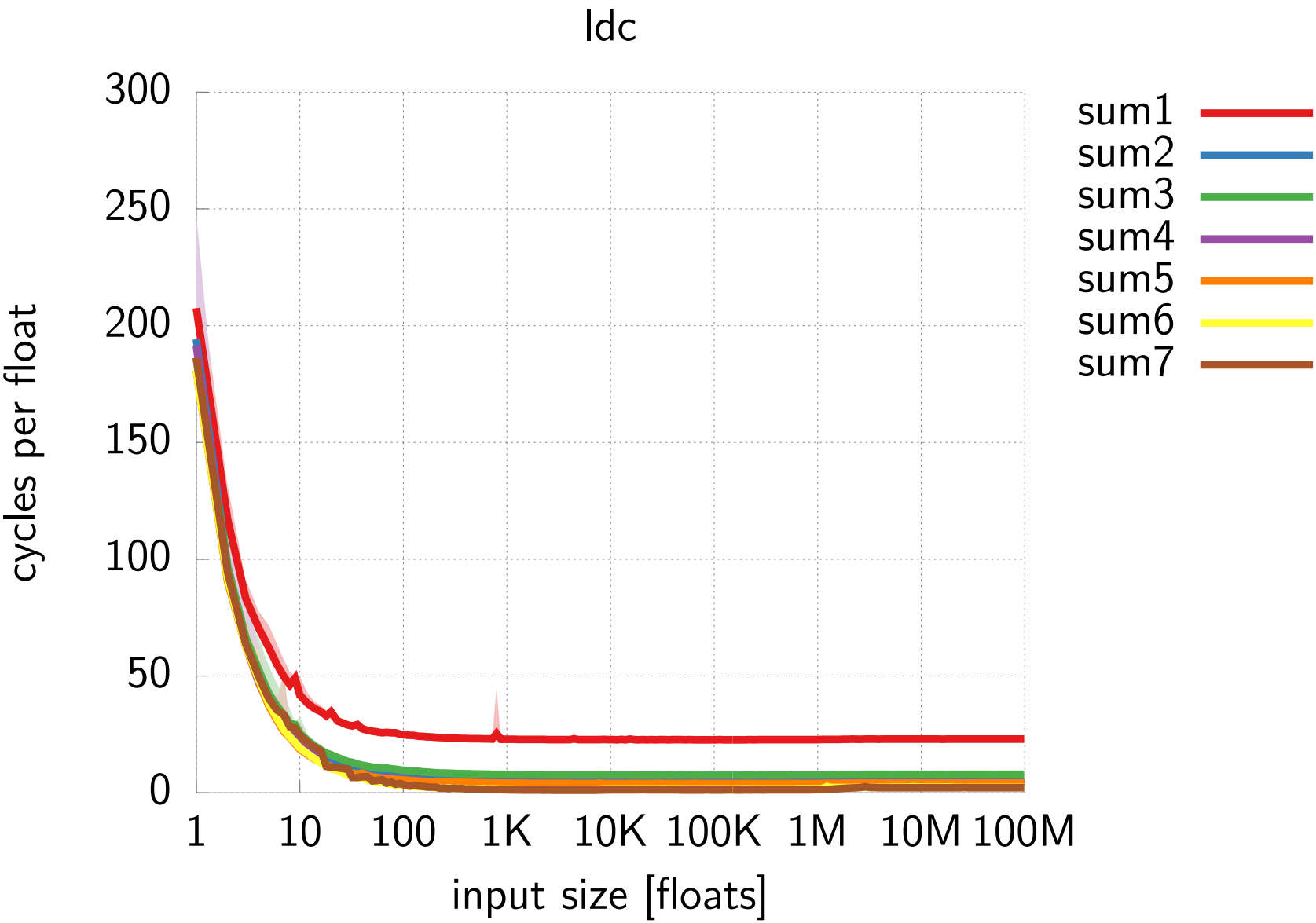
FLOPS (cont.)


gdc
flops [GFLOP/s]
0
2
4
6
8
10
12
1
10
100
1K
10K
100K
1M
10M
100M
input size [floats]
peak
sum1
sum2
sum3
sum4
sum5
sum6
sum7

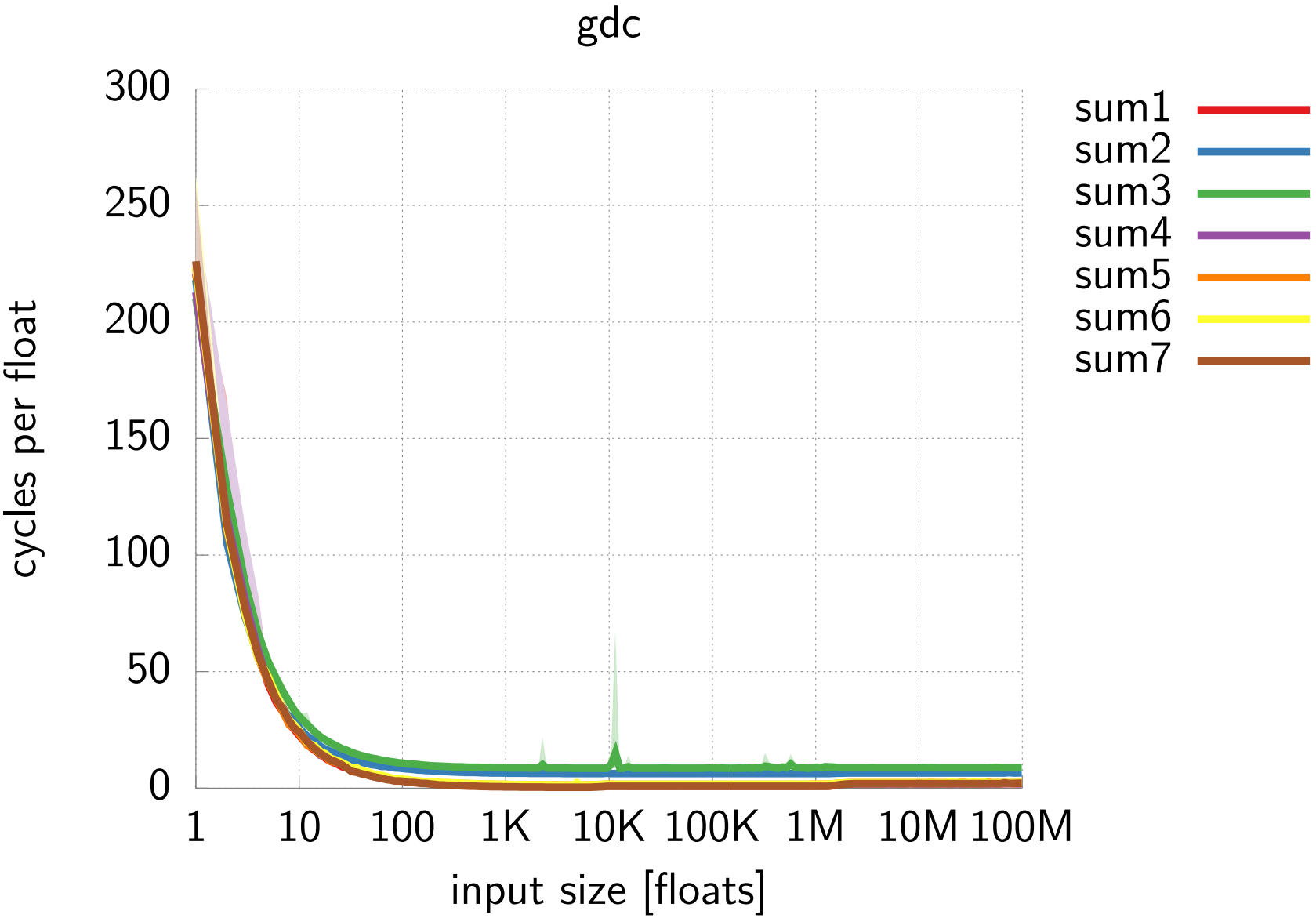
Cycles per Element



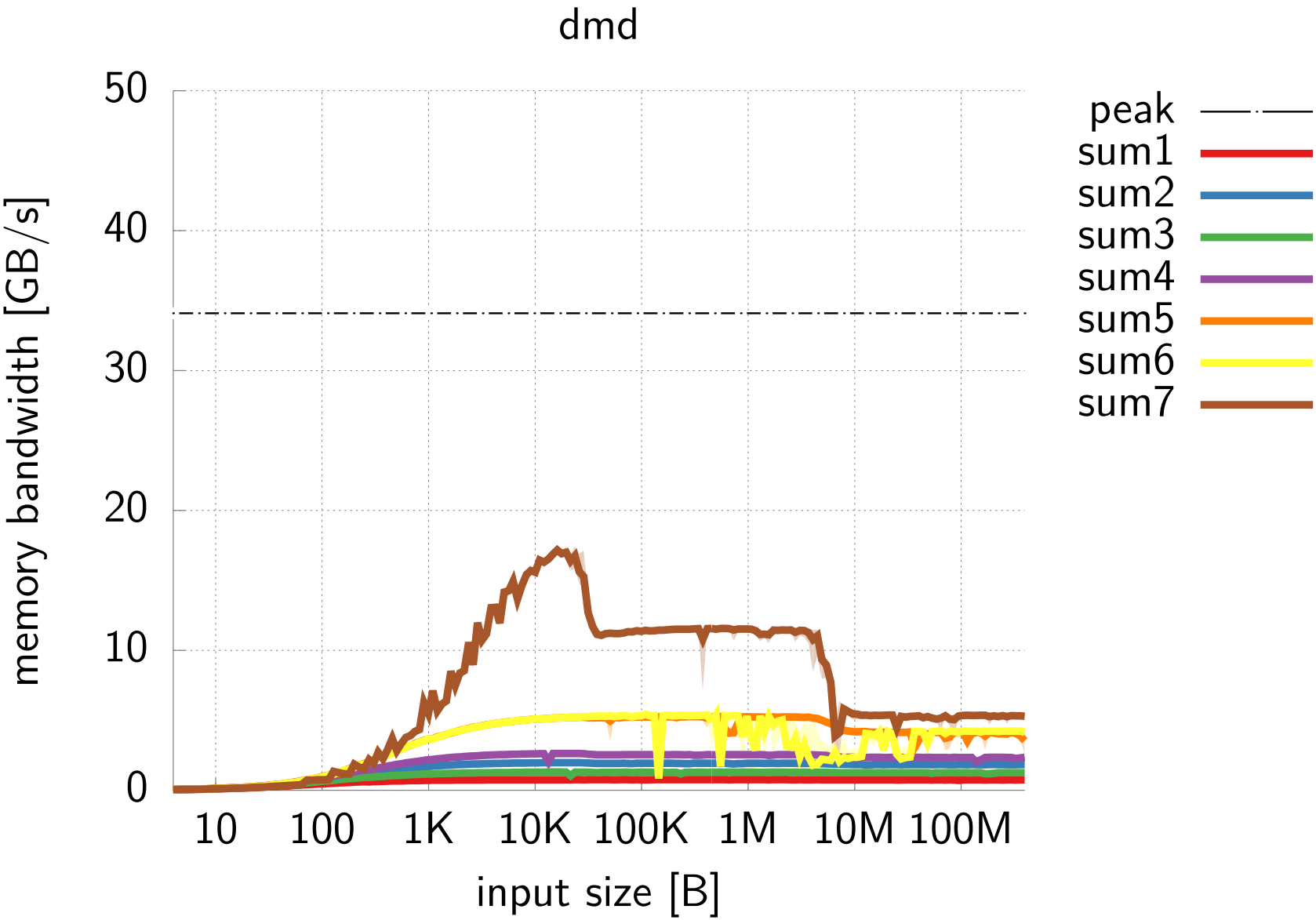
Cycles per Element (cont.)



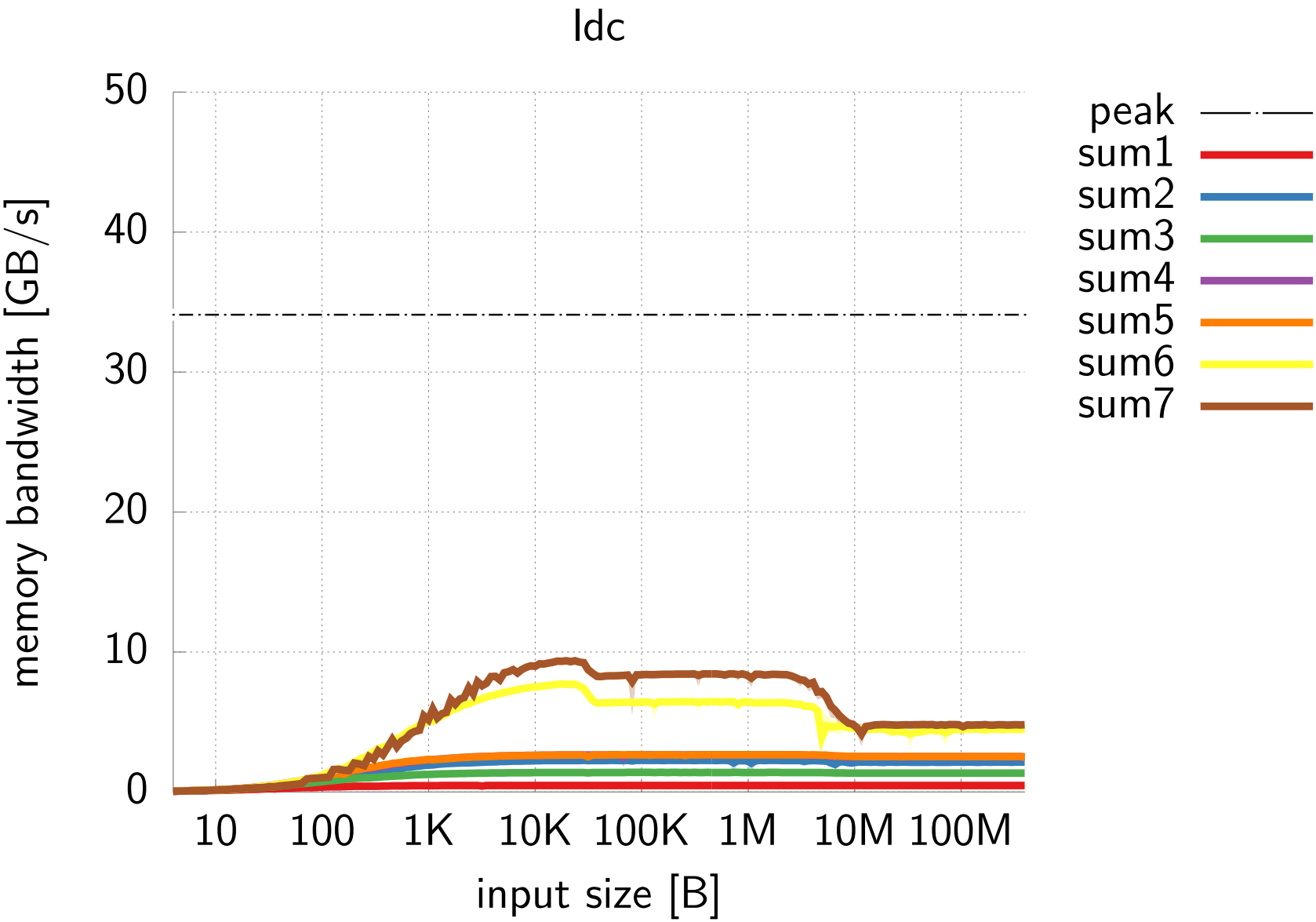
Cycles per Element (cont.)



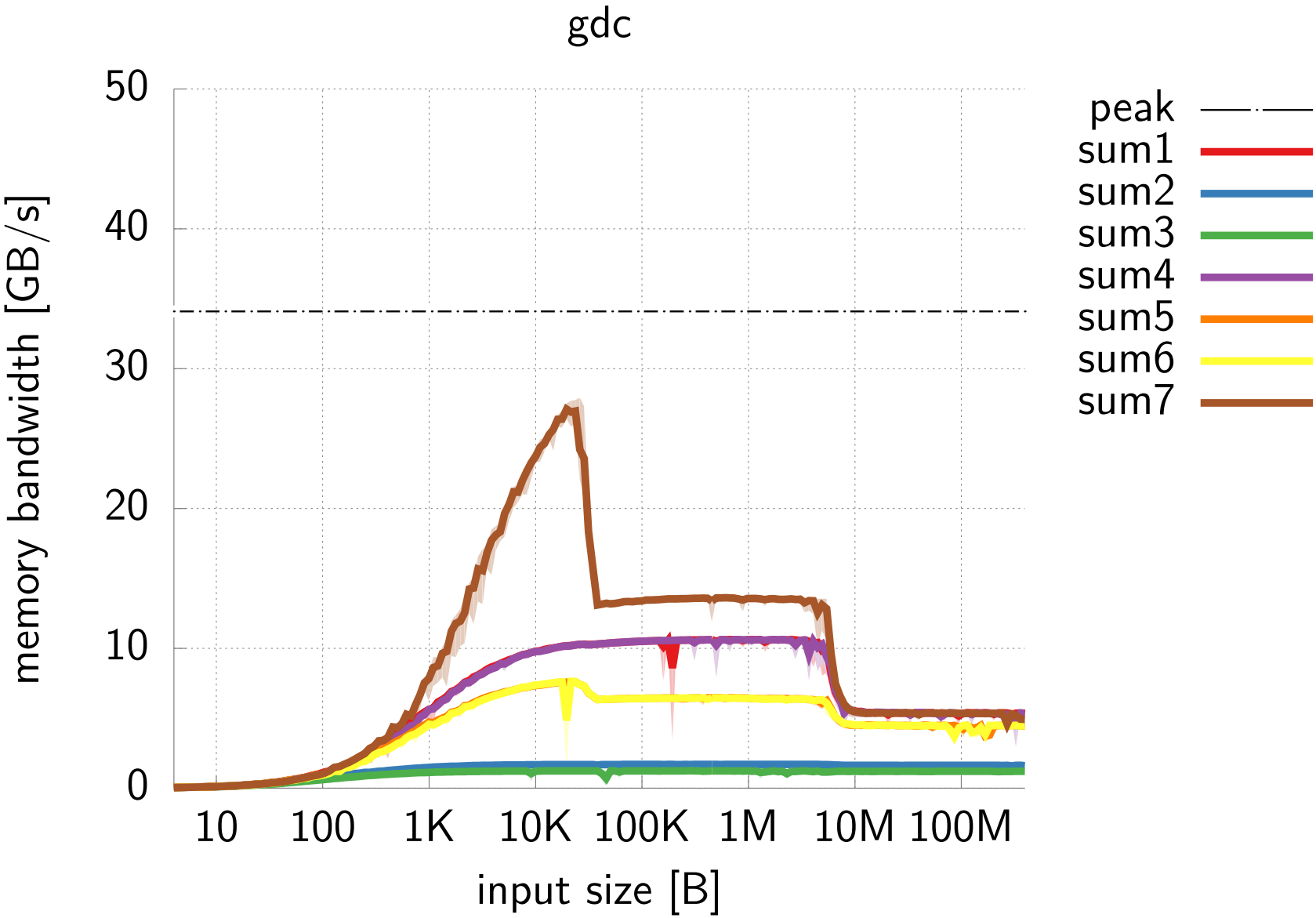
Memory Bandwidth



Memory Bandwidth (cont.)



Memory Bandwidth (cont.)



Homework

- Send your first project per email
- Choose second project (focuses on algorithms and data structures)