MARACAS: A Real-Time Multicore VCPU Scheduling Framework

Ying Ye, Richard West, Jingyi Zhang, Zhuoqun Cheng

Computer Science Department
Boston University

Overview

Motivation

- Multicore platforms are gaining popularity in embedded and real-time systems
  - concurrent workload support
  - less circuit area
  - lower power consumption
  - lower cost

Motivation

- Multicore platforms are gaining popularity in embedded and real-time systems
  - concurrent workload support
  - less circuit area
  - lower power consumption
  - lower cost
- Complex on-chip memory hierarchies pose significant challenges for applications with real-time requirements

Motivation

- Shared cache contention:
  - page coloring
  - hardware cache partitioning (Intel CAT)
  - static VS dynamic

Motivation

- Shared cache contention:
  - page coloring
  - hardware cache partitioning (Intel CAT)
  - static VS dynamic
- Memory bus contention:
  - bank-aware memory management
  - memory throttling

Contribution

- We proposed the use of foreground (reservation) + background (surplus) scheduling model
  - improves application performance
  - effectively reduces resource contention
  - well-integrated with real-time scheduling algorithms

Contribution

- We proposed the use of foreground (reservation) + background (surplus) scheduling model
  - improves application performance
  - effectively reduces resource contention
  - well-integrated with real-time scheduling algorithms
- We proposed a new bus monitoring metric that accurately detects traffic

Application

- Imprecise computation/Numeric integration
  - MPEG video decoding: mandatory to process I-frames, optional to process B- and P-frames to improve frame rate

Application

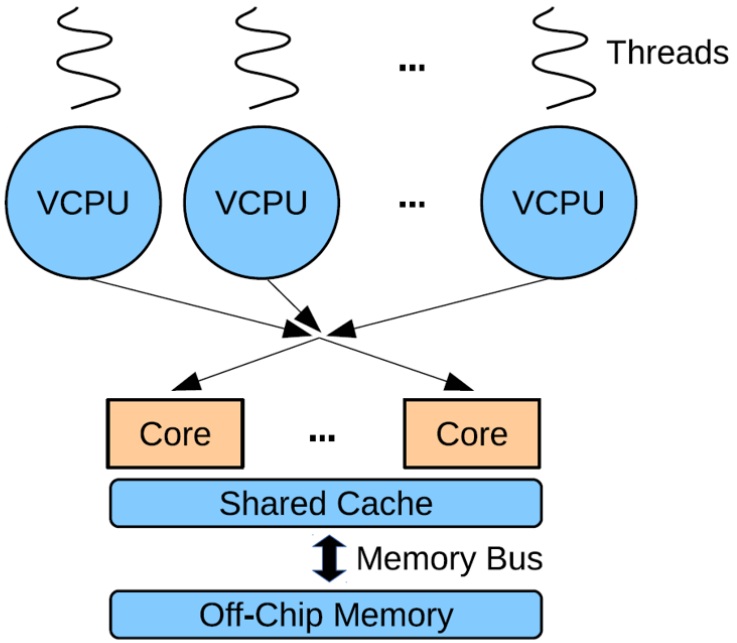
- Imprecise computation/Numeric integration
  - MPEG video decoding: mandatory to process I-frames, optional to process B- and P-frames to improve frame rate
- Mixed-criticality systems running performance-demanding applications
  - machine learning
  - computer vision

Quest RTOS

- VCPU model (C, T) in Quest RTOS
  - C: Capacity
  - T: Period

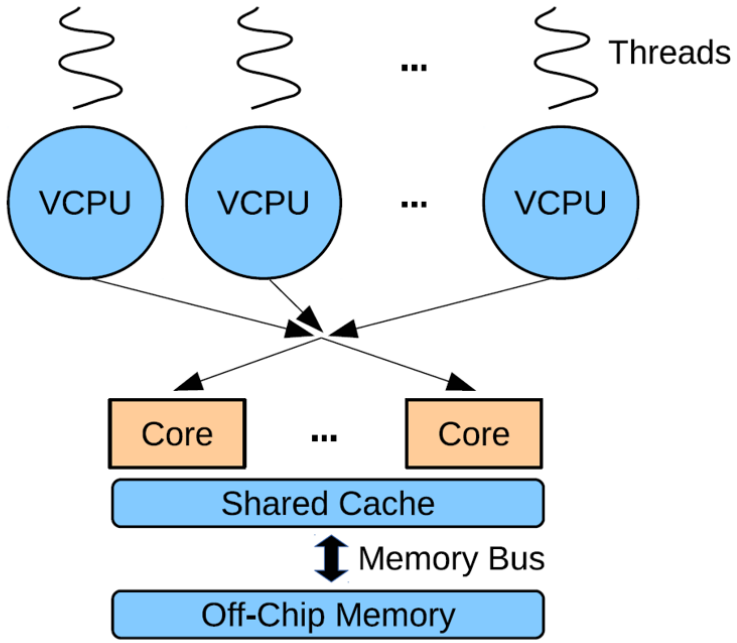
Quest RTOS

- VCPU model (C, T) in Quest RTOS
  - C: Capacity
  - T: Period
- Partitioned scheduling using RMS



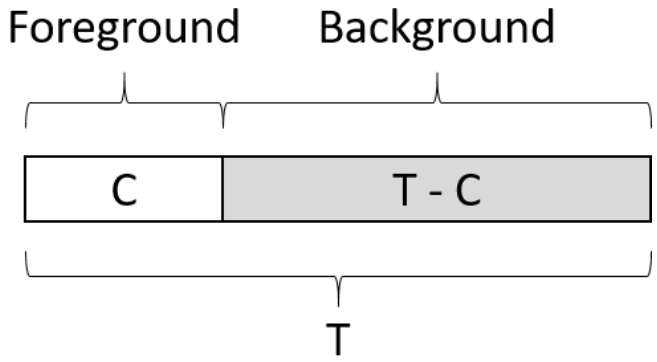
Quest RTOS

- VCPU model (C, T) in Quest RTOS
  - C: Capacity
  - T: Period
- Partitioned scheduling using RMS
- Schedulability test $\sum_1^n(\frac{C_i}{T_i}) \leq n(\sqrt[n]{2}-1)$



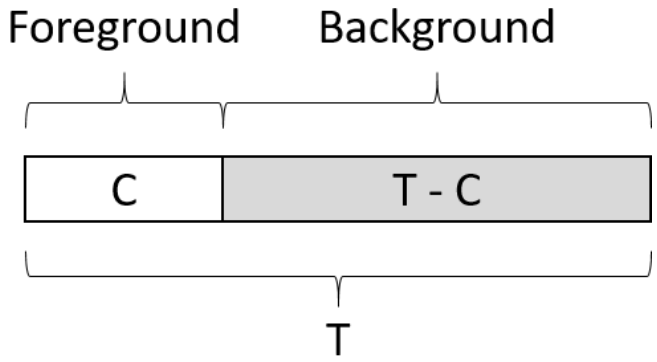
Background Scheduling

- VCPU enters background mode upon depleting its budget (C)



Background Scheduling

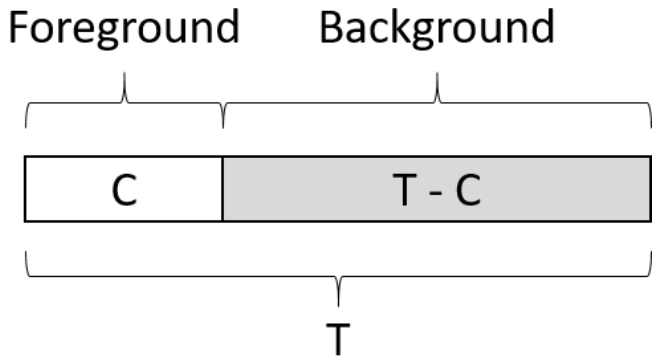
- VCPU enters background mode upon depleting its budget (C)



- Core enters background mode when all VCPUs are in background mode

Background Scheduling

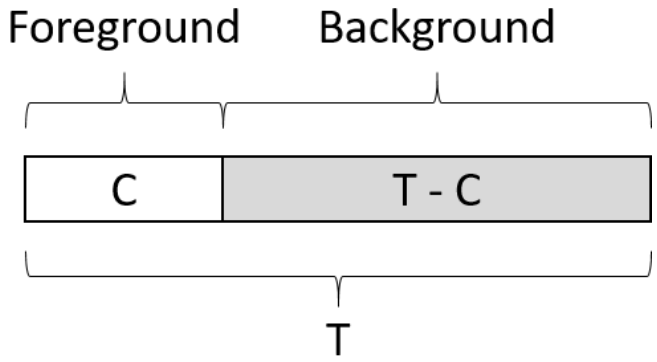
- VCPU enters background mode upon depleting its budget (C)



- Core enters background mode when all VCPUs are in background mode
- Background CPU Time (**BGT**): time a VCPU runs when core in background mode

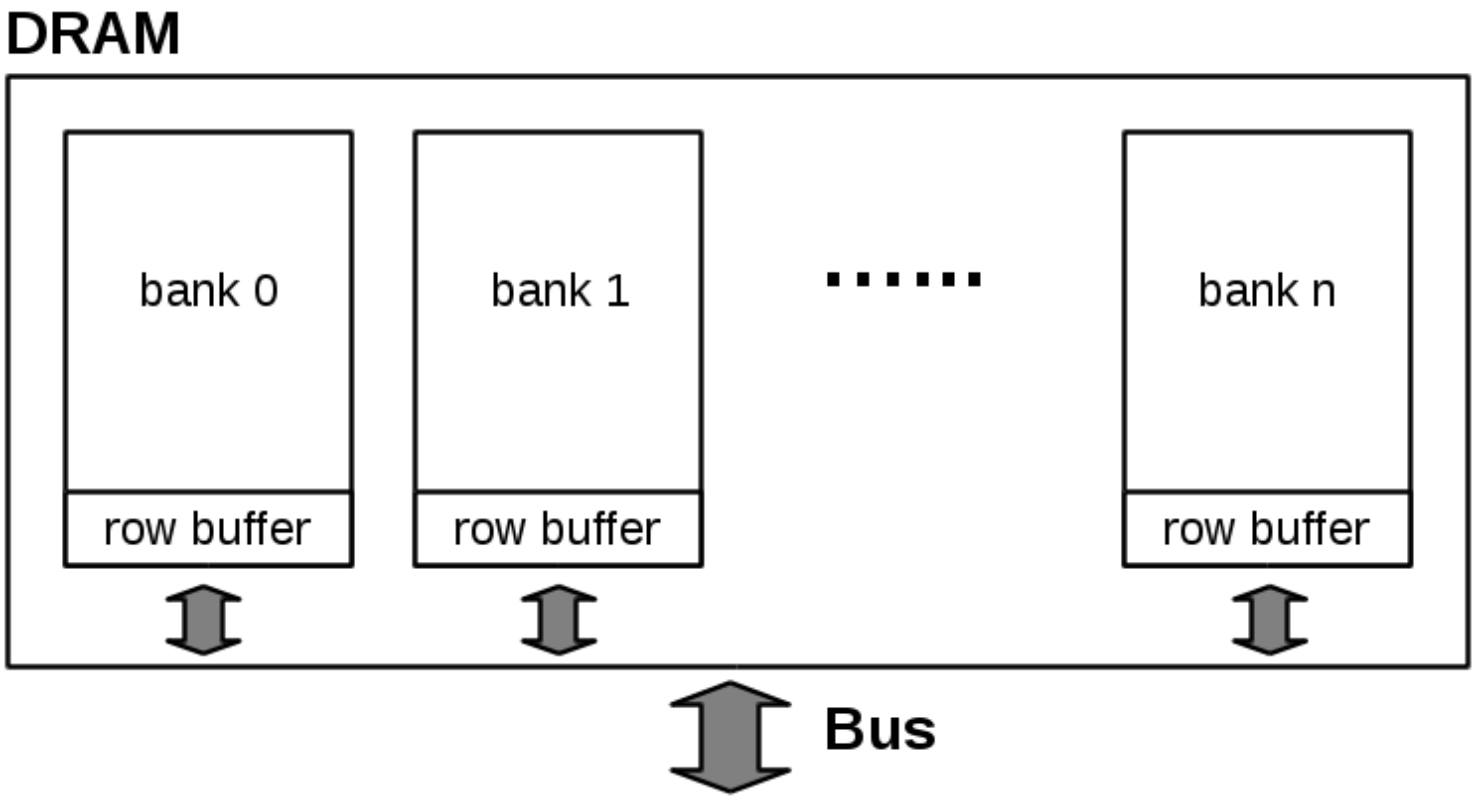
Background Scheduling

- VCPU enters background mode upon depleting its budget (C)



- Core enters background mode when all VCPUs are in background mode
- Background CPU Time (**BGT**): time a VCPU runs when core in background mode
- Background scheduling: schedule VCPUs when core is in background mode
  - fair share of **BGT** amongst VCPUs on core

DRAM structure



Memory-Aware Scheduling

- Prior work [MemGuard] uses "Rate Metric": number of DRAM accesses over a certain period

Memory-Aware Scheduling

- Prior work [MemGuard] uses "Rate Metric": number of DRAM accesses over a certain period
  - Bank-level parallelism

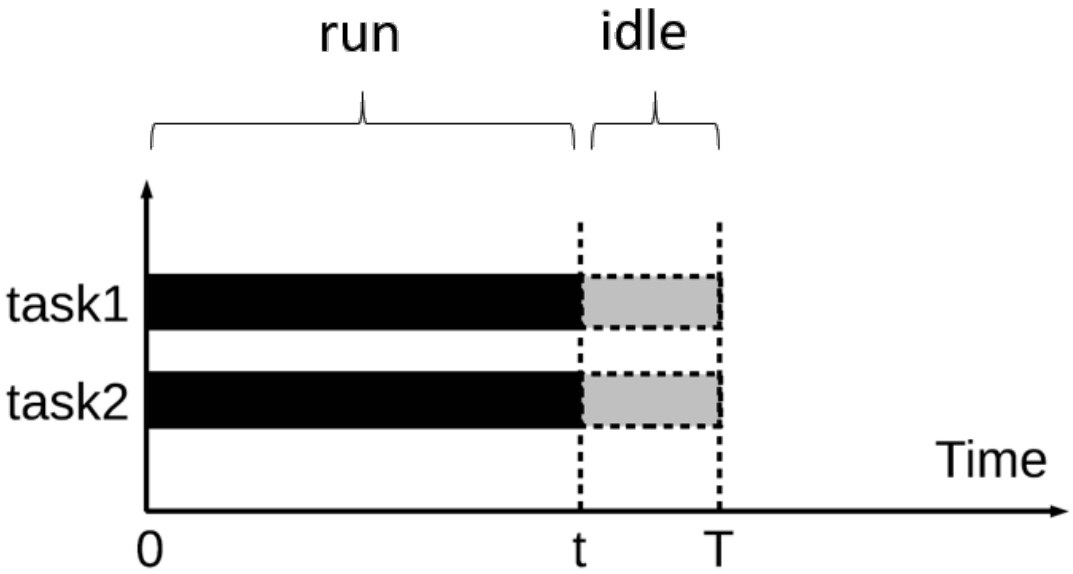
Memory-Aware Scheduling

- Prior work [MemGuard] uses "Rate Metric": number of DRAM accesses over a certain period
  - Bank-level parallelism
  - Row buffers

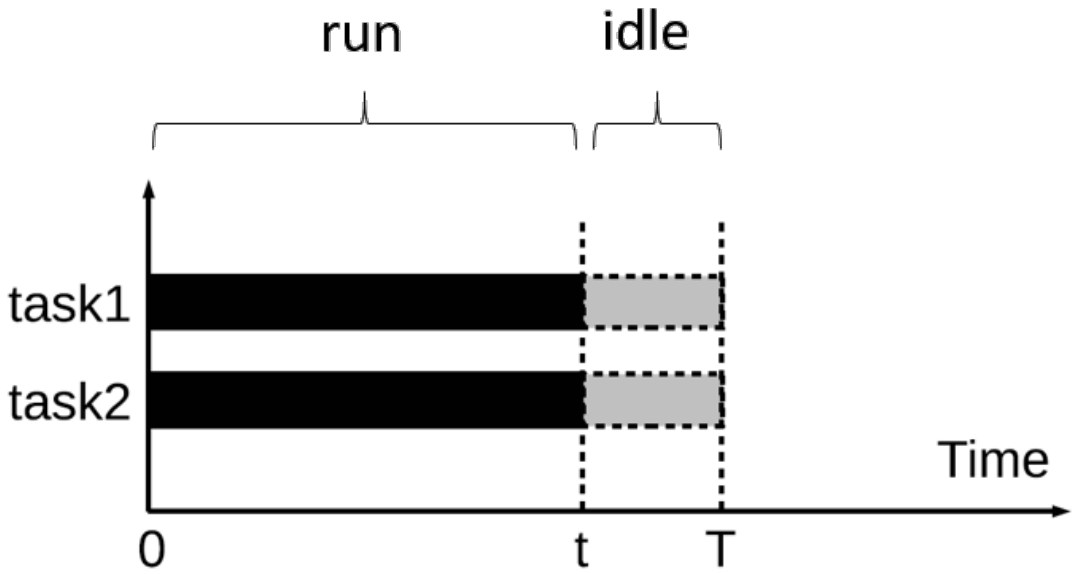
Memory-Aware Scheduling

- Prior work [MemGuard] uses "Rate Metric": number of DRAM accesses over a certain period
  - Bank-level parallelism
  - Row buffers
  - Sync Effect

Sync Effect

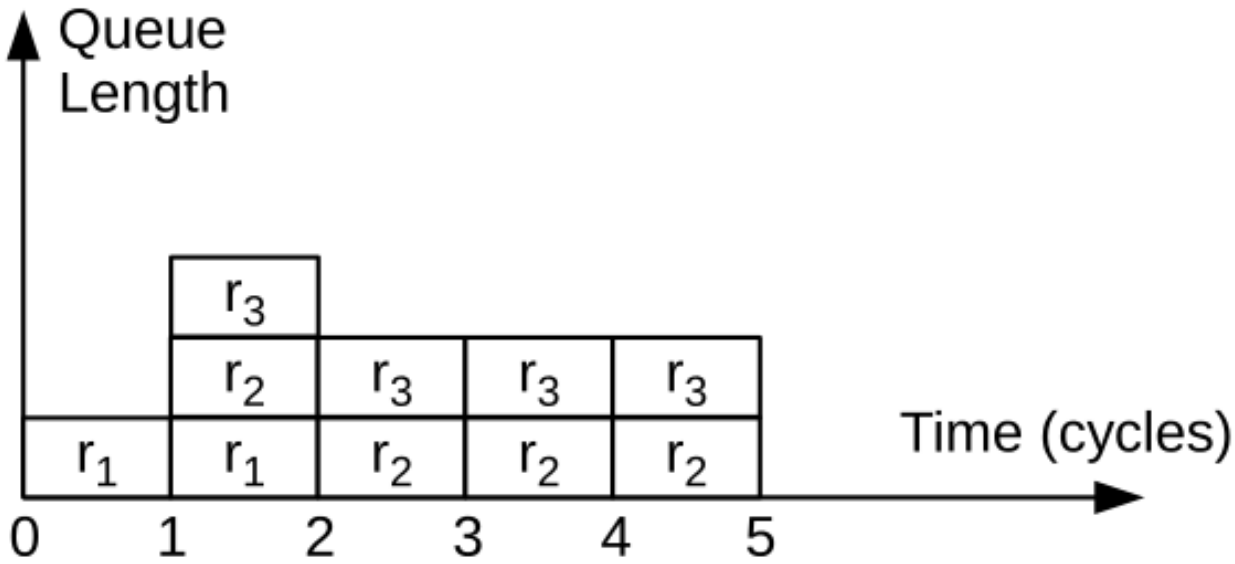


Sync Effect



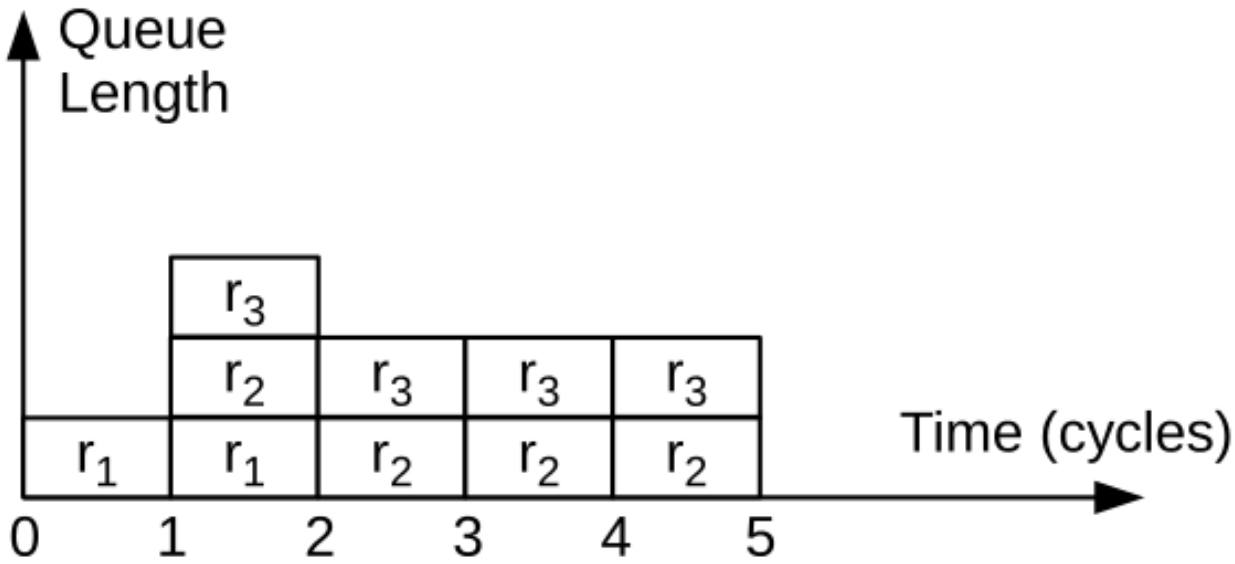
- Each task reduces its access rate by a factor of **(T-t)/T**
- Contention in [0, t] remains the same

Latency Metric



- $requests = 3,\ occupancy = 10$

Latency Metric



- $requests = 3,\ occupancy = 10$
- $latency = \frac{10}{3} = 3.3$

Latency Metric

- UNC_ARB_TRK_REQUEST.ALL (**requests**): counts all memory requests going to the memory controller request queue
- UNC_ARB_TRK_OCCUPANCY.ALL (**occupancy**): counts cycles weighted by the number of pending requests in the queue

Latency Metric

- UNC_ARB_TRK_REQUEST.ALL (**requests**): counts all memory requests going to the memory controller request queue
- UNC_ARB_TRK_OCCUPANCY.ALL (**occupancy**): counts cycles weighted by the number of pending requests in the queue
- Average latency: $latency = \frac{occupancy}{requests}$

Memory Throttling

- When core gets throttled, background scheduling is disabled

Memory Throttling

- When core gets throttled, background scheduling is disabled
- Latency threshold: MAX_MEM_LAT
  if $latency \geq MAX\_MEM\_LAT$ **then**
  $num\_throttle++$

Memory Throttling

- When core gets throttled, background scheduling is disabled
- Latency threshold: MAX_MEM_LAT
  if $latency \geq MAX\_MEM\_LAT$ **then**
  $num\_throttle++$
- Proportional throttling
  - Every core is throttled at some point
  - Throttled time proportional to core's DRAM access rate

Predictable Migration

Predictable Migration

- Run migration thread with highest priority on each core: pushing local VCPUs to other cores (starts from highest utilization ones)

Predictable Migration

- Run migration thread with highest priority on each core: pushing local VCPUs to other cores (starts from highest utilization ones)
- Only one migration thread active during a migration period

Predictable Migration

- Run migration thread with highest priority on each core: pushing local VCPUs to other cores (starts from highest utilization ones)
- Only one migration thread active during a migration period
- Its execution of its entire capacity C does not lead to any other local VCPUs missing their deadlines

Predictable Migration

- Run migration thread with highest priority on each core: pushing local VCPUs to other cores (starts from highest utilization ones)
- Only one migration thread active during a migration period
- Its execution of its entire capacity C does not lead to any other local VCPUs missing their deadlines
- Constraint on C:

$$C \geq 2 \times E_{lock} + E_{struct}$$

VCPU Load Balancing

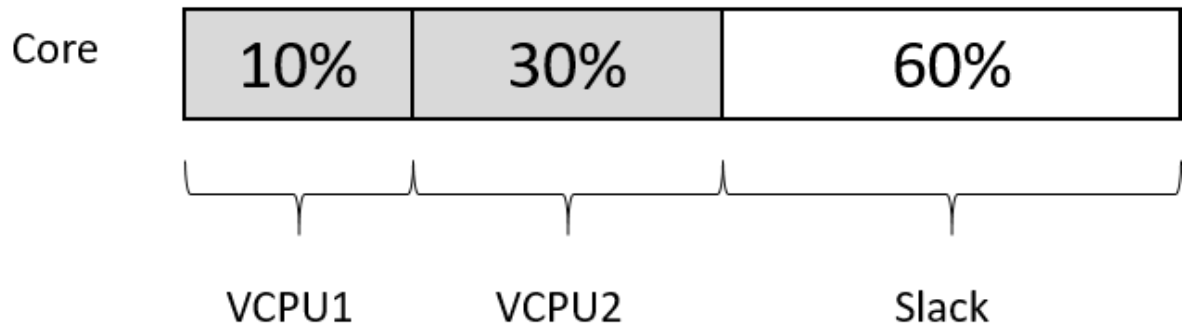
- For every core, define Slack-Per-VCPU (**SPV**):

$$SPV = \frac{1 - \sum_1^n (C_i / T_i)}{n}$$

VCPU Load Balancing

- For every core, define Slack-Per-VCPU (**SPV**):

$$SPV = \frac{1-\sum_1^n (C_i/T_i)}{n}$$



$$SPV = \frac{1-(10\%+30\%)}{2} = 30\%$$

VCPU Load Balancing

- Balance Background CPU Time (**BGT**) used by every VCPU across cores: equalize **SPV**s of all cores

VCPU Load Balancing

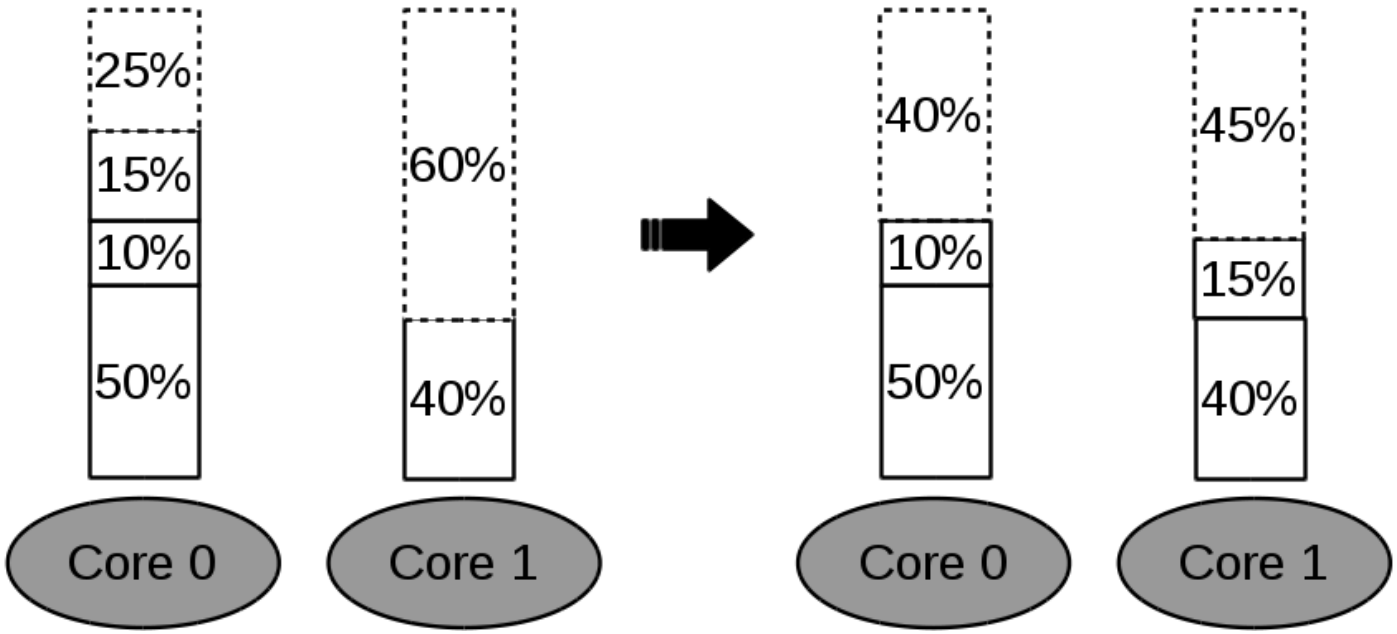
- Balance Background CPU Time (**BGT**) used by every VCPU across cores: equalize **SPV**s of all cores
  - **BGT** fair sharing

VCPU Load Balancing

- Balance Background CPU Time (**BGT**) used by every VCPU across cores: equalize **SPV**s of all cores
  - **BGT** fair sharing
  - balanced memory throttling capability on each core

VCPU Load Balancing

- Balance Background CPU Time (**BGT**) used by every VCPU across cores: equalize **SPV**s of all cores
  - **BGT** fair sharing
  - balanced memory throttling capability on each core



Cache-Aware Scheduling

- Static cache partitioning amongst cores
  - page coloring

Cache-Aware Scheduling

- Static cache partitioning amongst cores
  - page coloring
- New API:
  bool vcpu_create(uint C, uint T, uint cache);

Cache-Aware Scheduling

- Static cache partitioning amongst cores
  - page coloring
- New API:
  bool vcpu_create(uint C, uint T, uint cache);
- Extension of VCPU Load Balancing:
  destination core meets the cache requirement

Evaluation

- MARACAS running on the following hardware platform:

Processor	Intel Core i5-2500k quad-core
Caches	6MB L3 cache, 12-way set associative, 4 cache slices
Memory	8GB 1333MHz DDR3, 1 channel, 2 ranks, 8KB row buffers

Rate VS Latency

- Micro-benchmark m_jump:

  byte array[6M];
  for (uint32 j = 0; j < 8K; j += 64)
  for (uint32 i = j; i < 6M; i += 8K)
  *< **Variable delay added here** >*
  (uint32)array[i] = i;

Rate VS Latency

- Micro-benchmark m_jump:
  byte array[6M];
  for (uint32 j = 0; j < 8K; j += 64)
  for (uint32 i = j; i < 6M; i += 8K)
  *< **Variable delay added here** >*
  (uint32)array[i] = i;

- Three m_jump (task 1,2,3) running on separate cores without memory throttling, utilization (C/T) 50%

Rate VS Latency

- Micro-benchmark m_jump:
  byte array[6M];
  for (uint32 j = 0; j < 8K; j += 64)
  for (uint32 i = j; i < 6M; i += 8K)
  *< **Variable delay added here** >*
  (uint32)array[i] = i;

- Three m_jump (task 1,2,3) running on separate cores without memory throttling, utilization (C/T) 50%

- Each run, insert a different time delay in task1 and task2, task3 has no delay

Rate VS Latency

- Record the total memory bus traffic, average memory request latency and task3's instructions retired in foreground mode:

Rate VS Latency

- Record the total memory bus traffic, average memory request latency and task3's instructions retired in foreground mode:

	Bus Traffic (GB)	Latency	*task3* Instructions Retired ($\times 10^8$)
H	1128	228	249
M	1049	183	304
L	976	157	357

Rate VS Latency

- Record the total memory bus traffic, average memory request latency and task3's instructions retired in foreground mode:

	Bus Traffic (GB)	Latency	*task3* Instructions Retired ($\times 10^8$)
H	1128	228	249
M	1049	183	304
L	976	157	357

- Setting comparable thresholds:
  - rate-based: derived from Bus Traffic (1128/time)
  - latency-based: from Latency (228)

Rate VS Latency

- Record the total memory bus traffic, average memory request latency and task3's instructions retired in foreground mode:

	Bus Traffic (GB)	Latency	*task3* Instructions Retired ($\times 10^8$)
H	1128	228	249
M	1049	183	304
L	976	157	357

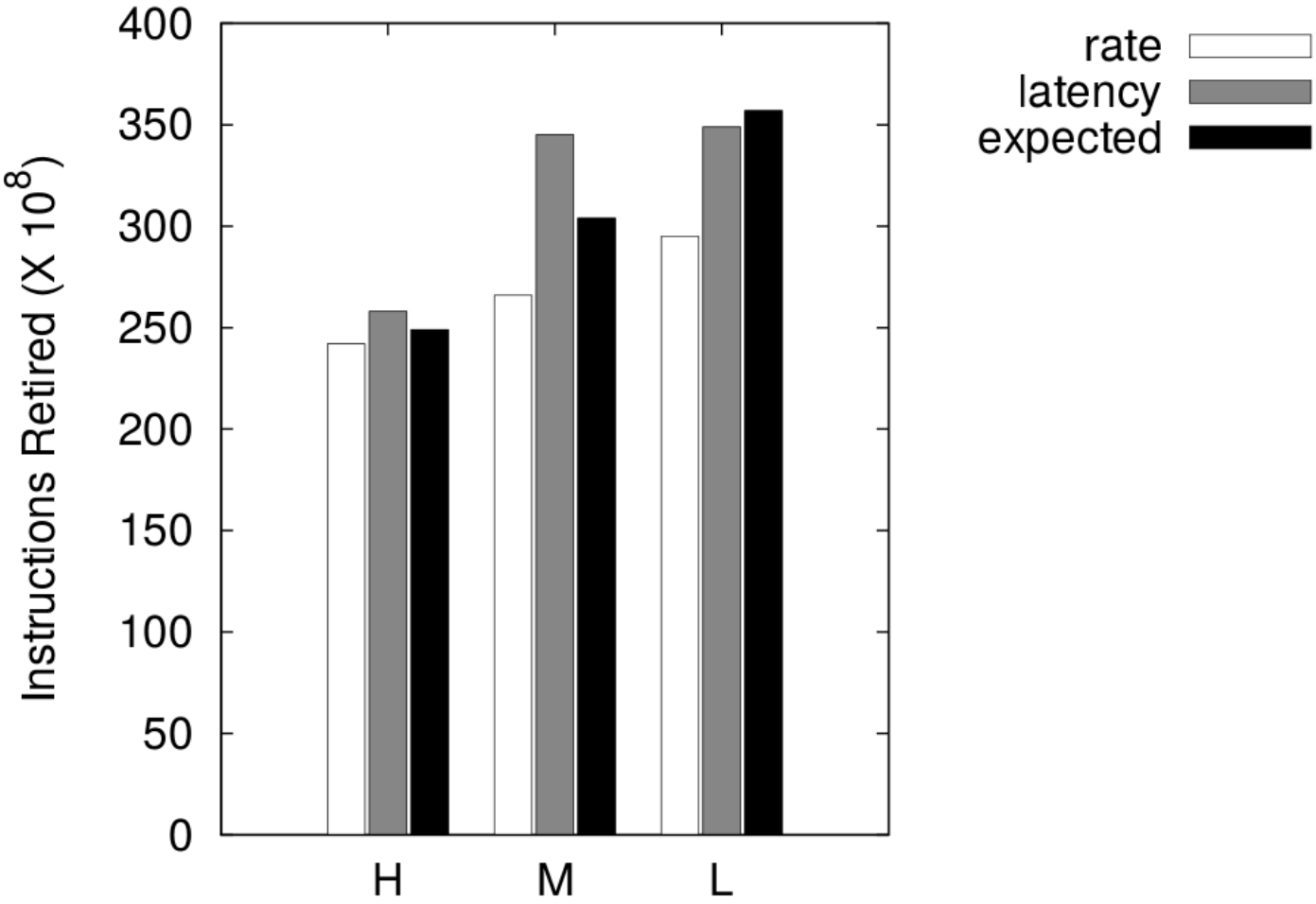
- Setting comparable thresholds:
  - rate-based: derived from Bus Traffic (1128/time)
  - latency-based: from Latency (228)
- Last column serves as reference, showing the **expected** performance of task3 using the corresponding thresholds

Rate VS Latency

- Repeat experiment with memory throttling enabled and fixed delay for task1/task2

Rate VS Latency

- Repeat experiment with memory throttling enabled and fixed delay for task1/task2



Conclusion

- MARACAS uses background time to improve task performance; when memory bus is contended, it gets disabled through throttling

Conclusion

- MARACAS uses background time to improve task performance; when memory bus is contended, it gets disabled through throttling
- MARACAS uses a latency metric to trigger throttling, outperforming prior rate-based approach

Conclusion

- MARACAS uses background time to improve task performance; when memory bus is contended, it gets disabled through throttling
- MARACAS uses a latency metric to trigger throttling, outperforming prior rate-based approach
- MARACAS fairly distributes background time across cores, for both fairness and better throttling

Thank you!

Ying Ye

Boston University