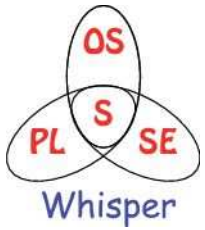


Protecting Real-Time Applications against Memory Induced Slowdown on a Small Multicore System

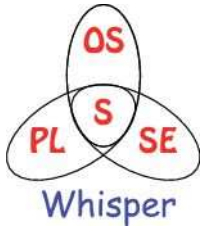
Ongoing work

Gilles Muller *

Antoine Blin+*, Julien Sopena*, Julia Lawall*

+ Renault

* Inria / Lip6 / Sorbonne Universités



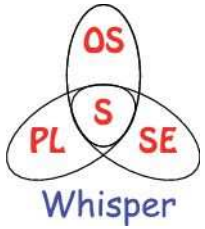
The problem

- Real time
- Best effort

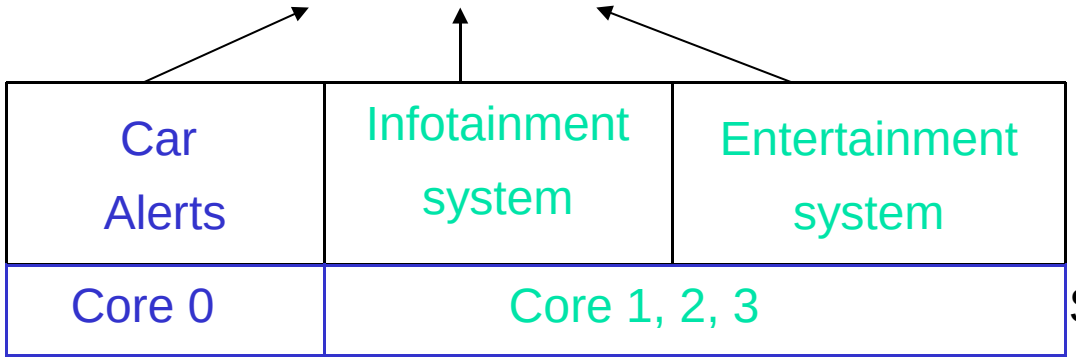
Car Alerts

Infotainment system

Entertainment system

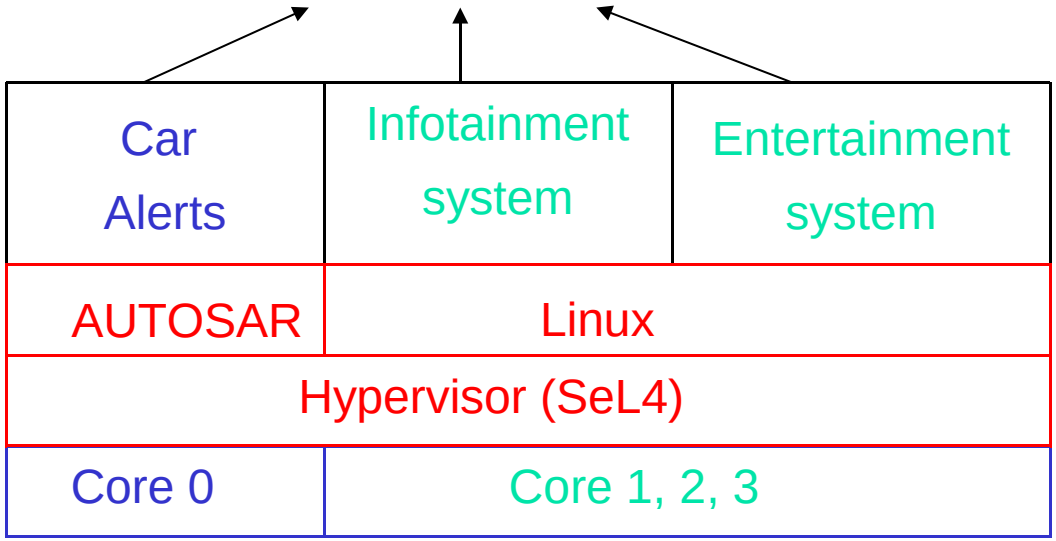


How to reduce the cost of hardware?



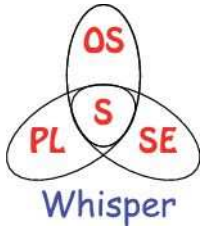
Sabre Lite IMX 6

The software stack

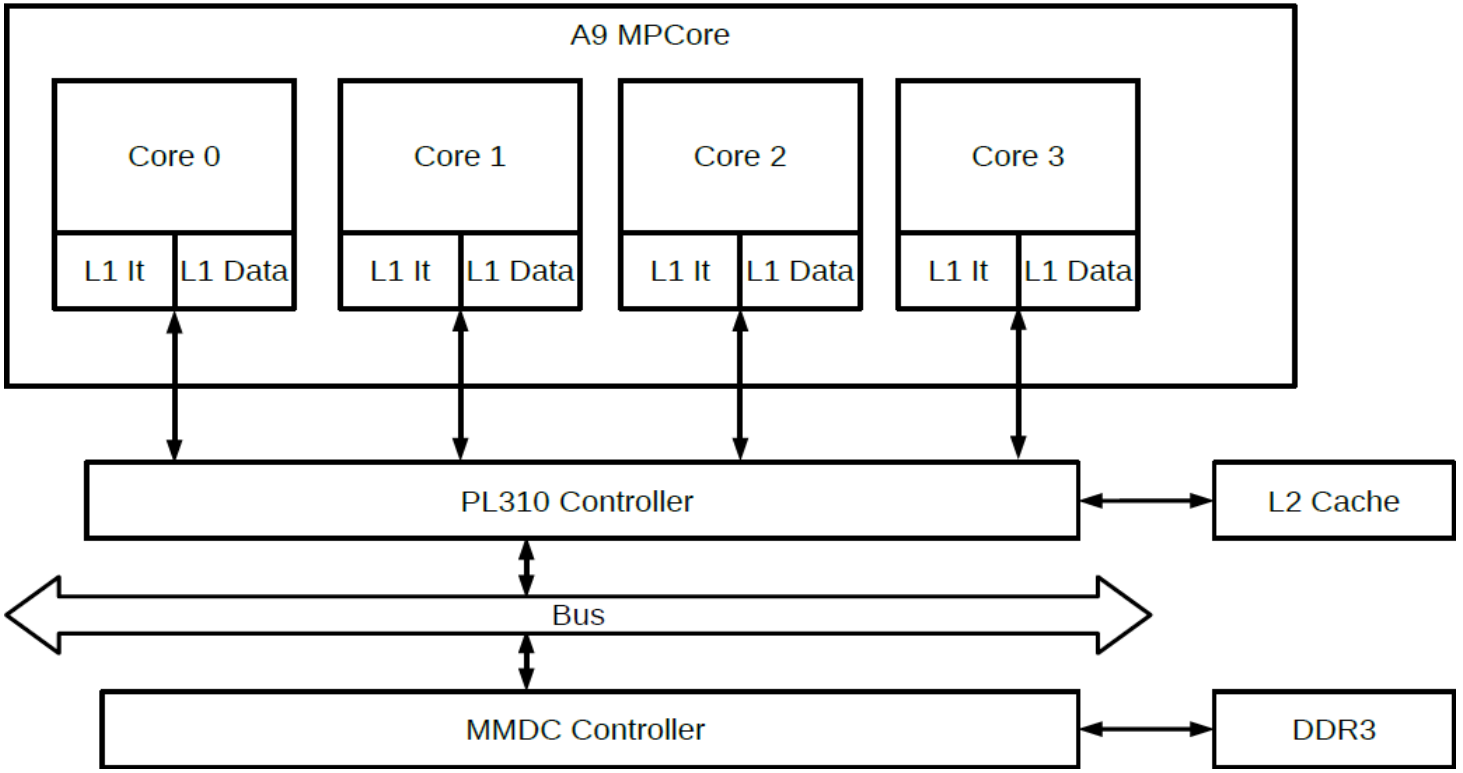


- Legacy software
- CPU isolation

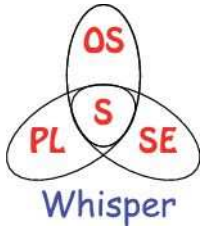
Sabre Lite IMX 6



The research problem: L2 and Memory shared by all cores !



What happens if the Best Effort applications use a lot of memory?

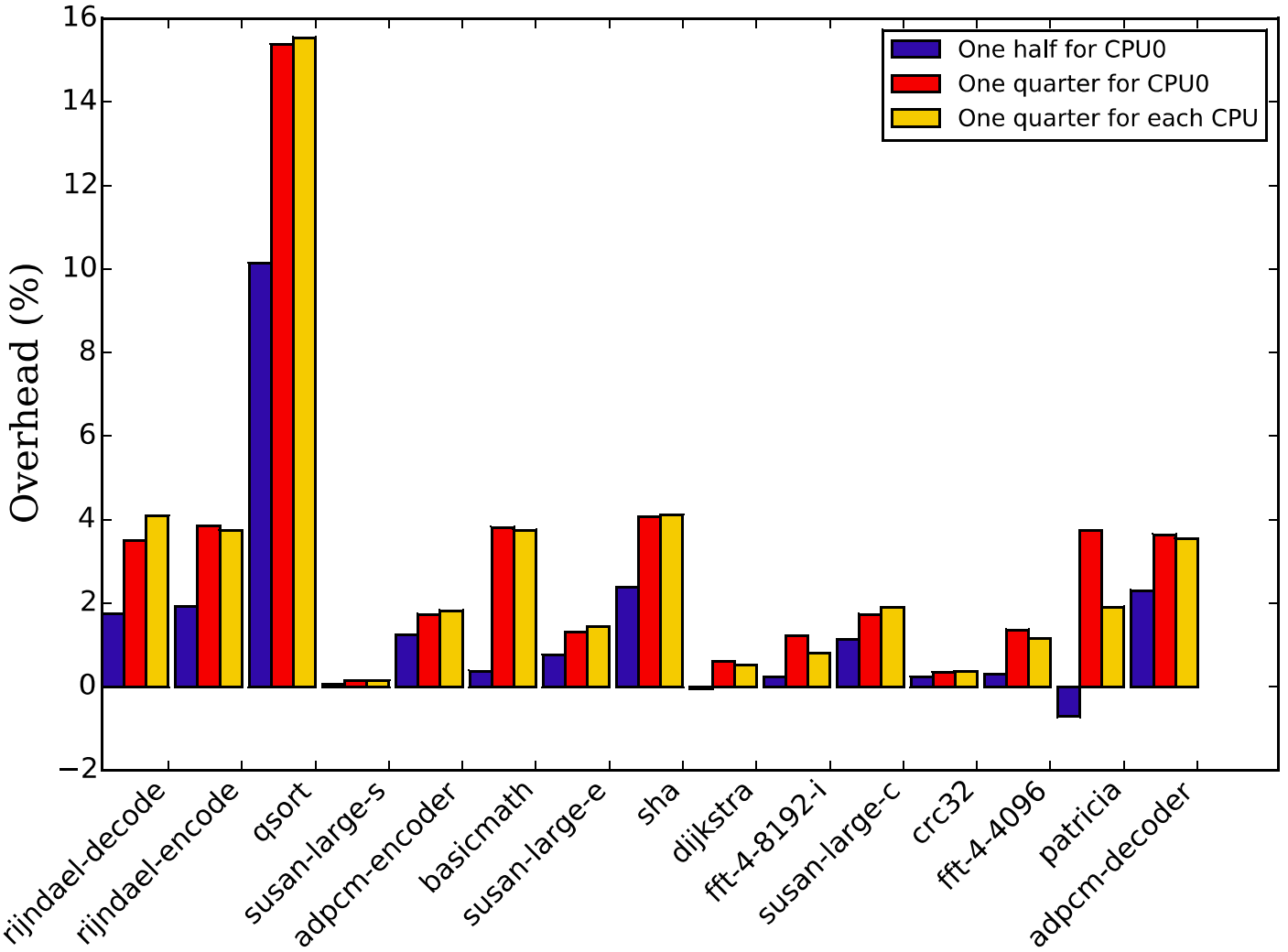


MiBench under contention

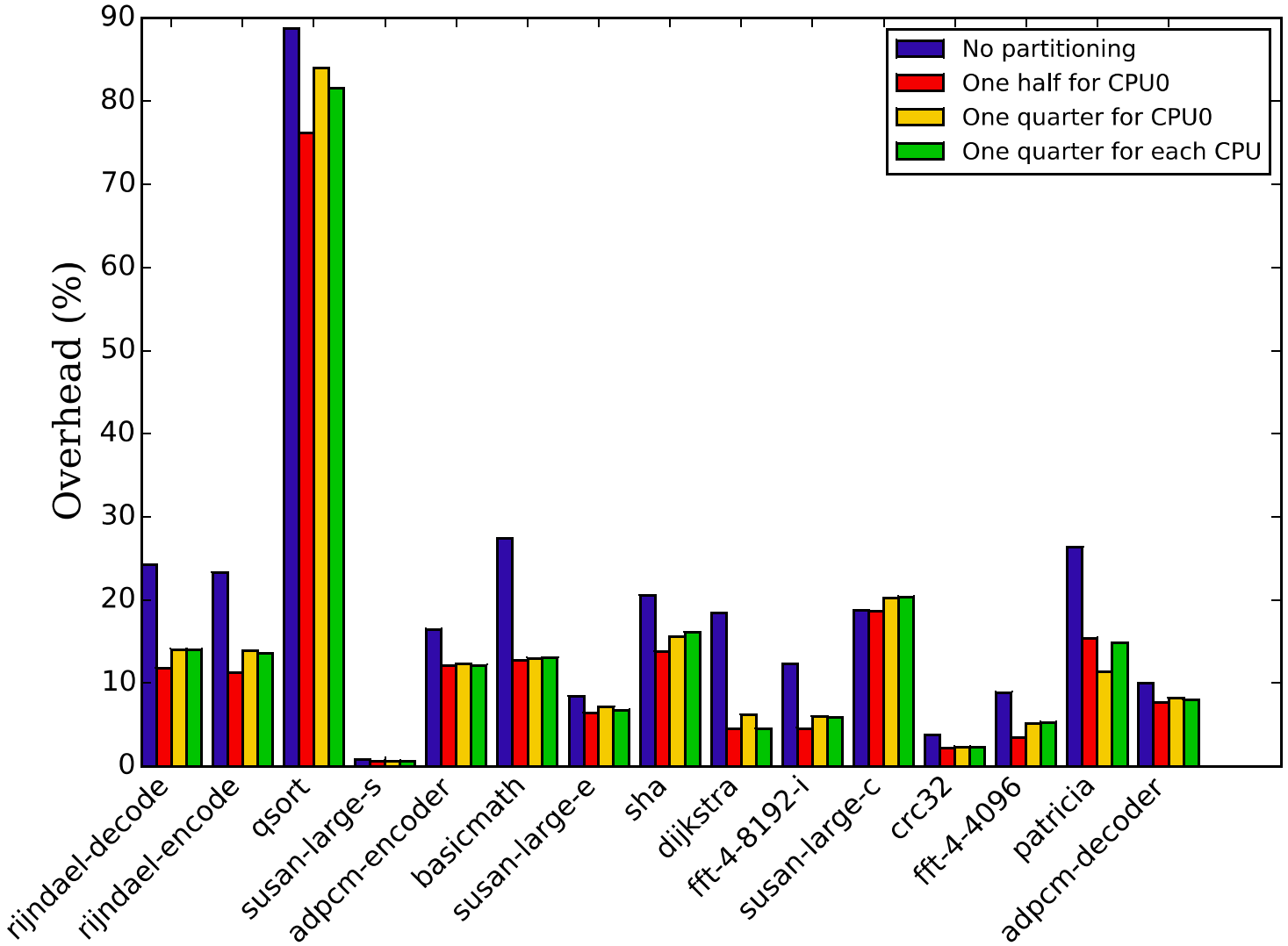
- Benchmark suite for embedded systems
  - Automotive. Industrial Control, Networking, and Telecommunications
  - 35 applications (different data sets)
  - Exclude 19 applications
    - X86, office related or long running

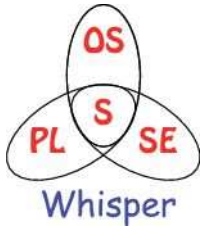
- MiBench on one core, 3 loads on the other cores
  - **"Add"** kernel from the Stream suite
  - Compilation options selected to generate the highest load

Impact of L2 cache partitioning



Partitioning and contention

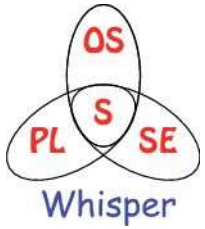




Our Goals

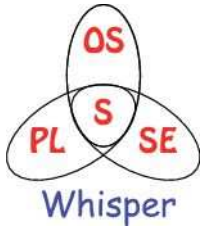
- Protection : Ensure that the memory induced overhead for RT applications remains below a threshold
  - Suspend best-effort applications if the threshold is reached

- Parallelism : Avoid suspending the best-effort applications when acceptable
  - Baseline: run real-time application and best-effort applications in exclusion



Our approach

- Estimate the overhead based on memory traffic
  - Periodic sample (100 µs) of the memory traffic during execution
  - Conservative computing of the overhead for the current sample
- Suspend the Best-Effort applications if the cumulated overhead is greater than the desired threshold

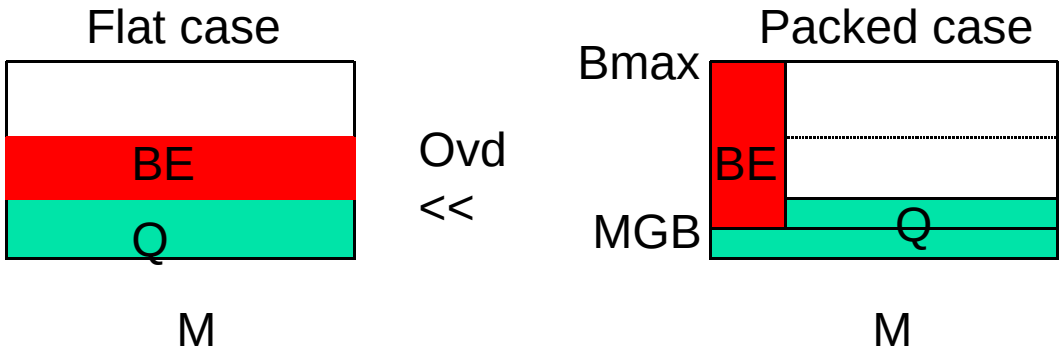


Conservative Computing of the Overhead

What is the worst overhead for a given sample value M?

- Off-line profiling of the Real-Time application
  - *Q : maximum bandwidth for the real-time application*

- We measure a quantity, not a bandwidth
  - Estimate the worst packing case from the flat case

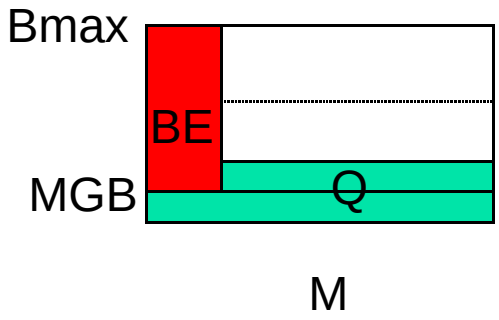


Estimating the packed case (1)

M = Measured Bandwidth

Bmax = Maximum measured Bandwidth for a realtime Q

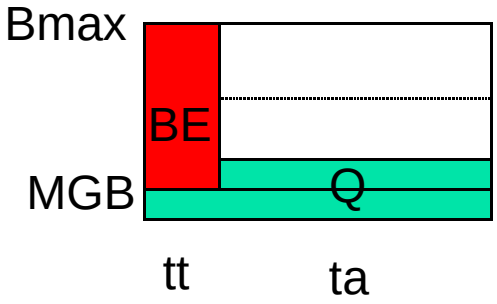
MGB = Minimum measured Guaranteed Bandwidth



$$MGB = \frac{Q}{Flat\_overhead(Bmax) + 1}$$

Estimating the packed case (2)

tt = time together
ta = time alone

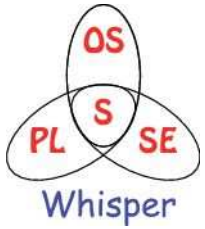


$$M = Bmax \cdot tt + Q \cdot ta$$

$$1 = ta + tt$$

$$PackedOvd = \frac{Q}{MGB \cdot tt + Q \cdot ta} - 1$$

$$= \frac{Q}{MGB \cdot tt + Q - Q \cdot tt} - 1$$

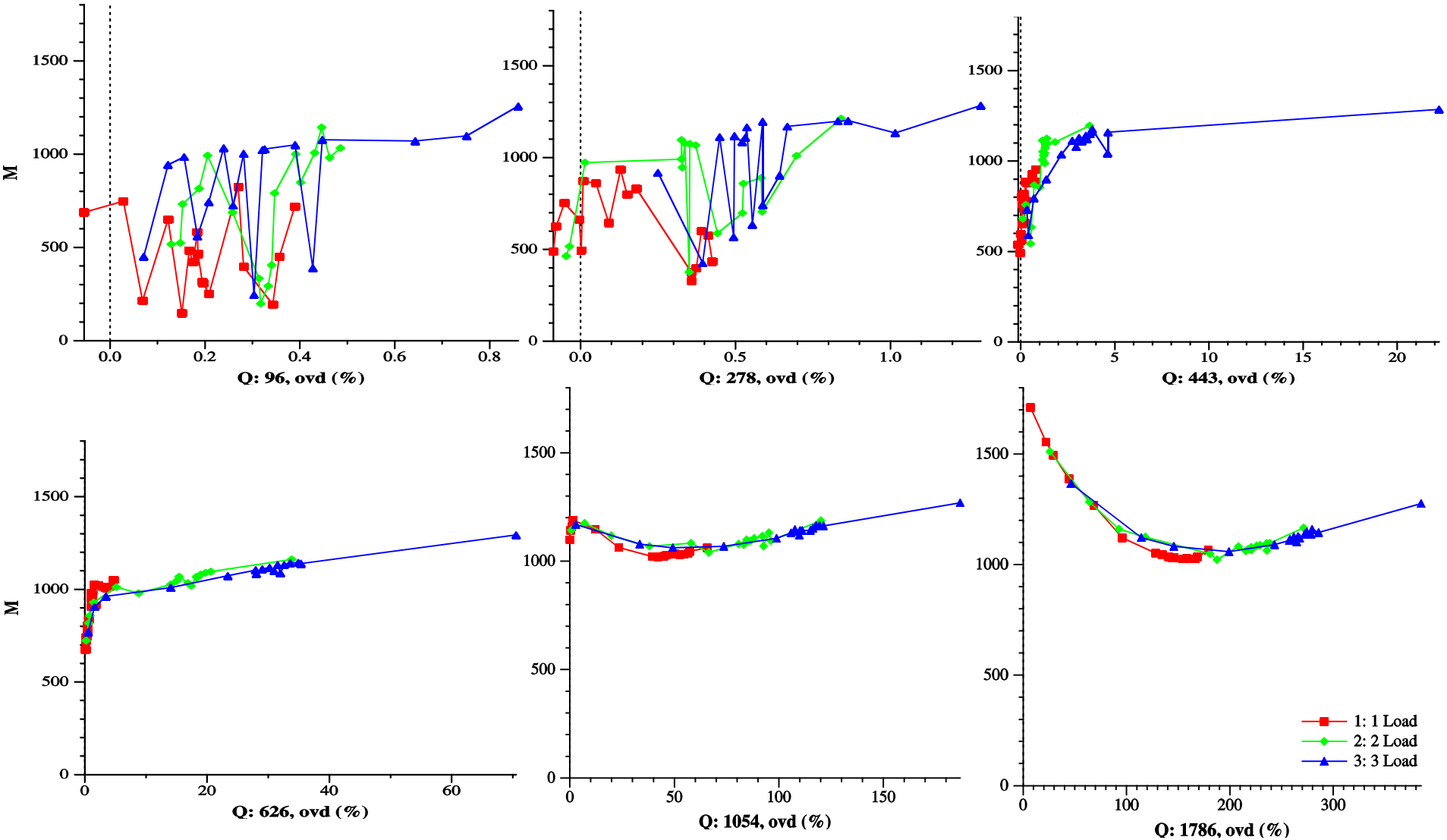


Measuring the Flat case

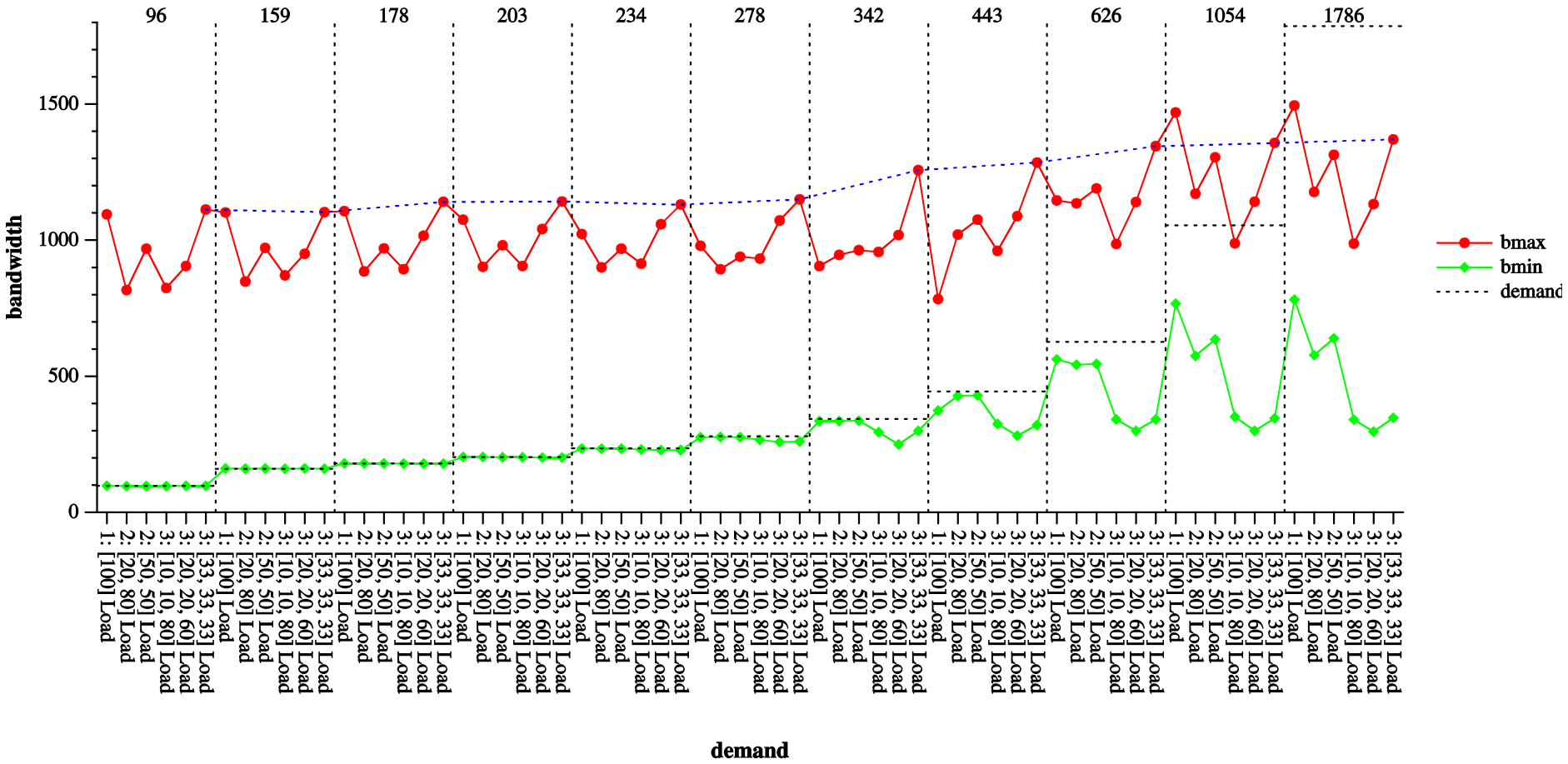
- RT periodic microbenchmark with constant rate of memory access
  - Array copy to generate traffic
  - Delay loop to generate lower traffic
  - GCC 4.6.3 using the -O2 option

- "Add" kernel as load with varying delay
  - Loads from 86MB/s to 1786MB/s

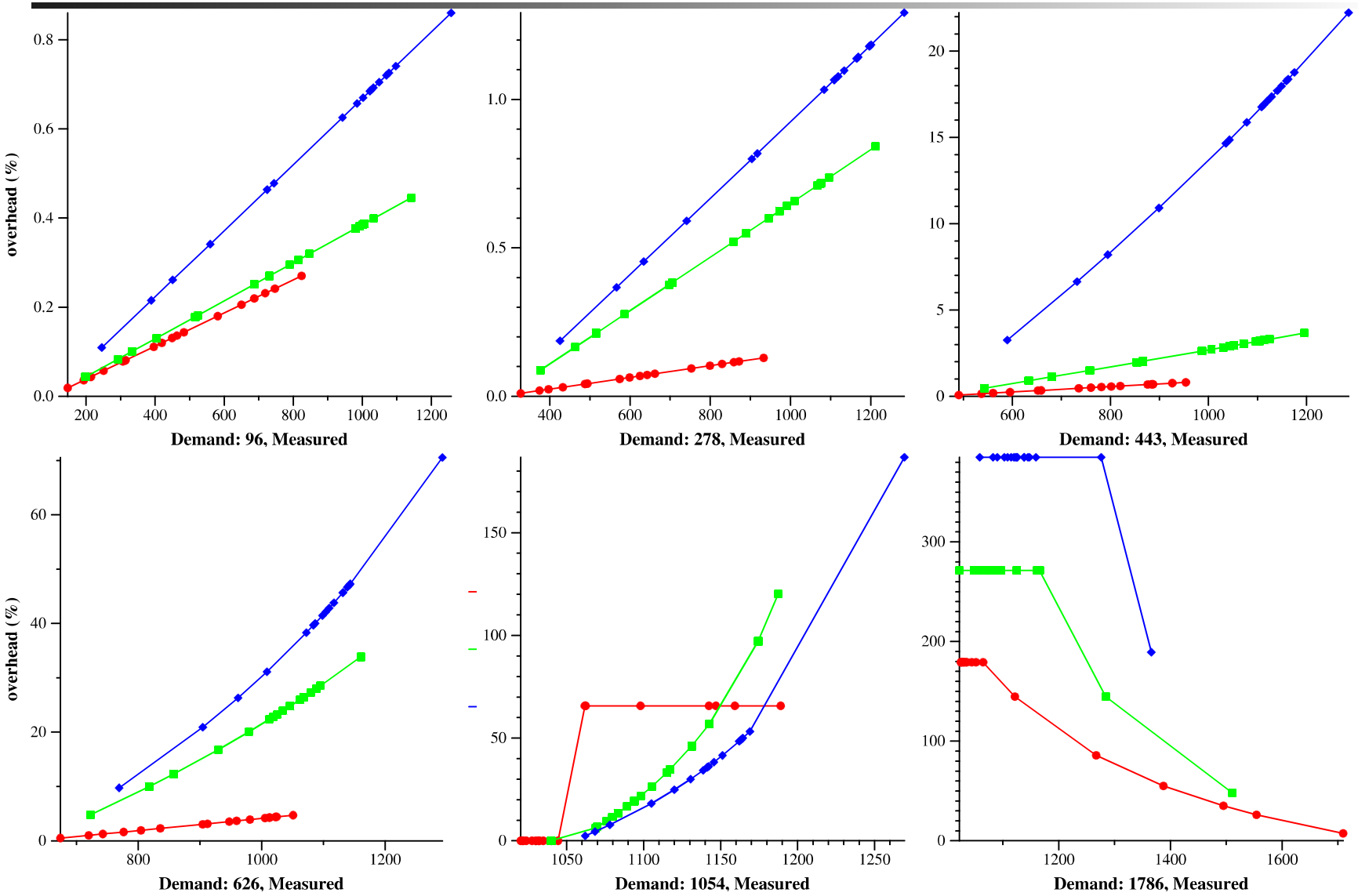
Flat case results

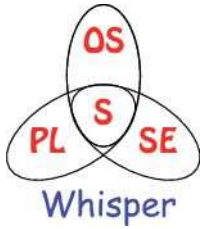


MGB and Bmax results



Packed Overhead

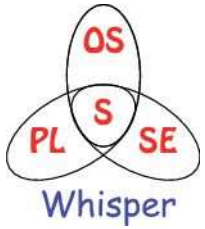




Off-line preparation

Minimize the cost of computations at run-time

- Pre-compute a set of curves for possible values of Q
  - Measured bandwidth is an index into a table of overhead values
- Values for Q
  - Based on profiling of the real-time application
  - [0…Maximum] or set of possible values

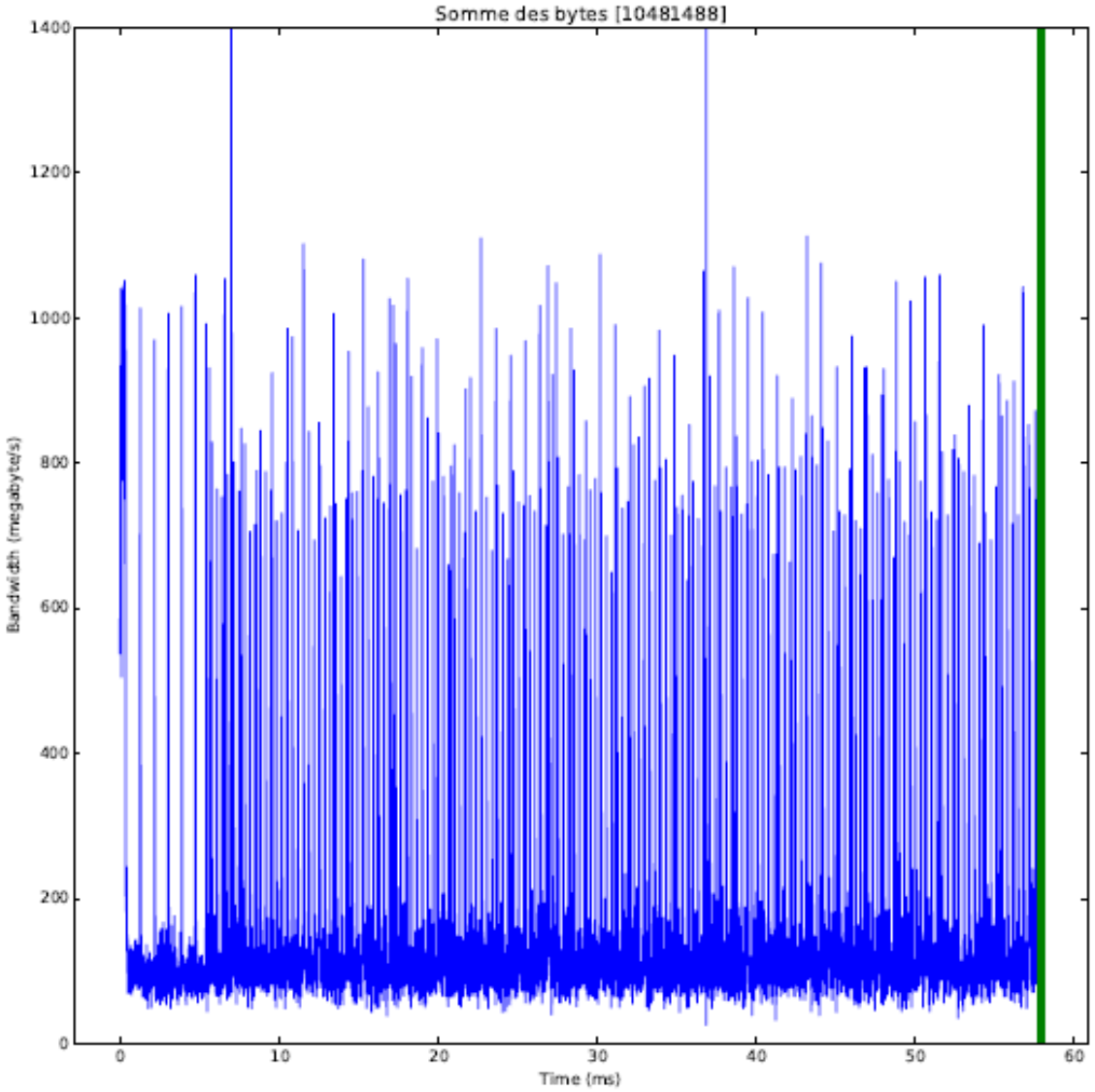


Constructing the memory profile of the real-time application

What is the maximum bandwith requirement at a given sample?

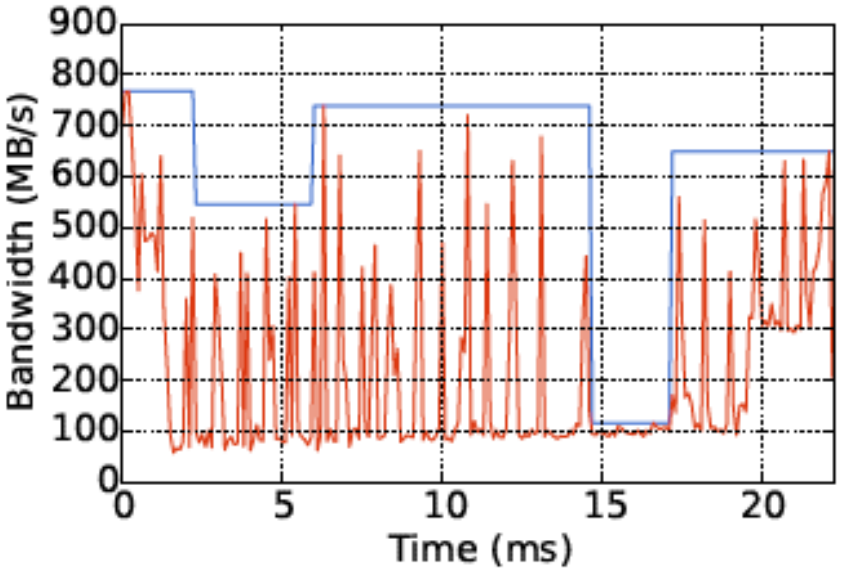
- High resolution sampling approach
- Similar problem to WCET estimation using tests
- Consider the maximum value in all possible paths
- Smooth the samples into a set of plateaux to mask execution variations
  - Adaptive approximation by piecewise constants
  - Merge samples that generate the least approximation

HiRes profiling - Patricia

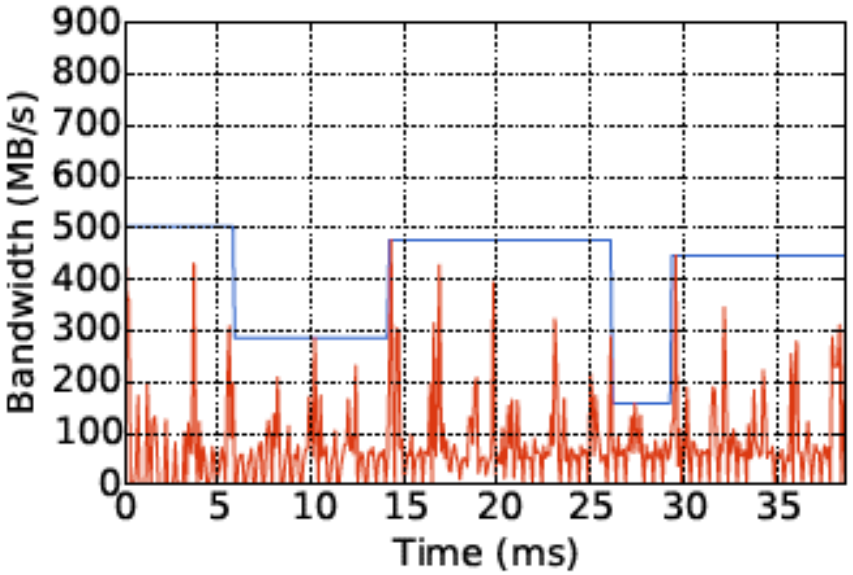


- 5 us resolution

MiBench profiles

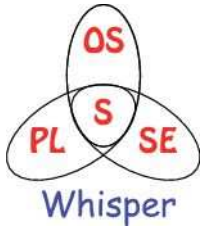


(a) Susan large -c



(g) Rijndael encode

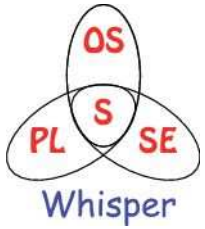
- 5 plateaux were sufficient to capture the main variations in memory bandwidth of the 8 MiBench applications



Run-time System

- Implementation within the Linux kernel
  - Kernel module, application profile communicated using sysfs

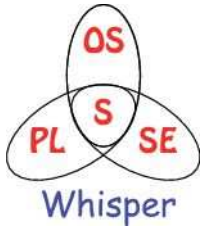
- Given the sample index, computation of the plateau which then determines the micro-benchmark instance
- Given the sample bandwidth value and the memory characterization table, we get the sample overhead



Preservation of real-time properties

- 1 to 3 instances of the "Add" kernel as loads

Application	Max run time alone (ms)	Overhead	
		No limit	5% limit
susan large -c	25.46	32%	3%
fft 4 8192 -i	24.34	22%	1%
qsort	51.68	20%	0%
fft 4 4096	10.74	18%	4%
rijndael encode	43.91	16%	3%
patricia	60.37	15%	1%
susan large -e	56.87	10%	2%
rijndael decode	41.52	10%	2%

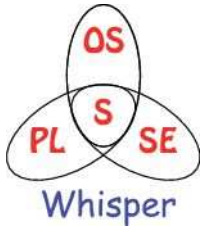


Progress of Best-Effort applications (in progress)

- Benefit: percentage of time when the BE applications run concurrently with the RT one

- « Add » kernel loads
- Multicore GPS-like application

- MiBench application activated every 100ms



Conclusion

- Preservation of the real-time properties
- *Better concurrency between BE and RT applications*

- What's next:
  - Selective BE application suspension based on L1 cache misses
  - Plateau optimization
  - Support for multiple RT applications