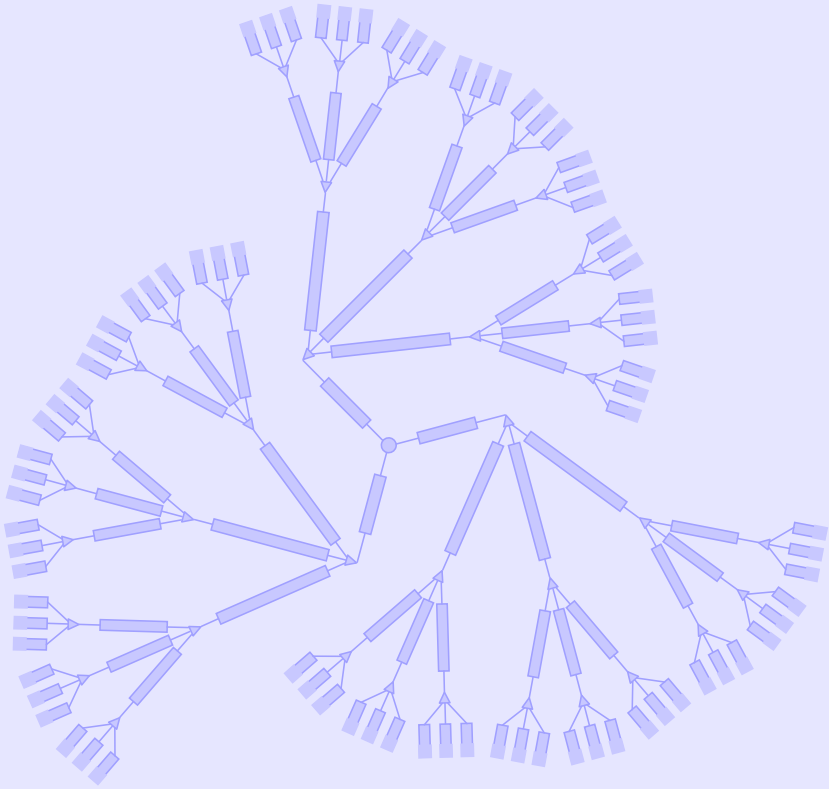
Engineering a Cache-Oblivious Sorting Algorithm



Gerth Stølting Brodal
University of Aarhus

Rolf Fagerberg
University of Aarhus

Kristoffer Vinther
Systematic Software Engineering

Motivation

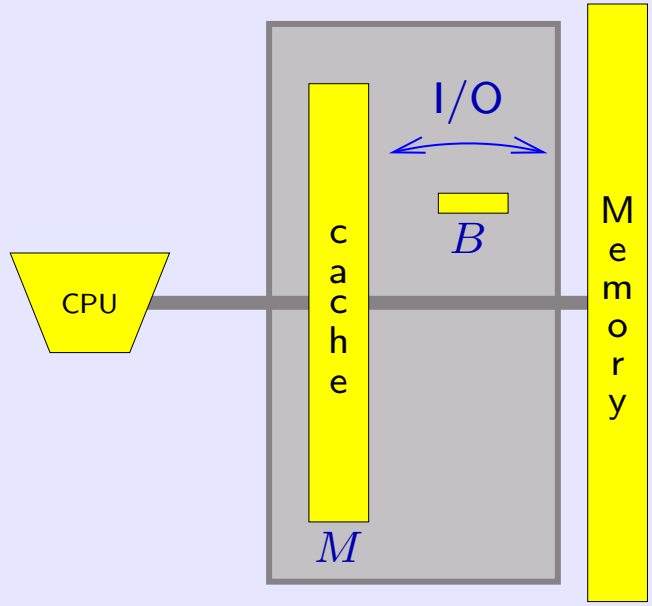
- Memory hierarchy has become a fact of life
- Accessing non-local storage may take a very long time
- Good locality is important to achieving high performance

	Latency	Relative to CPU
Register	0.5 ns	1
L1 cache	0.5 ns	1-2
L2 cache	3 ns	2-7
DRAM	150 ns	80-200
TLB	500+ ns	200-2000
Disk	10 ms	10^7

Cache-Oblivious Model

Frigo, Leiserson, Prokop, Ramachandran, FOCS'99

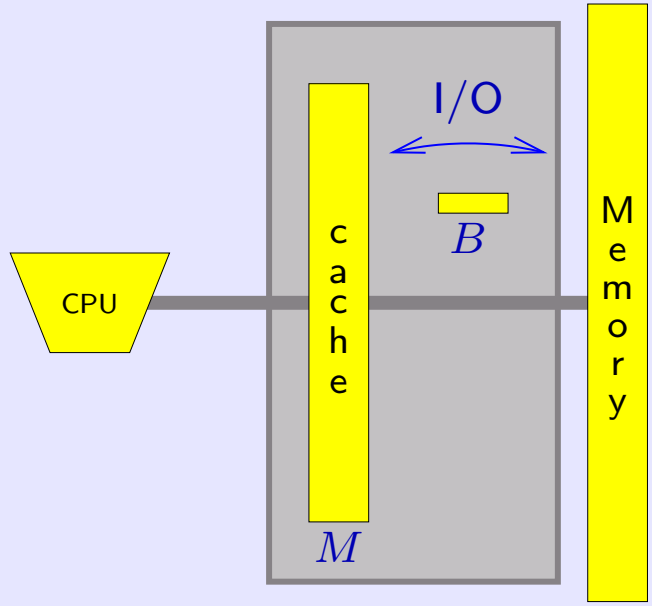
- Program in the RAM model
- Analyze in the I/O model for arbitrary B and M
- Optimal off-line cache replacement strategy



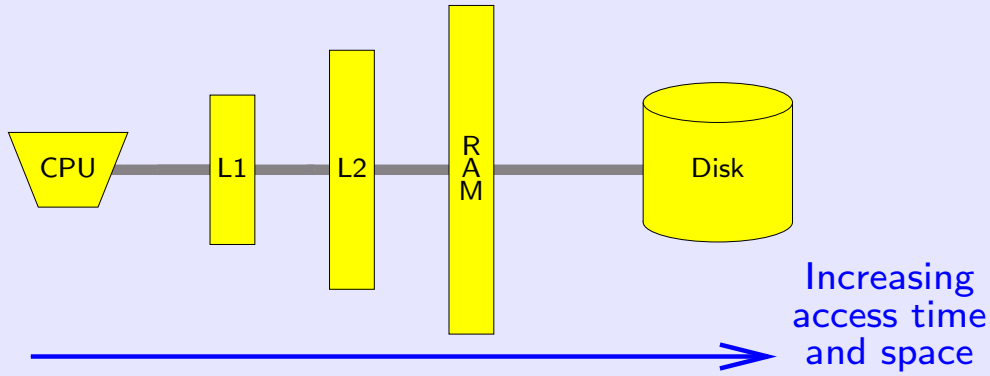
Cache-Oblivious Model

Frigo, Leiserson, Prokop, Ramachandran, FOCS'99

- Program in the RAM model
- Analyze in the I/O model for arbitrary B and M
- Optimal off-line cache replacement strategy



- Optimal on arbitrary level $\Rightarrow$ optimal on all levels
- Portability



Sorting — I/O Bounds

QuickSort ⋆ Binary MergeSort ⋆	$O\left(\frac{N}{B} \cdot \log_2 \frac{N}{M}\right)$
$\Theta\left(\frac{M}{B}\right)$-way MergeSort ⋆	$O\left(\mathrm{Sort}_{M,B}(N)\right)$ Aggarwal and Vitter 1988
(Lazy) Funnelsort ⋆ ⋆	$O\left(\mathrm{Sort}_{M,B}(N)\right)$ Frigo, Leiserson, Prokop and Ramachandran 1999 Brodal and Fagerberg 2002

⋆ cache-aware ⋆ cache-oblivious ⋆ requires $M \geq B^{1+\varepsilon}$

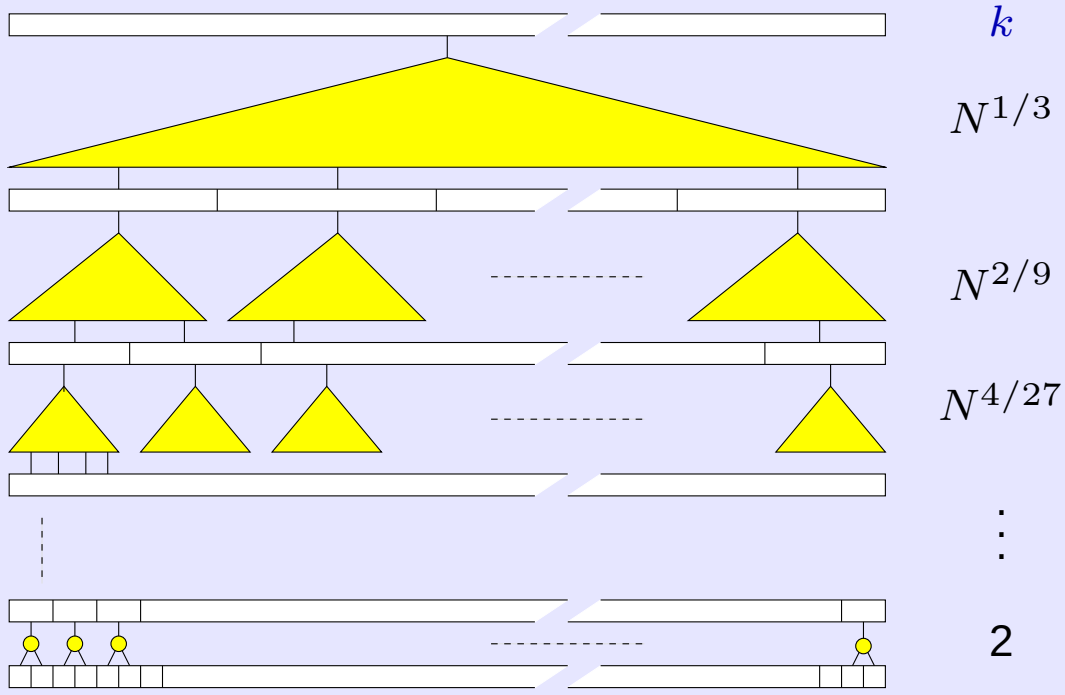
$$\mathrm{Sort}_{M,B}(N) = \frac{N}{B} \cdot \log_{M/B} \frac{N}{B}$$

Lazy Funnelsort

Divide input in $N^{1/3}$ segments of size $N^{2/3}$

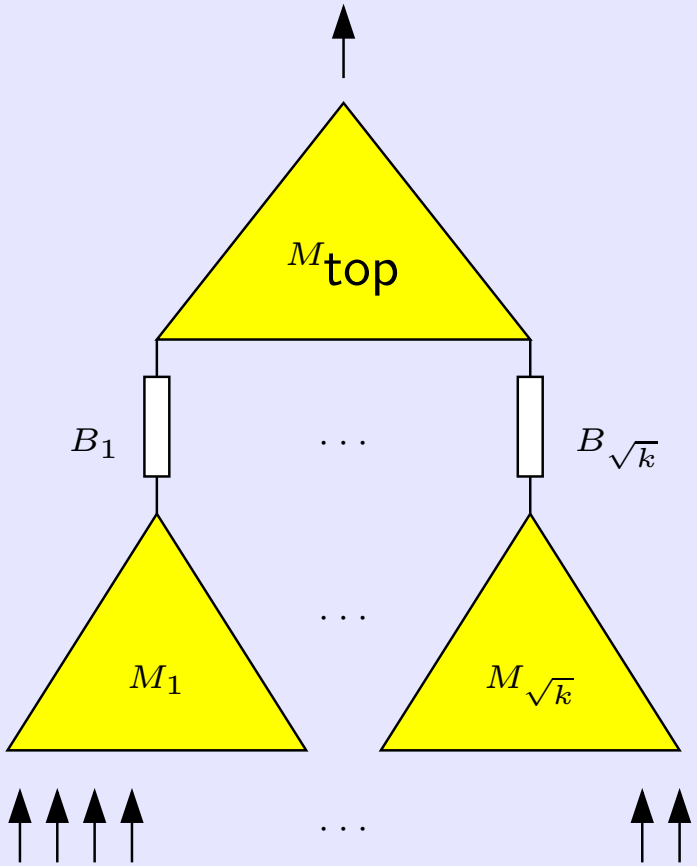
Recursively **Funnelsort** each segment

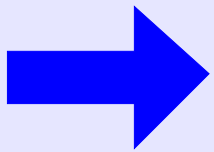
Merge sorted segments by an **lazy $N^{1/3}$-merger**

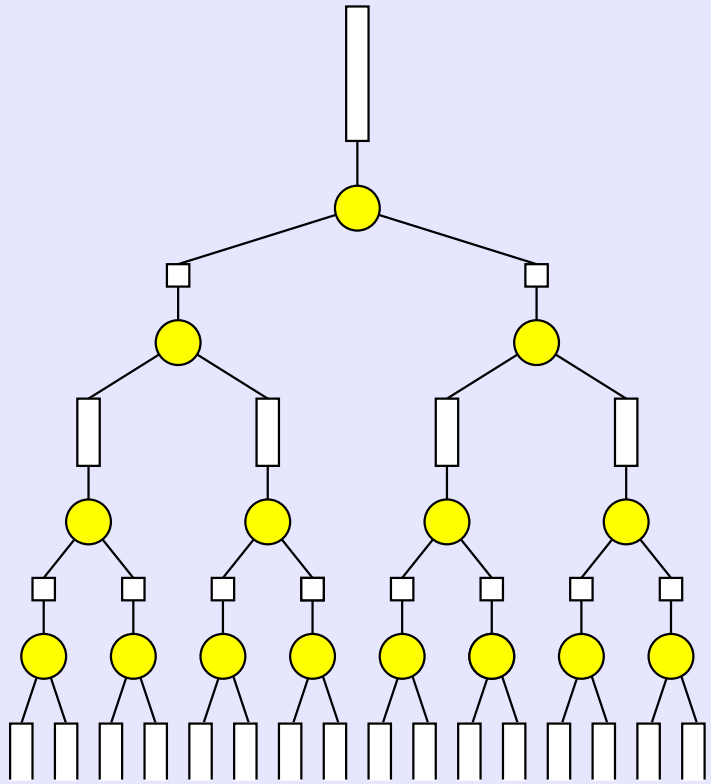


Brodal and Fagerberg 2002

Lazy k-merger







Buffer size $\alpha \cdot \sqrt{k}^{\,d}$

Brodal and Fagerberg 2002

Engineering Lazy Funnelsort

- Recursive implementation beats iterative implementation
- 4- or 5-way merge beats 2-way merge
- Standard memory allocator beats hand-coded allocator
- Pointer based van Emde Boas layout beats implicit layouts
- Nodes and buffers stored separately beats one layout
- Straightforward beats hand-coded branch elimination in core loop
- $d = 2.5$ and $\alpha = 16$
- Reuse merger data structures
- For k-mergers of height < 2 switch to Quicksort

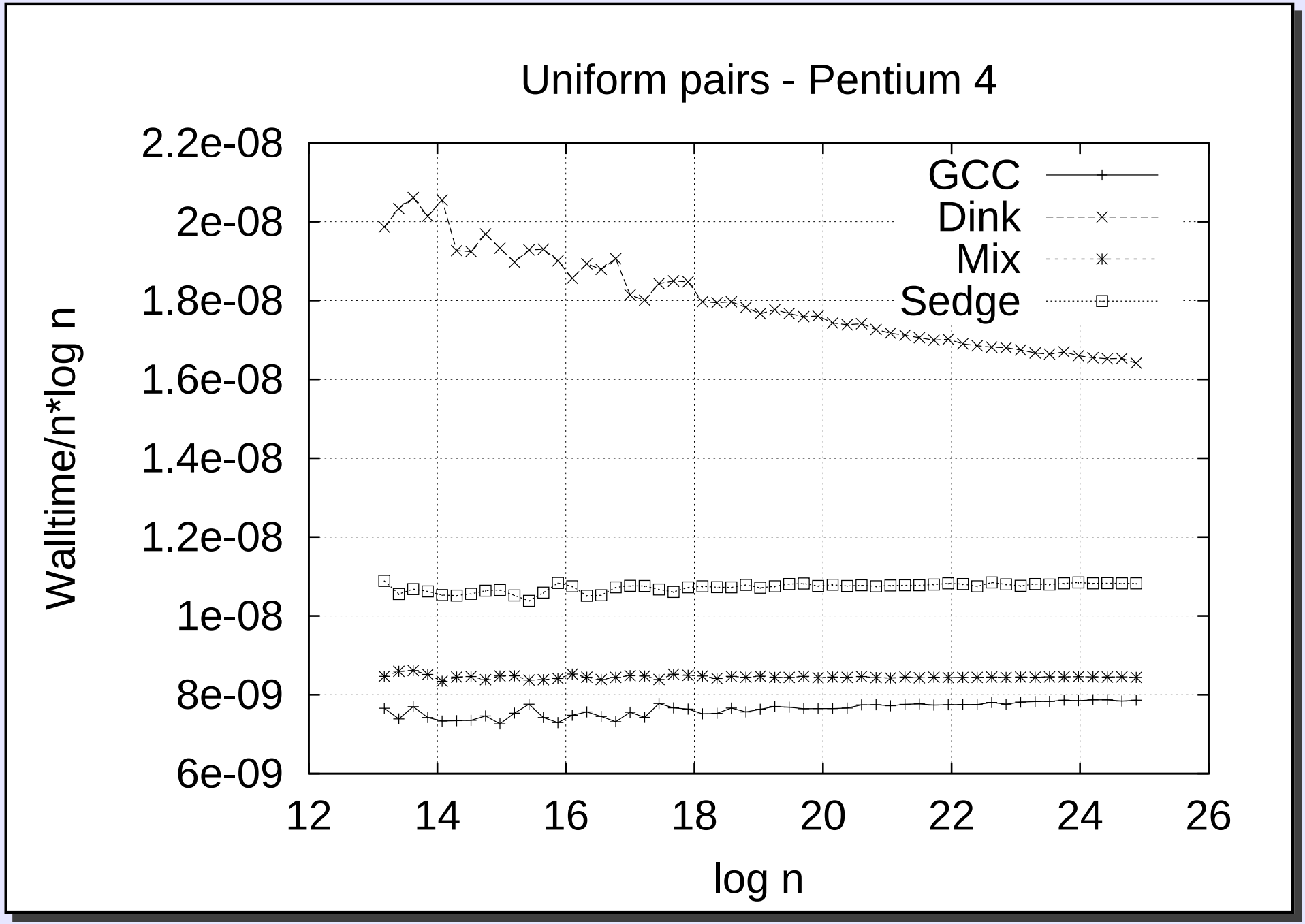
Evaluating Funnelsort

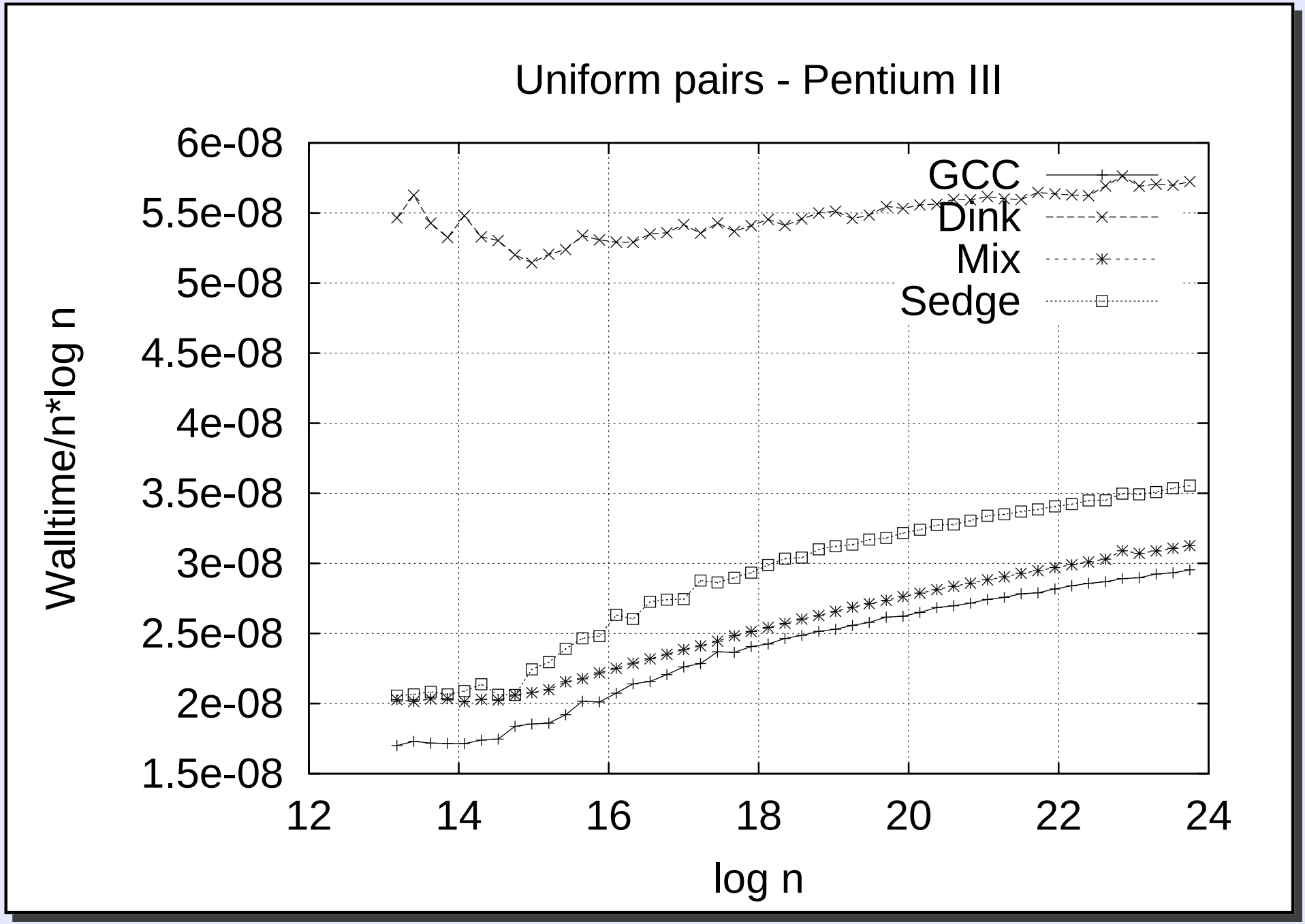
- 2-way and 4-way Funnelsort
- Quicksort
  - STL GCC
  - STL Intel C++
  - Sedgewick
  - Bentley & MacIlroy with pivot tuning
- TPIE - optimized for external memory
- Rmerge - optimized for registers
- Mergesort by LaMarca and Ladner

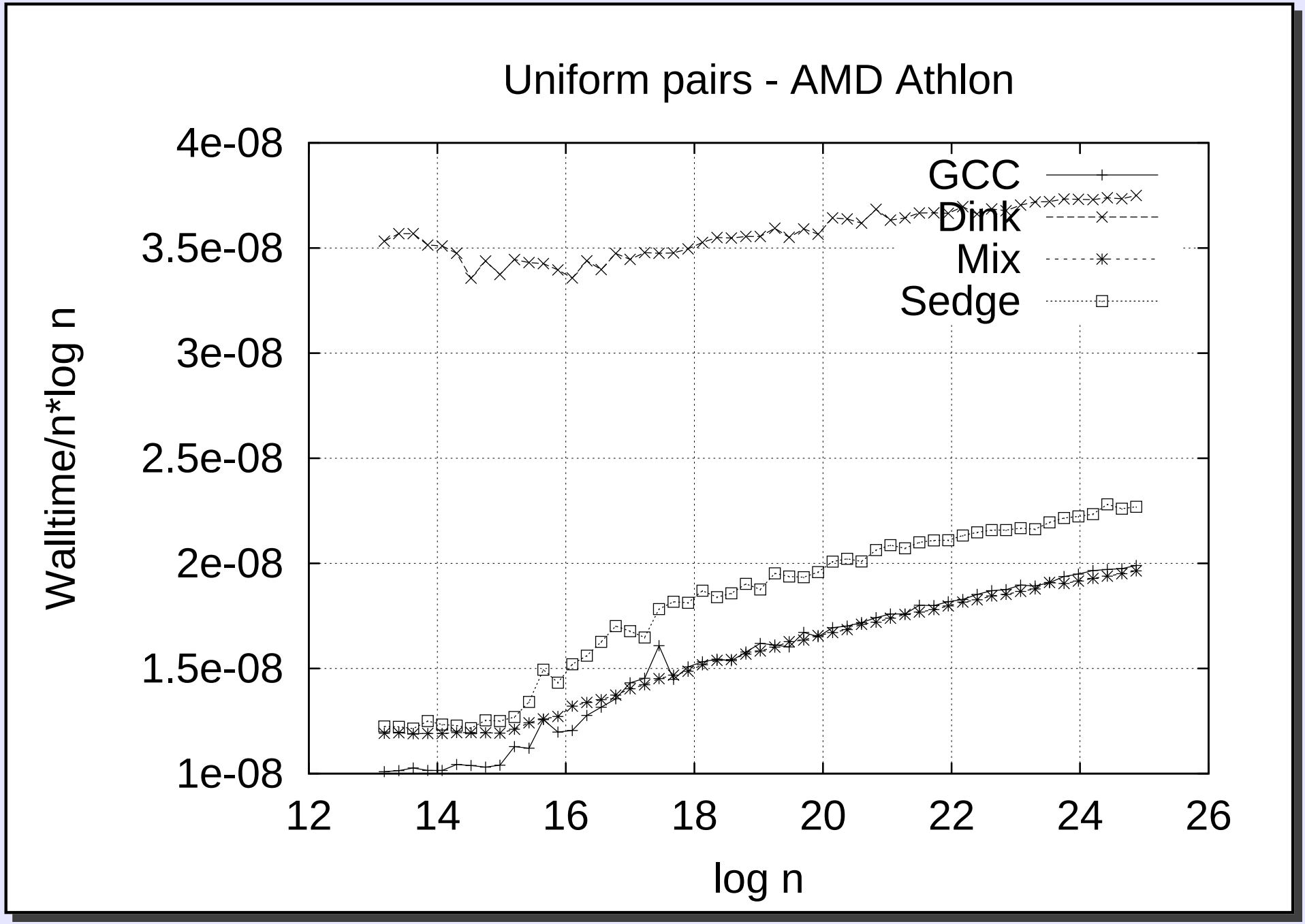
Hardware Specifications

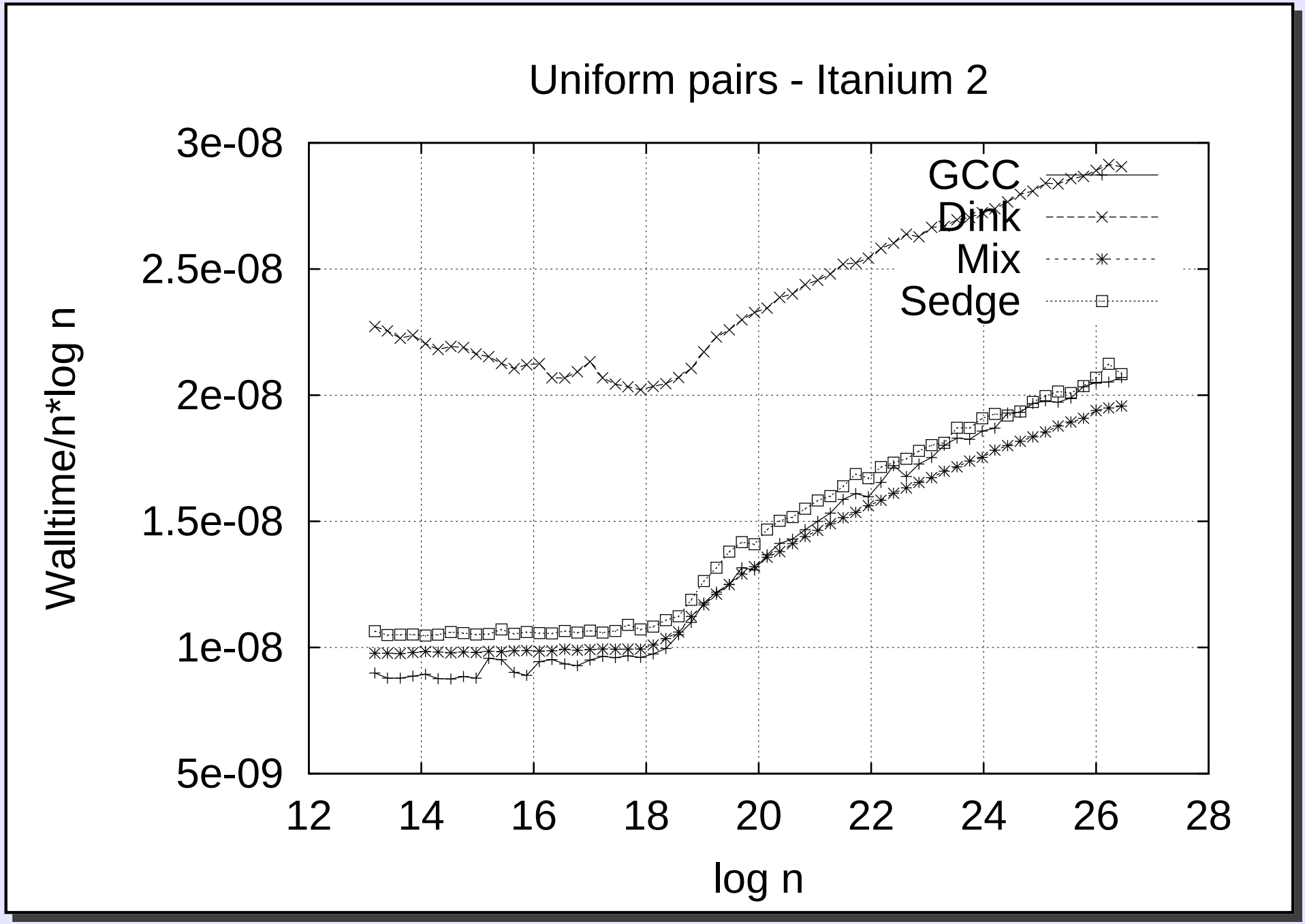
	Pentium 4	*Pentium III*	*MIPS 10000*	*AMD Athlon*	*Itanium 2*
Architecture type	Modern CISC	Classic CISC	RISC	Modern CISC	EPIC
Operation system	Linux v. 2.4.18	Linux v. 2.4.18	IRIX v. 6.5	Linux 2.4.18	Linux 2.4.18
Clock rate	2400MHz	800MHz	175MHz	1333 MHz	1137 MHz
Address space	32 bit	32 bit	64 bit	32 bit	64 bit
Pipeline stages	20	12	6	10	8
L1 data cache size	8 KB	16 KB	32 KB	128 KB	32 KB
L1 line size	128 B	32 B	32 B	64 B	64 B
L1 associativity	4-way	4-way	2-way	2-way	4-way
L2 cache size	512 KB	256 KB	1024 KB	256 KB	256 KB
L2 line size	128 B	32 B	32 B	64 B	128 B
L2 associativity	8-way	4-way	2-way	8-way	8-way
TLB entries	128	64	64	40	128
TLB associativity	full	4-way	64-way	4-way	full
TLB miss handling	hardware	hardware	software	hardware	?
RAM size	512 MB	256 MB	128 MB	512 MB	3072 MB

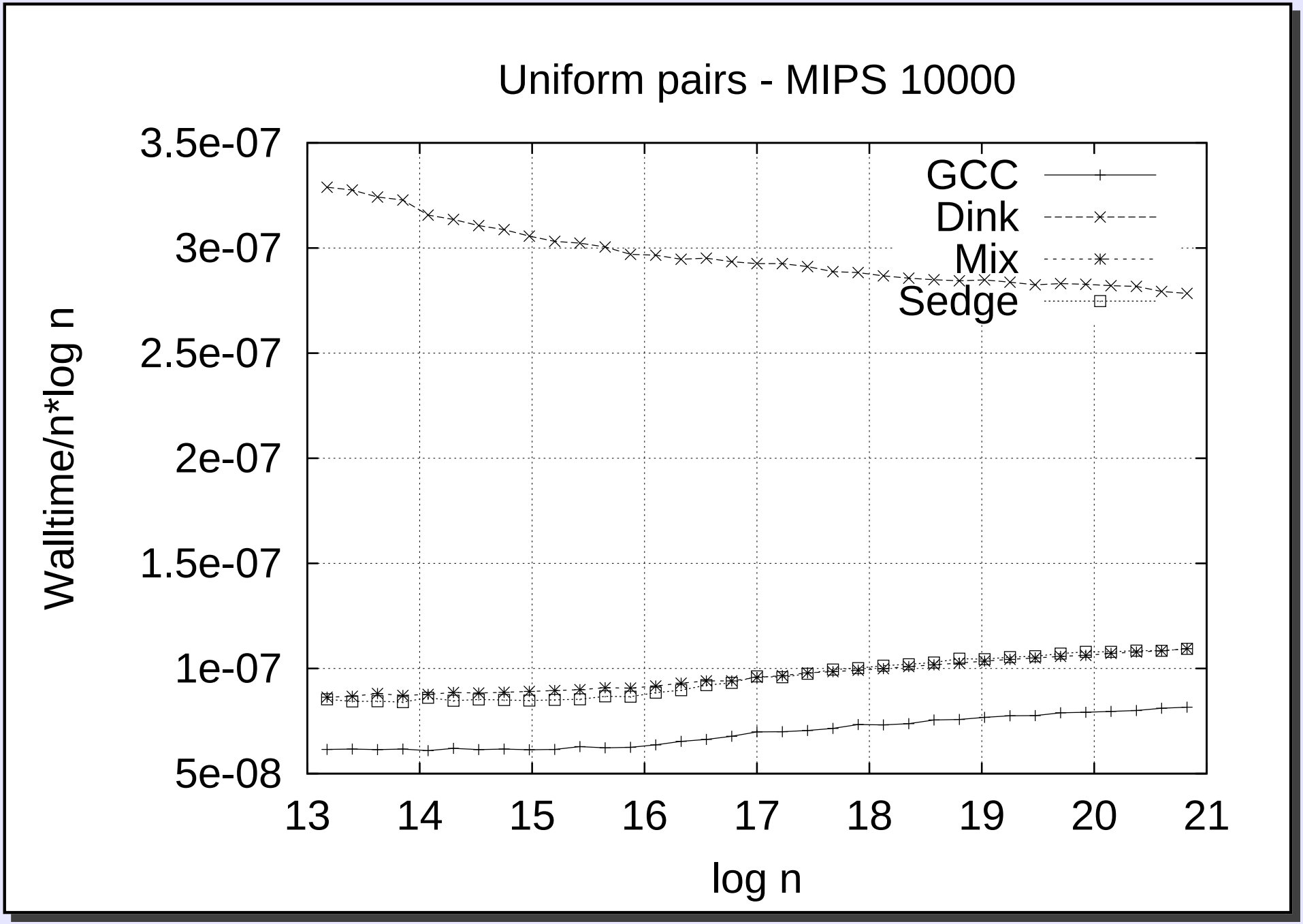
Comparison of Quicksort Implementations


Uniform pairs - Pentium 4
Walltime/n*log n
log n
GCC
Dink
Mix
Sedge
2.2e-08
2e-08
1.8e-08
1.6e-08
1.4e-08
1.2e-08
1e-08
8e-09
6e-09
12
14
16
18
20
22
24
26

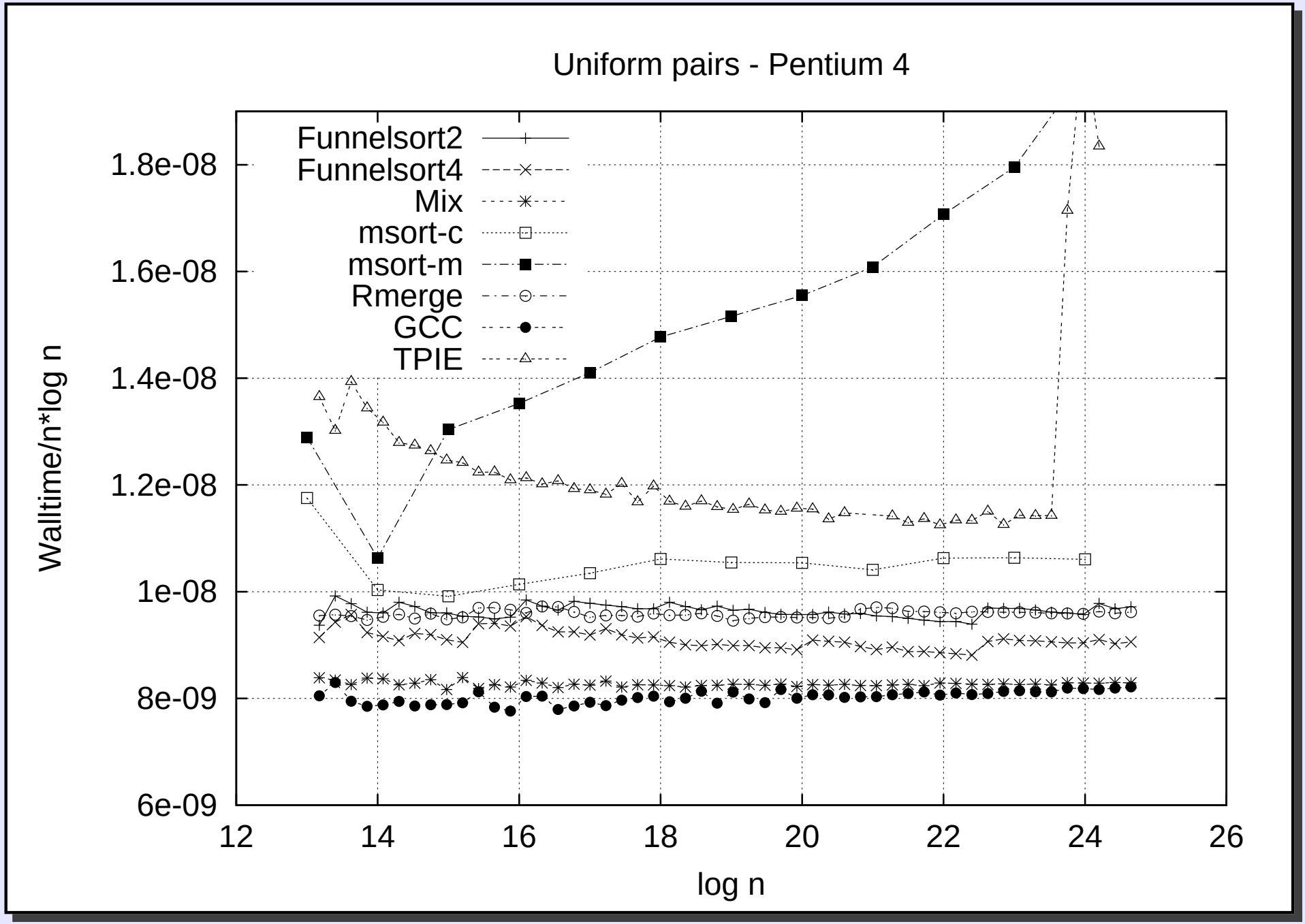

Uniform pairs - Pentium III
Walltime/n*log n
log n
6e-08
5.5e-08
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
12
14
16
18
20
22
24
GCC
Dink
Mix
Sedge

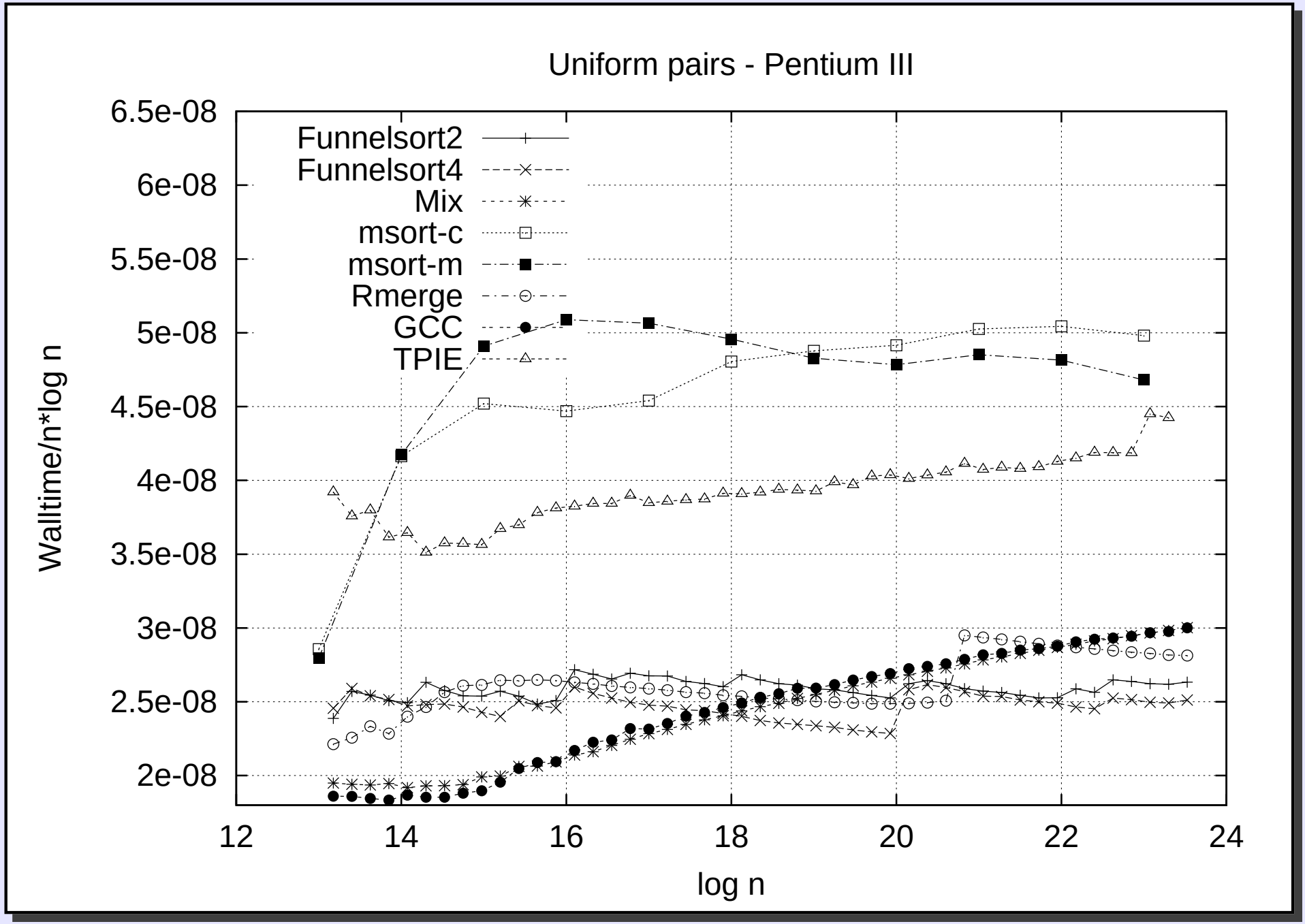

Uniform pairs - AMD Athlon
Walltime/n*log n
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
12
14
16
18
20
22
24
26
log n
GCC
Dink
Mix
Sedge

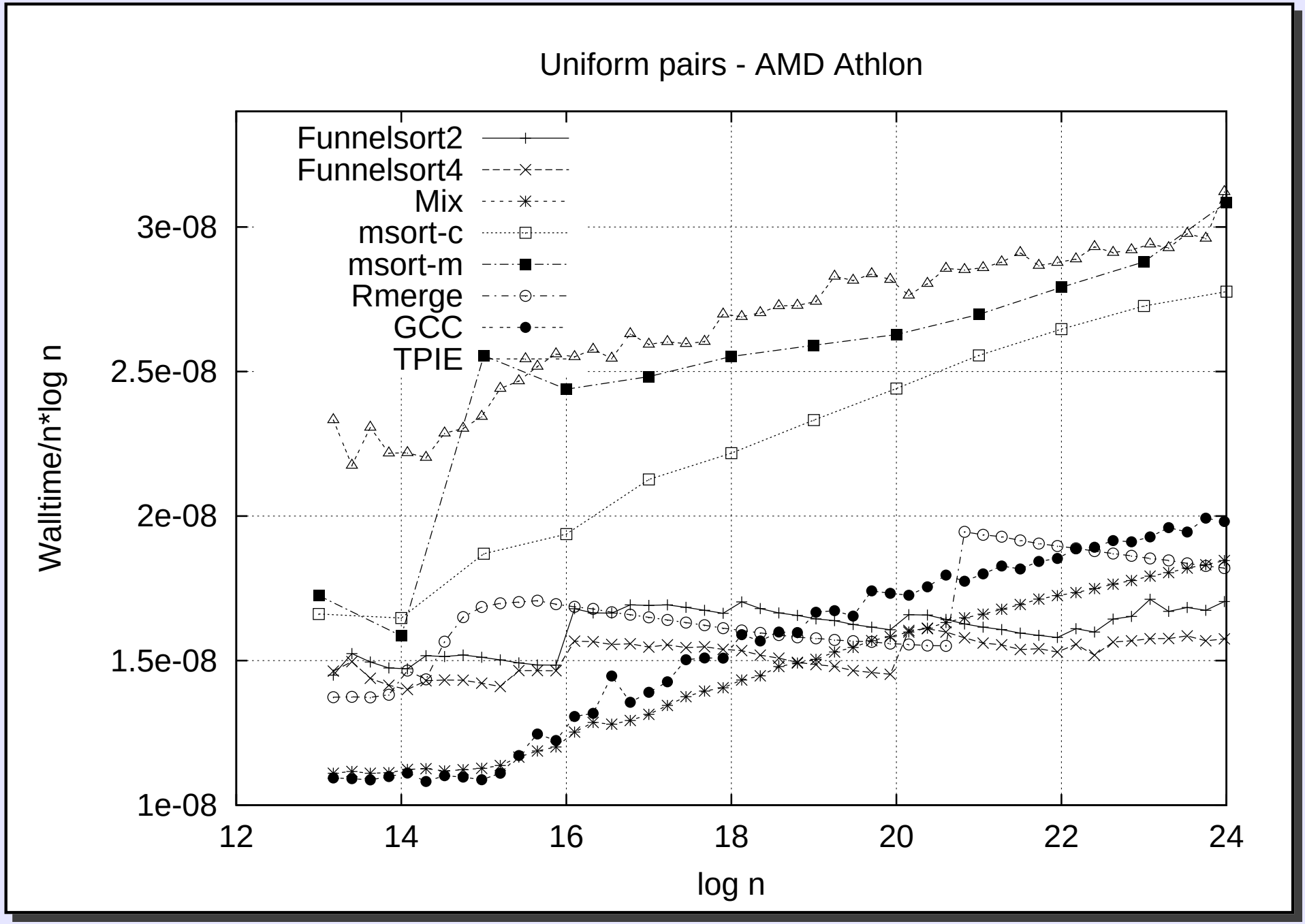

Uniform pairs - Itanium 2
Walltime/n*log n
log n
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
5e-09
12
14
16
18
20
22
24
26
28
GCC
Dink
Mix
Sedge

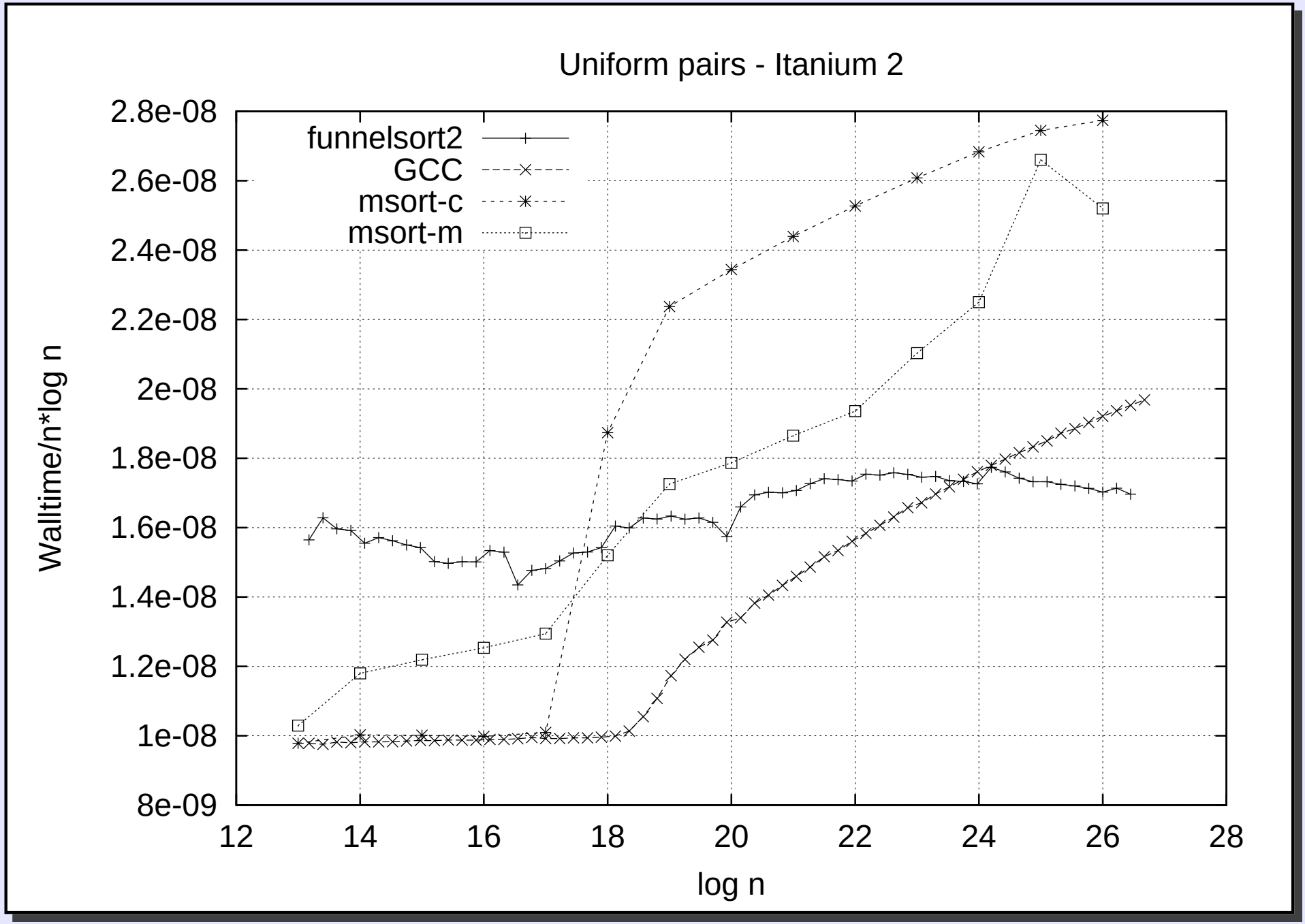

Uniform pairs - MIPS 10000
Walltime/n*log n
3.5e-07
3e-07
2.5e-07
2e-07
1.5e-07
1e-07
5e-08
13
14
15
16
17
18
19
20
21
log n
GCC
Dink
Mix
Sedge

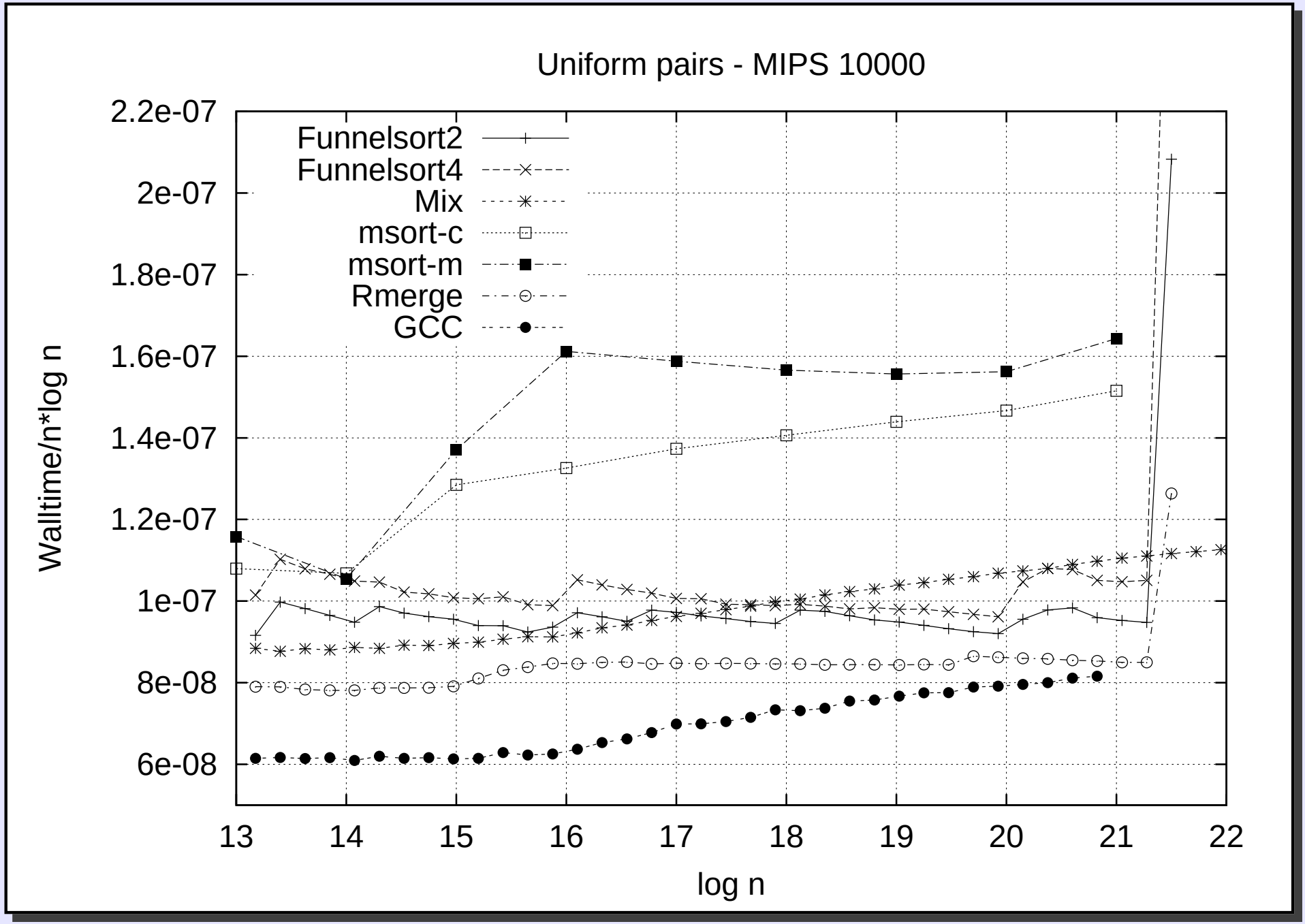
Results for Inputs in RAM


Uniform pairs - Pentium 4
Funnelsort2
Funnelsort4
Mix
msort-c
msort-m
Rmerge
GCC
TPIE
Walltime/n*log n
1.8e-08
1.6e-08
1.4e-08
1.2e-08
1e-08
8e-09
6e-09
12
14
16
18
20
22
24
26
log n

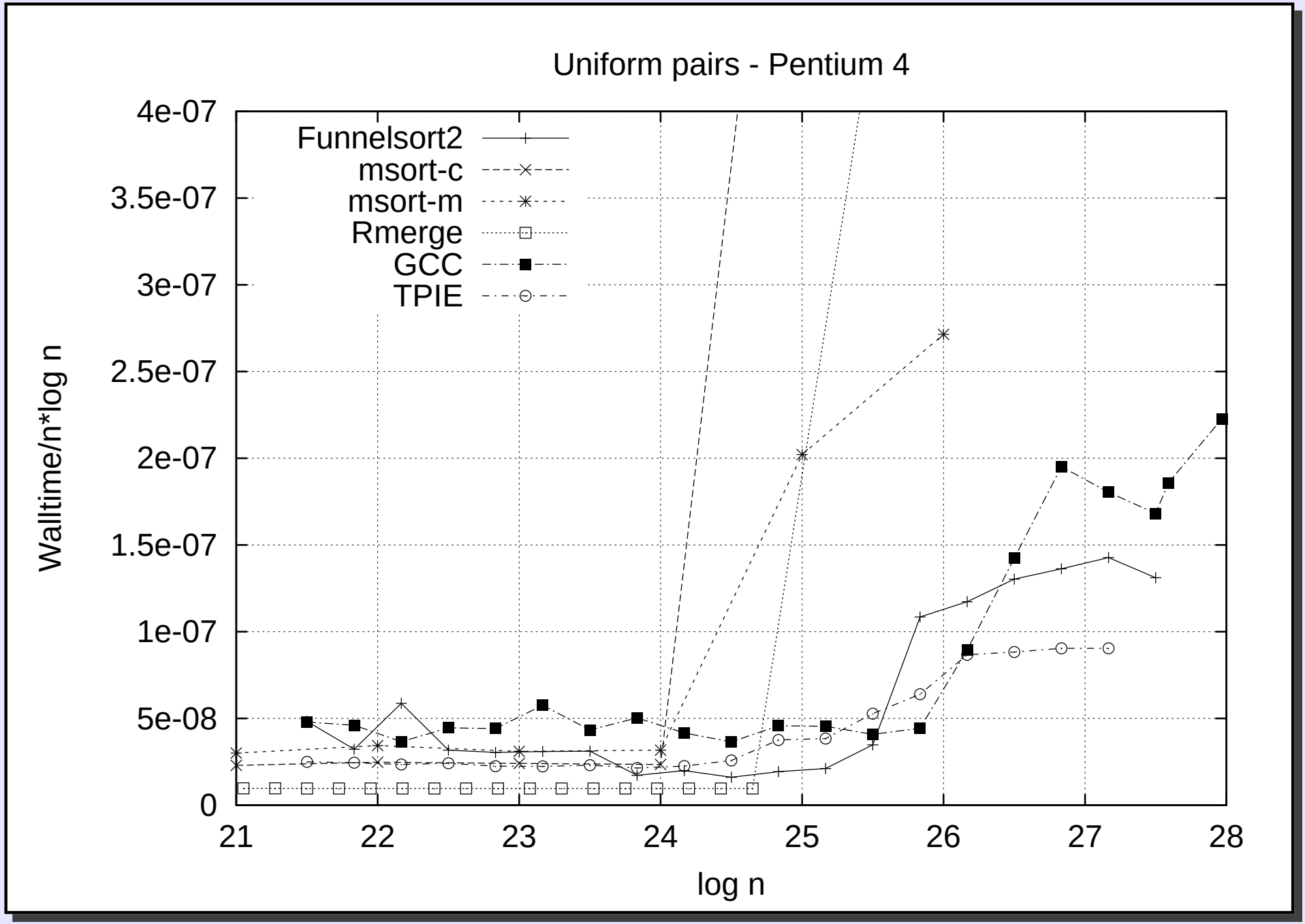

Uniform pairs - Pentium III
Funnelsort2
Funnelsort4
Mix
msort-c
msort-m
Rmerge
GCC
TPIE
Walltime/n*log n
log n
6.5e-08
6e-08
5.5e-08
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
12
14
16
18
20
22
24

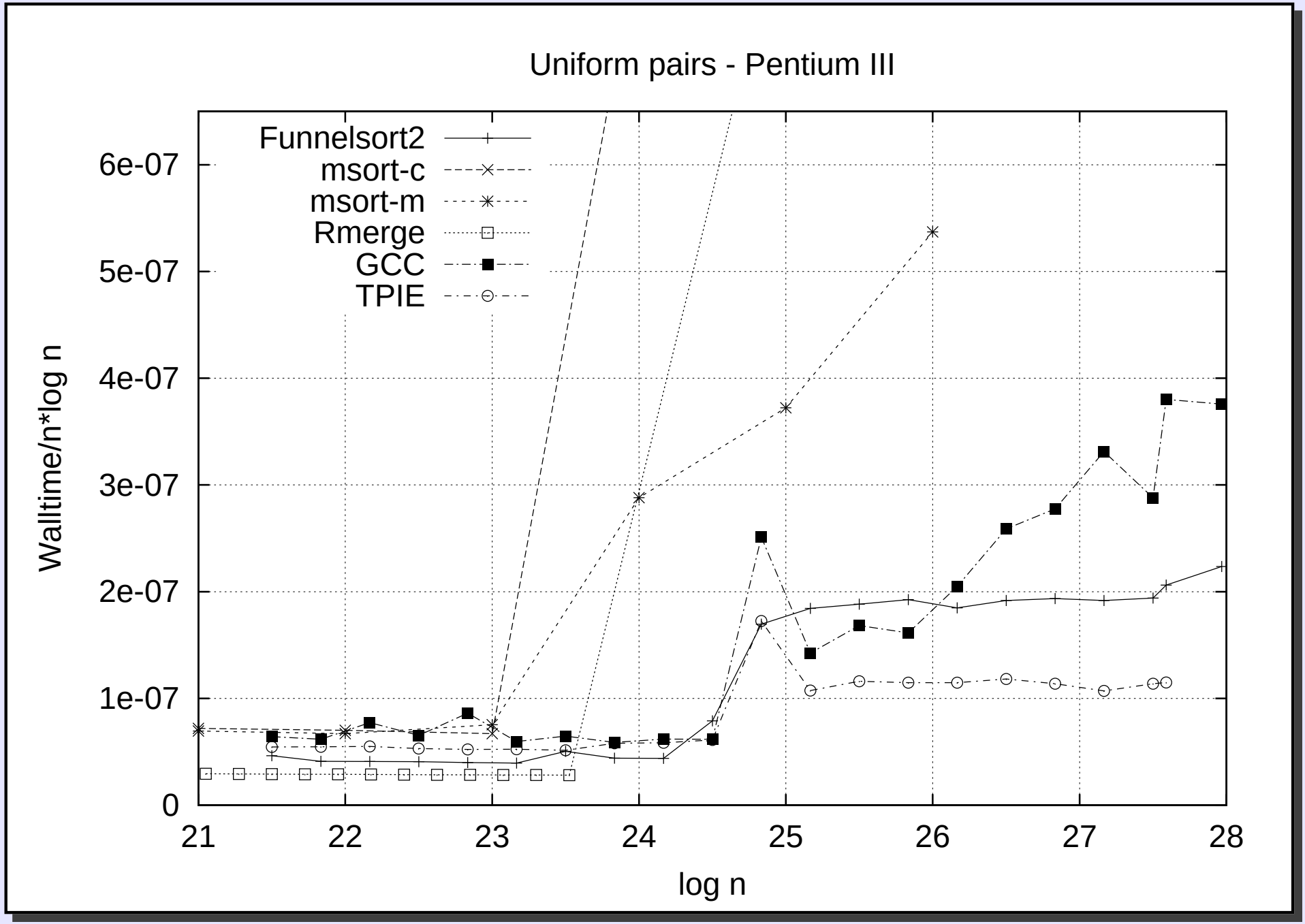

Uniform pairs - AMD Athlon
Funnelsort2
Funnelsort4
Mix
msort-c
msort-m
Rmerge
GCC
TPIE
Walltime/n*log n
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
12
14
16
18
20
22
24
log n

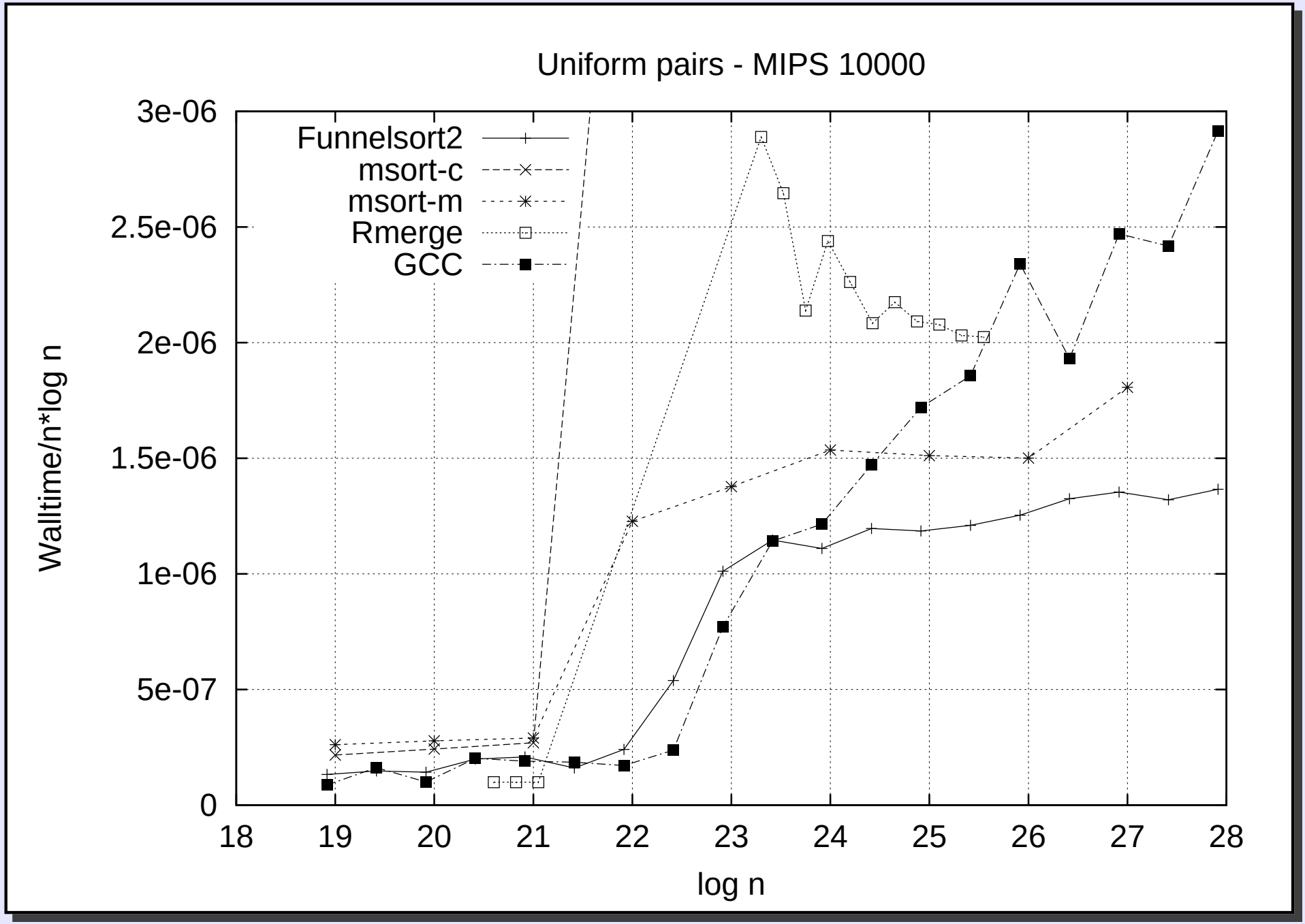

Uniform pairs - Itanium 2
funnelsort2
GCC
msort-c
msort-m
Walltime/n*log n
log n
2.8e-08
2.6e-08
2.4e-08
2.2e-08
2e-08
1.8e-08
1.6e-08
1.4e-08
1.2e-08
1e-08
8e-09
12
14
16
18
20
22
24
26
28


Uniform pairs - MIPS 10000
Funnelsort2
Funnelsort4
Mix
msort-c
msort-m
Rmerge
GCC
Walltime/n*log n
2.2e-07
2e-07
1.8e-07
1.6e-07
1.4e-07
1.2e-07
1e-07
8e-08
6e-08
13
14
15
16
17
18
19
20
21
22
log n

Results for Inputs on Disk


Uniform pairs - Pentium 4
Funnelsort2
msort-c
msort-m
Rmerge
GCC
TPIE
Walltime/n*log n
4e-07
3.5e-07
3e-07
2.5e-07
2e-07
1.5e-07
1e-07
5e-08
0
21
22
23
24
25
26
27
28
log n


Uniform pairs - Pentium III
Funnelsort2
msort-c
msort-m
Rmerge
GCC
TPIE
Walltime/n*log n
0
1e-07
2e-07
3e-07
4e-07
5e-07
6e-07
21
22
23
24
25
26
27
28
log n


Uniform pairs - MIPS 10000
Funnelsort2
msort-c
msort-m
Rmerge
GCC
Walltime/n*log n
3e-06
2.5e-06
2e-06
1.5e-06
1e-06
5e-07
0
18
19
20
21
22
23
24
25
26
27
28
log n

Conclusion

- Very high performing generic sorting algorithm
- Performance remains robust across wide range of input sizes
- Across several different processor and operating system architectures
- On several different data types
- On several different input distributions
- *Overhead involved in being cache-oblivious can be small enough for the nice theoretical properties to actually transfer into practical advantages*