PACT 2013
The 22nd International Conference on Parallel Architectures and Compilation Techniques

L1-Bandwidth Aware Thread Allocation in Multicore SMT Processors

J. Feliu, J. Sahuquillo, S. Petit and J. Duato

Universitat Politècnica de València

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

9 September 2013
Edinburgh, United Kingdom

Outline

- Introduction
- Experimental platform
- Effects of L1 bandwidth on performance of SMT processors
- L1-bandwidth aware thread allocation policies
- Evaluation methodology
- Performance evaluation results
- Conclusions

Outline

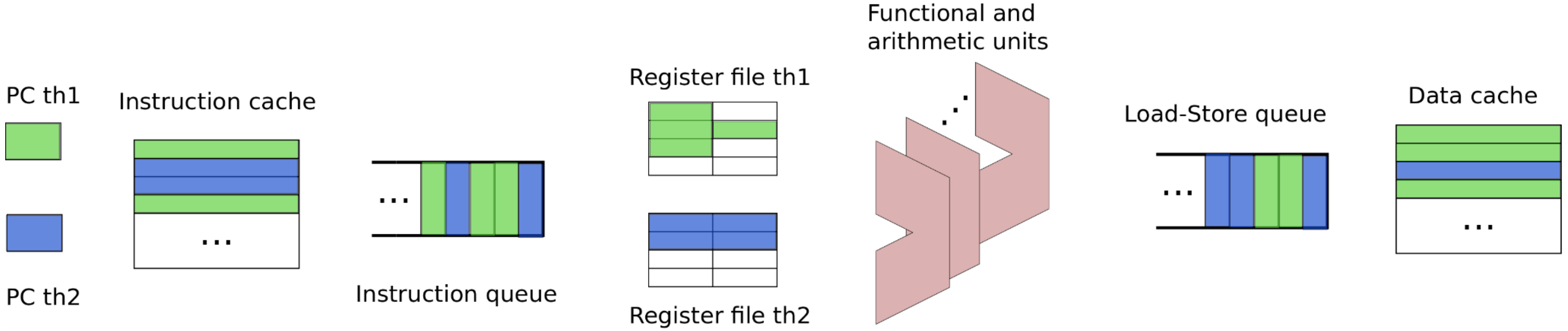
- Introduction
- Experimental platform
- Effects of L1 bandwidth on performance of SMT processors
- L1-bandwidth aware thread allocation policies
- Evaluation methodology
- Performance evaluation results
- Conclusions

Introduction

- Simultaneous multithreading (SMT) processors exploit:
  - Instruction-level parallelism
  - Thread-level parallelism

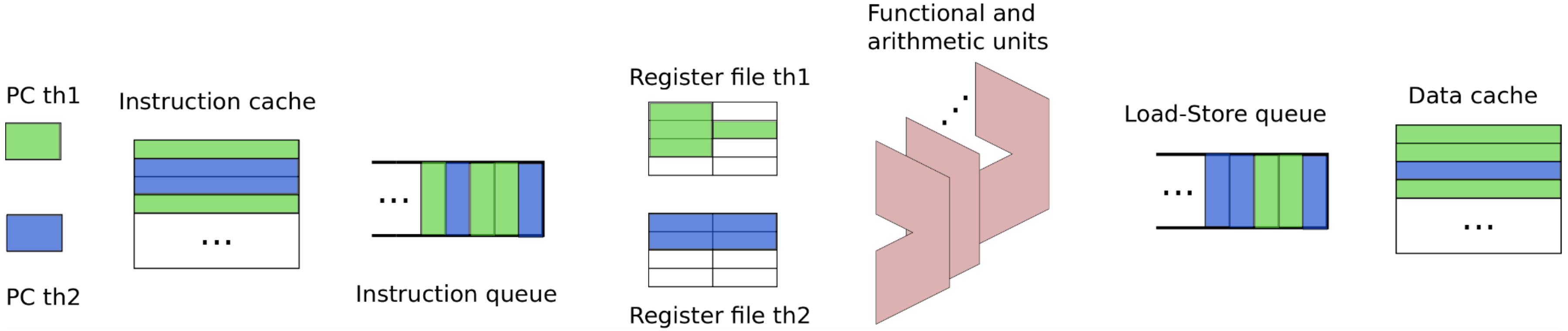
Introduction

- Simultaneous multithreading (SMT) processors exploit:
  - Instruction-level parallelism
  - Thread-level parallelism
- **Threads are continuously sharing some processor resources**



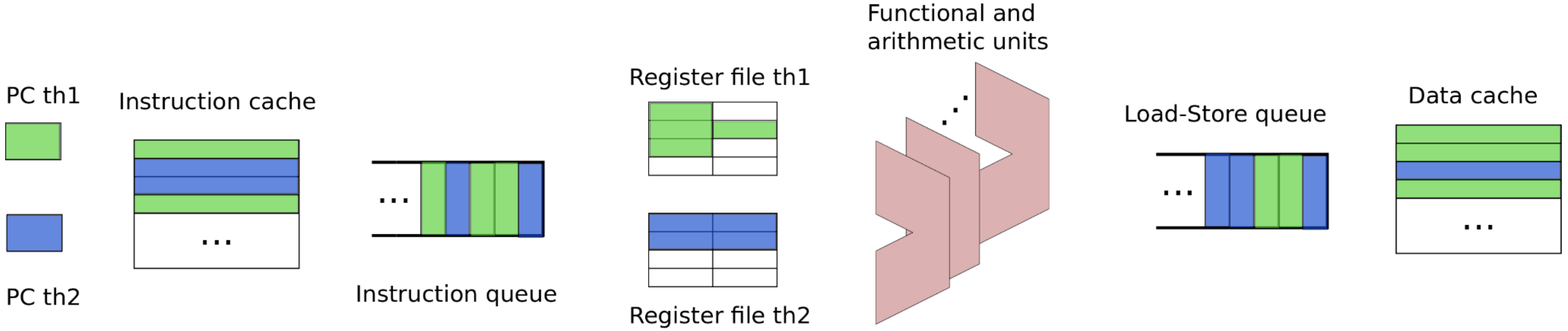
Introduction

- Simultaneous multithreading (SMT) processors exploit:
  - Instruction-level parallelism
  - Thread-level parallelism
- **Threads are continuously sharing some processor resources**
  - If the demand of a shared resource exceeds what it can provide
    →Performance can be damaged



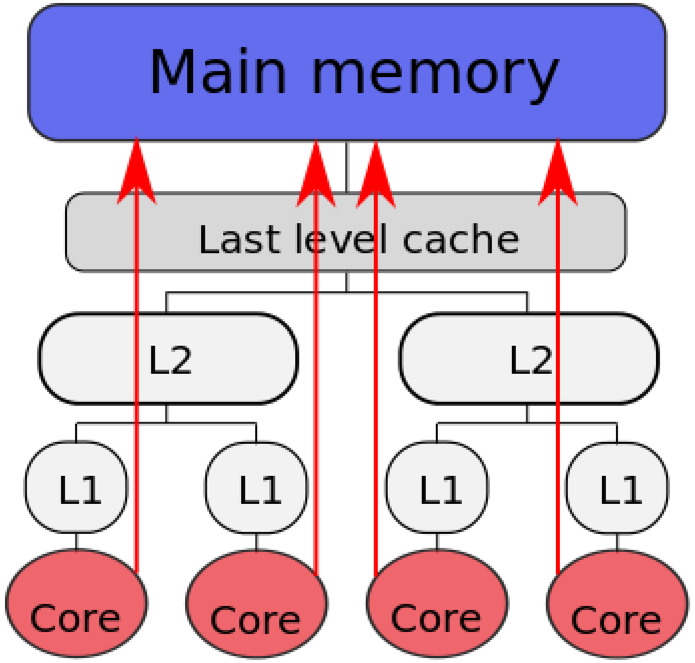
Introduction

- Simultaneous multithreading (SMT) processors exploit:
  - Instruction-level parallelism
  - Thread-level parallelism
- **Threads are continuously sharing some processor resources**
  - If the demand of a shared resource exceeds what it can provide →**Performance can be damaged**
- Smart thread to core mapping policies can help to alleviate the contention in the shared resources



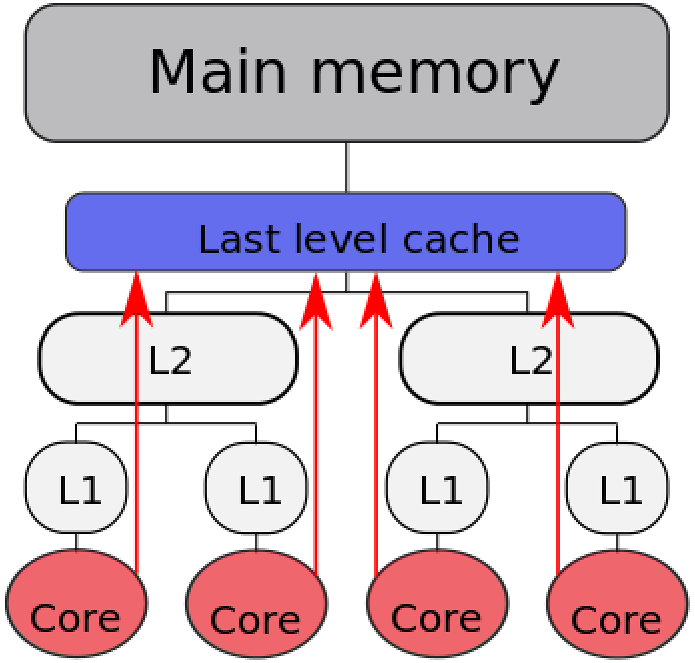
Introduction

- A critical shared resource in any CMP is the memory bandwidth
    - Main memory bandwidth



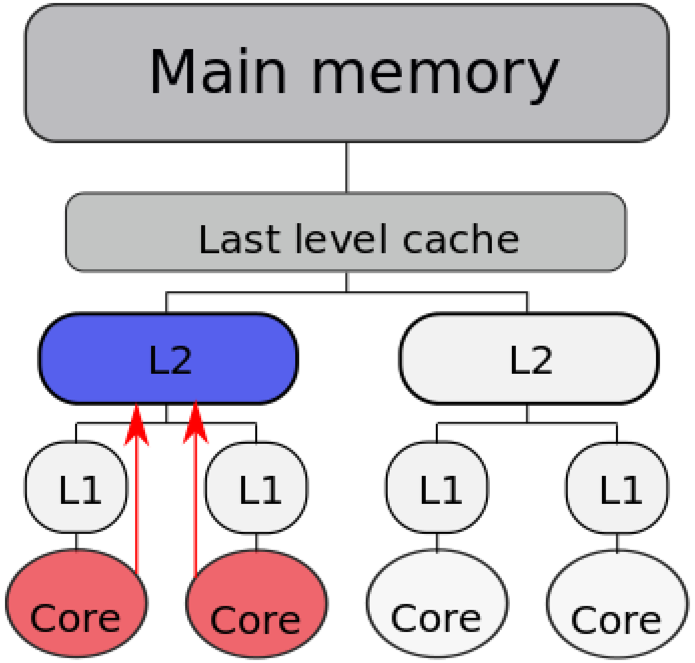
Introduction

- A critical shared resource in any CMP is the memory bandwidth
  - Main memory bandwidth
  - LLC bandwidth



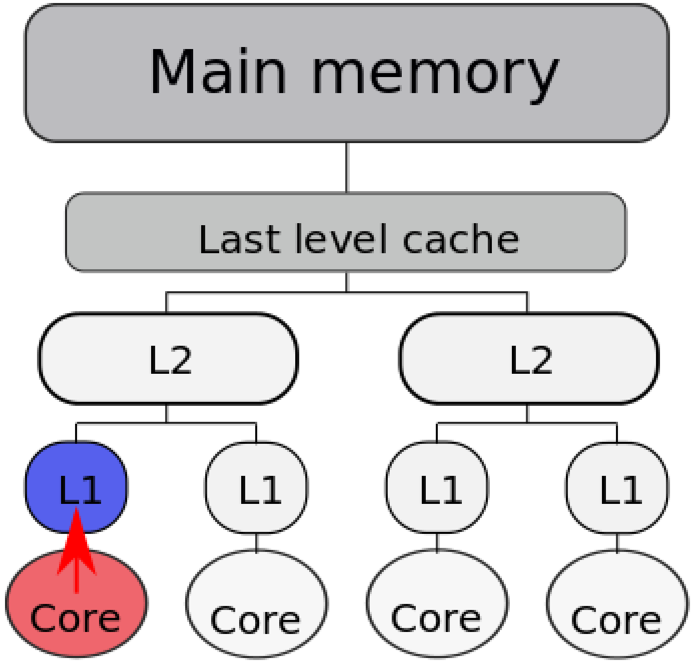
Introduction

- A critical shared resource in any CMP is the memory bandwidth
  - Main memory bandwidth
  - LLC bandwidth
  - Bandwidth at any shared cache



Introduction

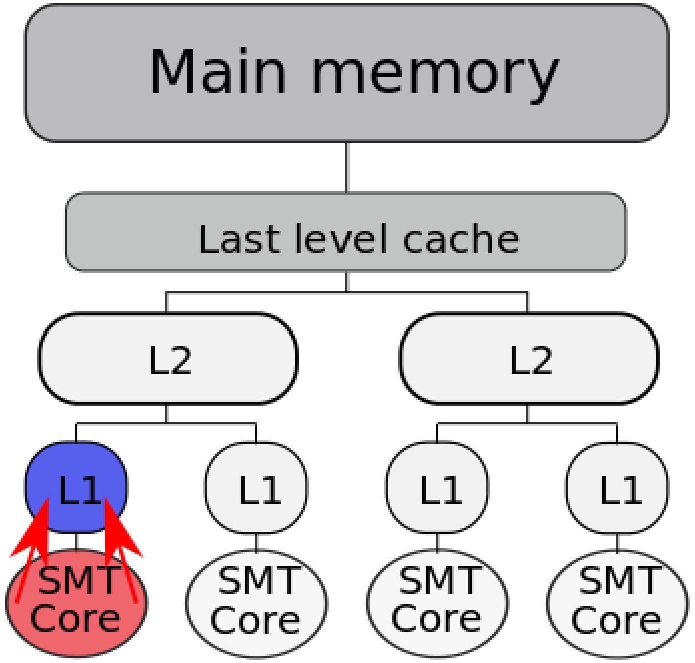
- A critical shared resource in any CMP is the memory bandwidth
    - Main memory bandwidth
    - LLC bandwidth
    - Bandwidth at any shared cache
- Addressed with bandwidth-aware schedulers
    - L1 caches are private to cores, and thus they have not been considered yet



Introduction

- A critical shared resource in any CMP is the memory bandwidth
    - Main memory bandwidth
    - LLC bandwidth
    - Bandwidth at any shared cache
- Addressed with bandwidth-aware schedulers
    - L1 caches are private to cores, and thus they have not been considered yet
- When the cores are SMT, the thread must share the L1 cache

  → L1 bandwidth contention may impact the performance



Introduction

Contributions

- Analysis of the connection between the L1 bandwidth and performance of the processes
    - Strong connection between the L1 bandwidth consumption and performance

Introduction

Contributions

- Analysis of the connection between the L1 bandwidth and performance of the processes
  - Strong connection between the L1 bandwidth consumption and performance

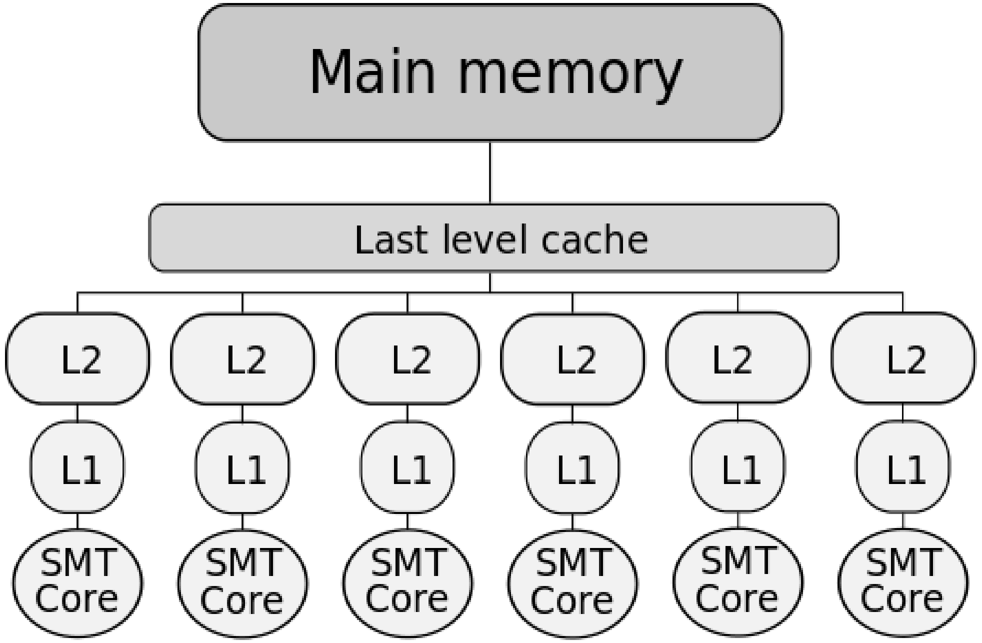
- Thread allocation strategies to deal with L1 bandwidth contention

Outline

- Introduction
- Experimental platform
- Effects of L1 bandwidth on performance of SMT processors
- L1-bandwidth aware thread allocation policies
- Evaluation methodology
- Performance evaluation results
- Conclusions

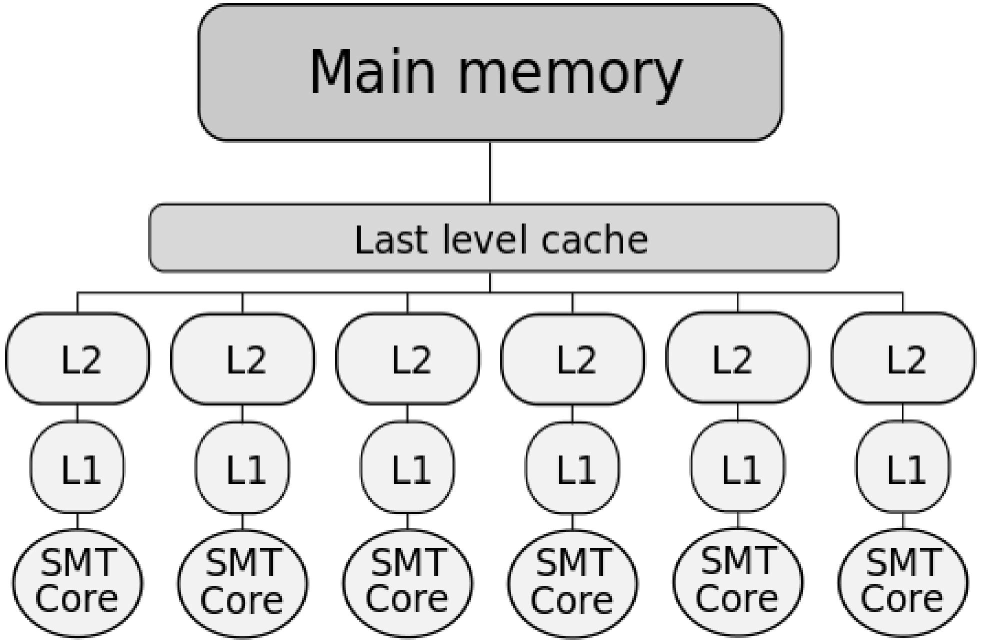
Experimental platform

- Experiments carried out in a Intel Xeon E5645
  - 6 dual-thread cores
  - Private L1 (32 KB x 6) and L2 (256 KB x 6) caches
  - Shared 12 MB LLC



Experimental platform

- Experiments carried out in a Intel Xeon E5645
  - 6 dual-thread cores
  - Private L1 (32 KB x 6) and L2 (256 KB x 6) caches
  - Shared 12 MB LLC
- Linux with kernel 3.3.0
- *Libpfm* 4.3 is used to manage performance counters
  - L1 requests, instructions and cycles for each running process are gathered dynamically
- SPEC CPU2006 benchmarks with reference inputs



Outline

- Introduction
- Experimental platform
- Effects of L1 bandwidth on performance of SMT processors
- L1-bandwidth aware thread allocation policies
- Evaluation methodology
- Performance evaluation results
- Conclusions

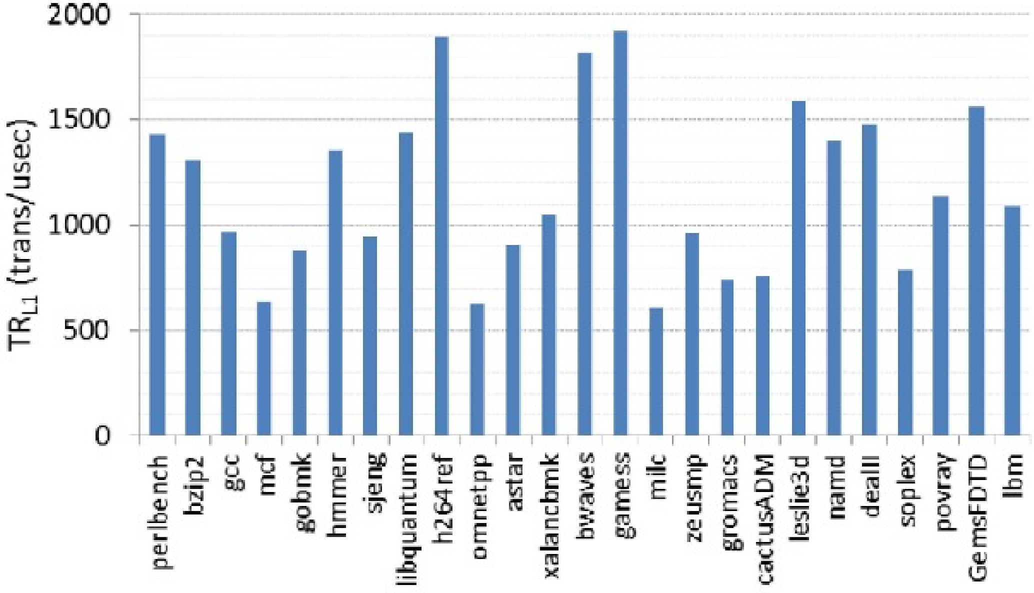
Effects of L1 bandwidth on performance of SMT processors

- Stand-alone execution
  - Average
    - Average TR_{L1} and IPC in stand-alone execution
  - Dynamic
- Concurrent execution

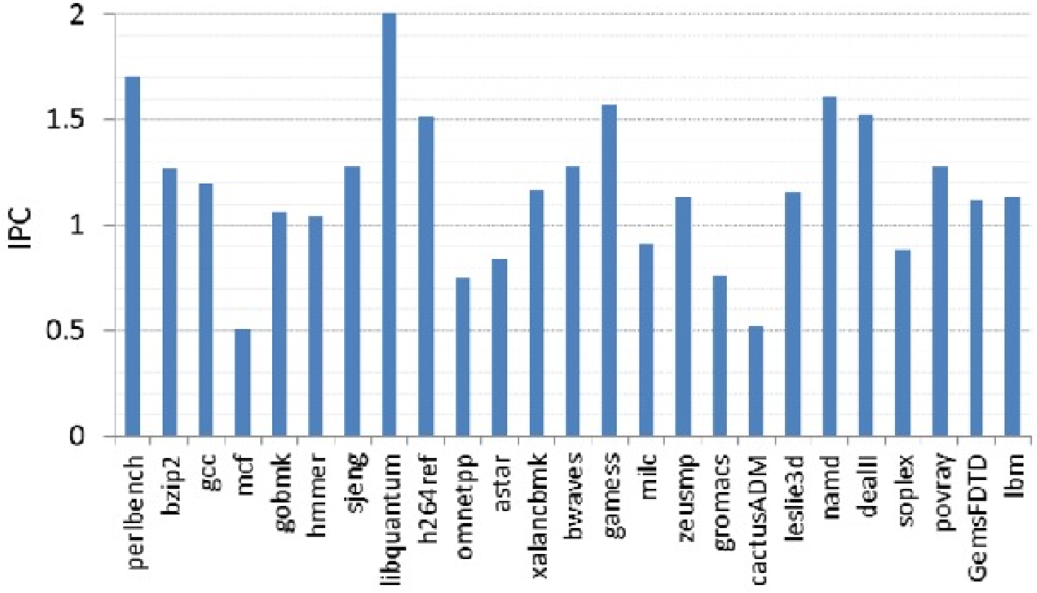
Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Average values

L1 bandwidth – TR_{L1}



Performance – IPC

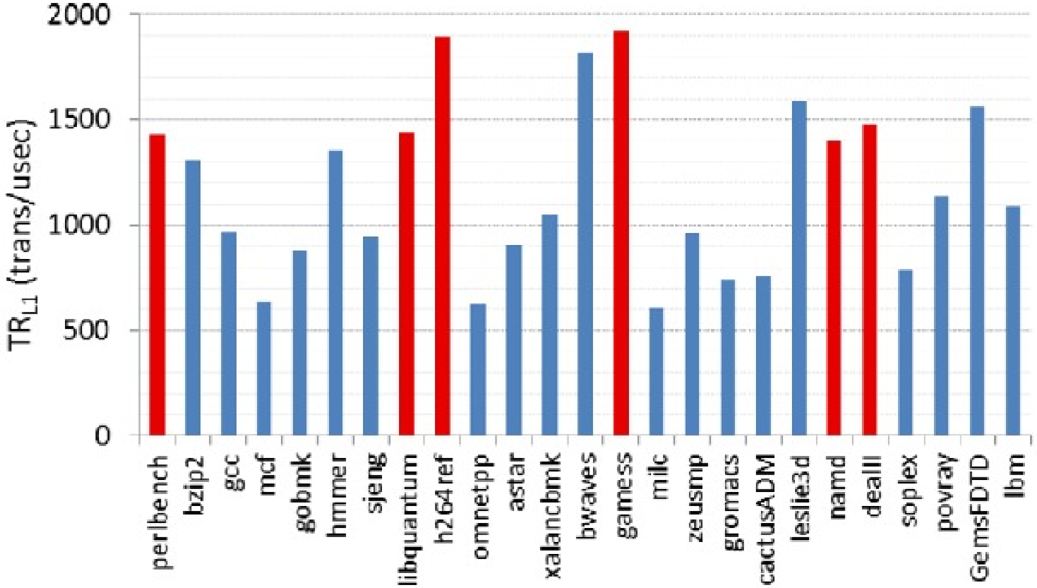


- Certain correlation between both metrics

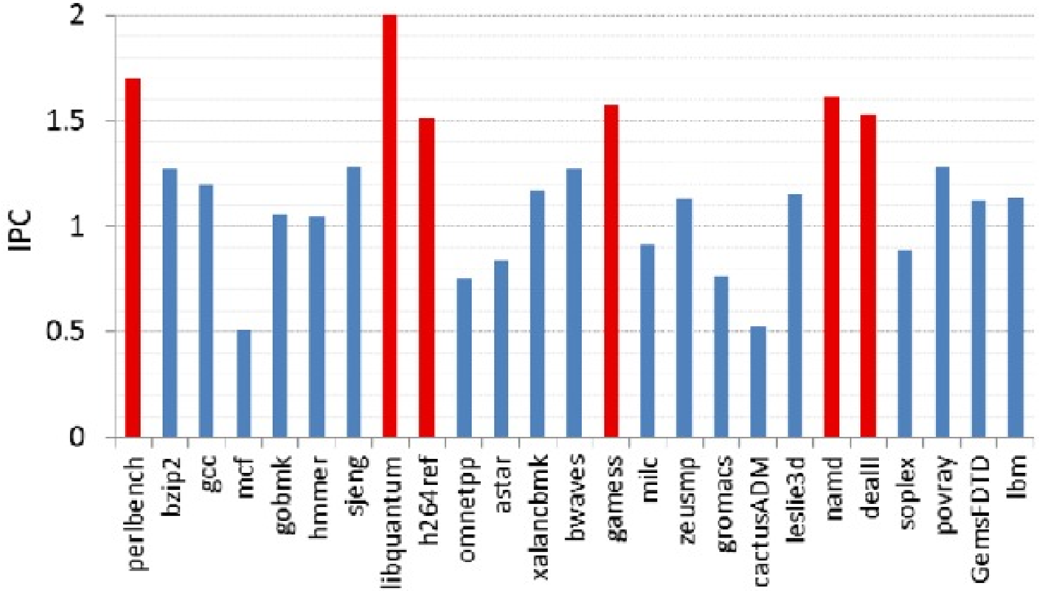
Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Average values

L1 bandwidth – TR $_{L1}$



Performance – IPC

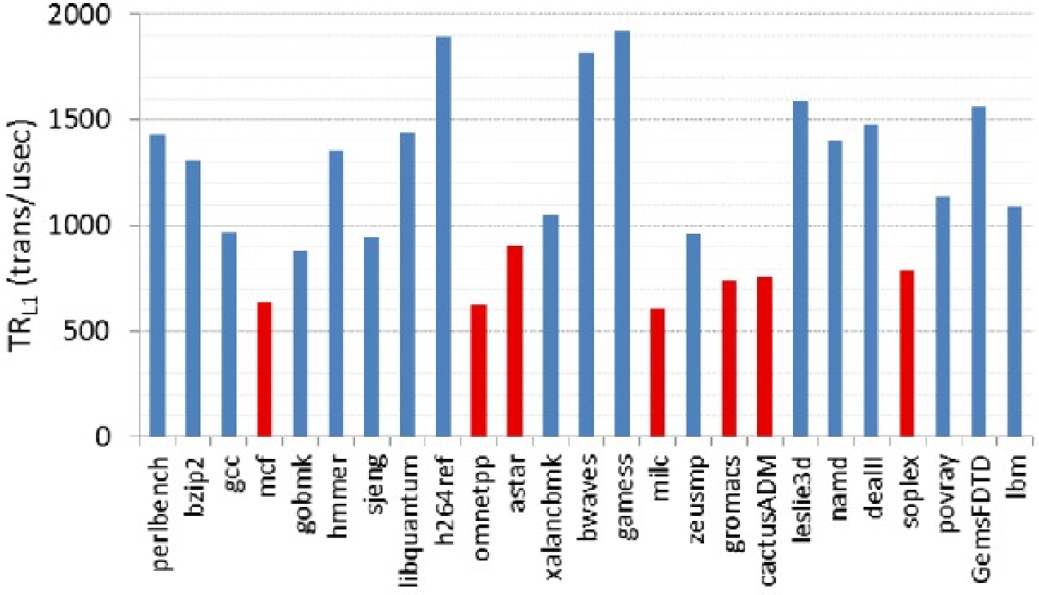


- Certain correlation between both metrics
  - Benchmarks with high TR$_{L1}$ present high IPC

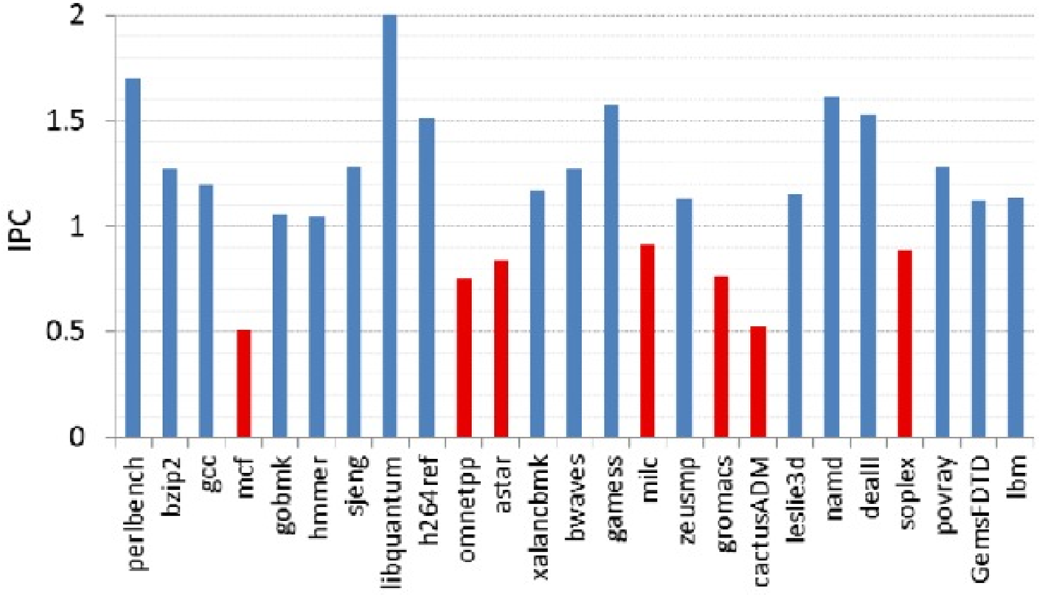
Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Average values

L1 bandwidth – TR $_{L1}$



Performance – IPC

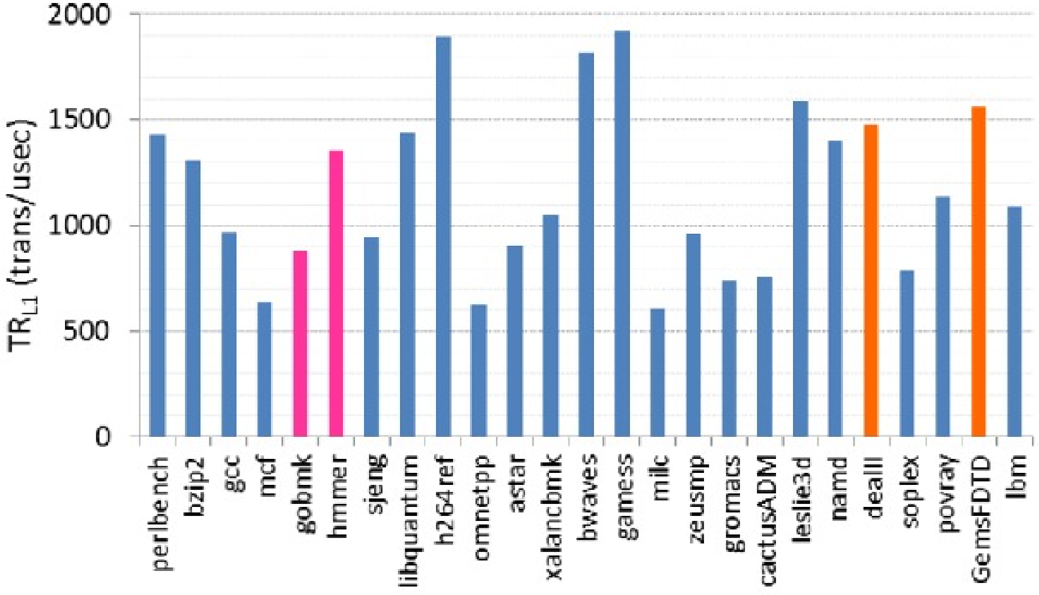


- Certain correlation between both metrics
  - Benchmarks with high TR$_{L1}$ present high IPC
  - Benchmarks with low TR$_{L1}$ present low IPC

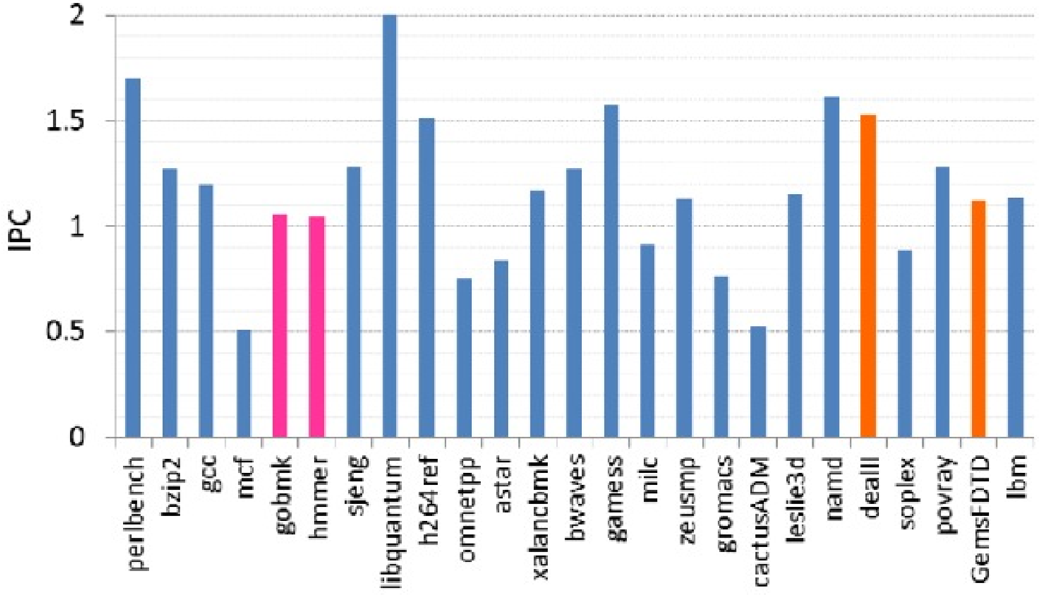
Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Average values

L1 bandwidth – TR_{L1}



Performance – IPC



- Certain correlation between both metrics
  - Benchmarks with high TR_{L1} present high IPC
  - Benchmarks with low TR_{L1} present low IPC
- Benchmarks with similar TR_{L1} (or IPC) can also show different TR_{L1} (or IPC)

Effects of L1 bandwidth on performance of SMT processors

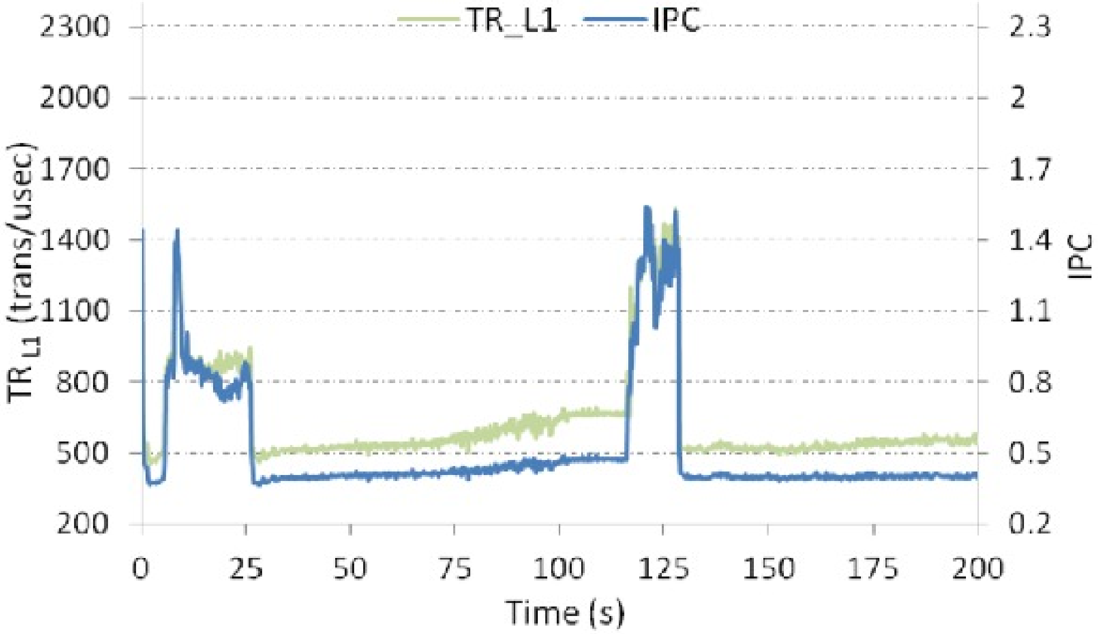
- Stand-alone execution
  - Average
    - Certain similarities appear among average values of L1 bandwidth and IPC, but there is no clear evidence about the connection between them
  - Dynamic
- Concurrent execution

Effects of L1 bandwidth on performance of SMT processors

- Stand-alone execution
  - Average
    - Certain similarities appear among average values of L1 bandwidth and IPC, but there is no clear evidence about the connection between them
  - Dynamic
    - The process behavior can widely vary during the execution, so lets analyze the dynamic value of both metrics
- Concurrent execution

Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Dynamic values

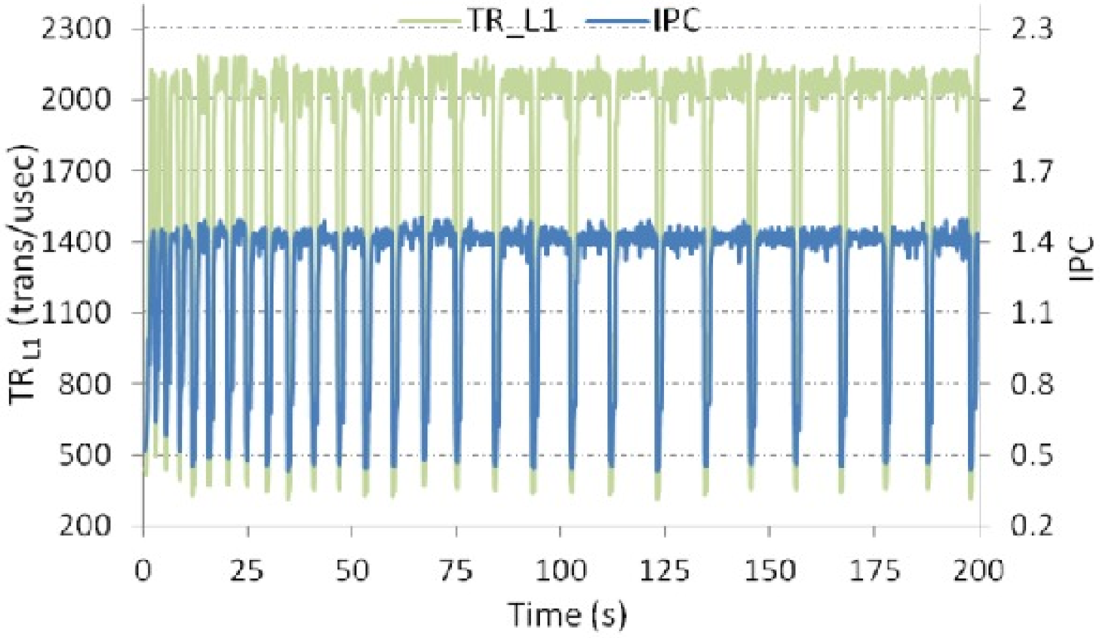


TR_{L1} and IPC evolution with time for ***mcf***

- The plot presents
  - L1 bandwidth
  - IPC
- Strong connection between L1 bandwidth and IPC dynamically

Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Dynamic values

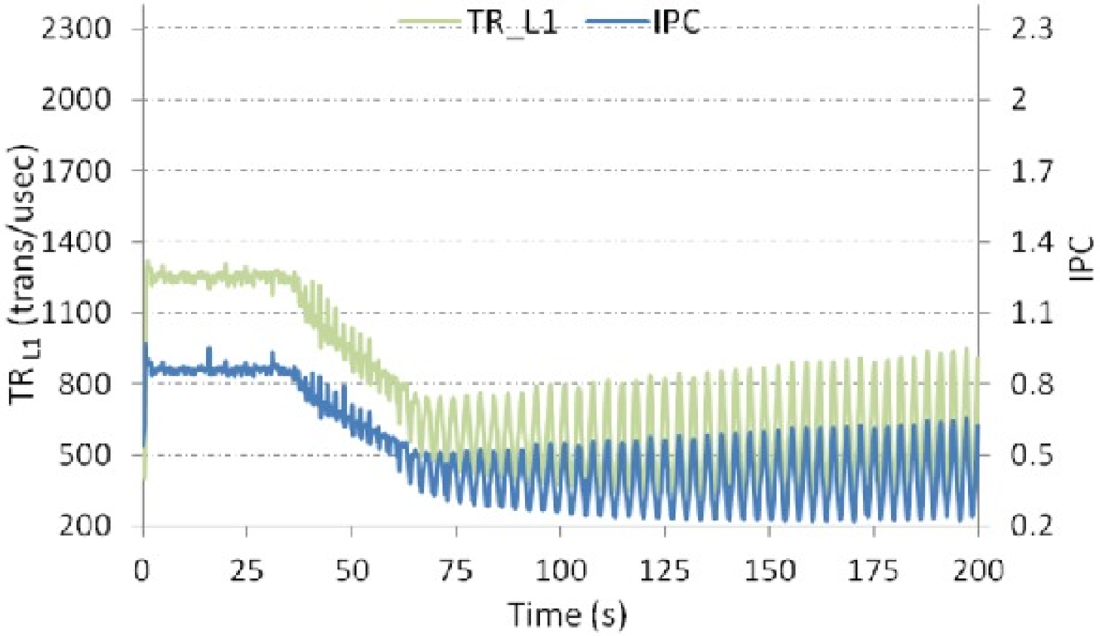


TR_{L1} and IPC evolution with time for ***bwaves***

- The plot presents
  - L1 bandwidth
  - IPC
- Strong connection between L1 bandwidth and IPC dynamically
  - **Almost identical shape**

Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Dynamic values

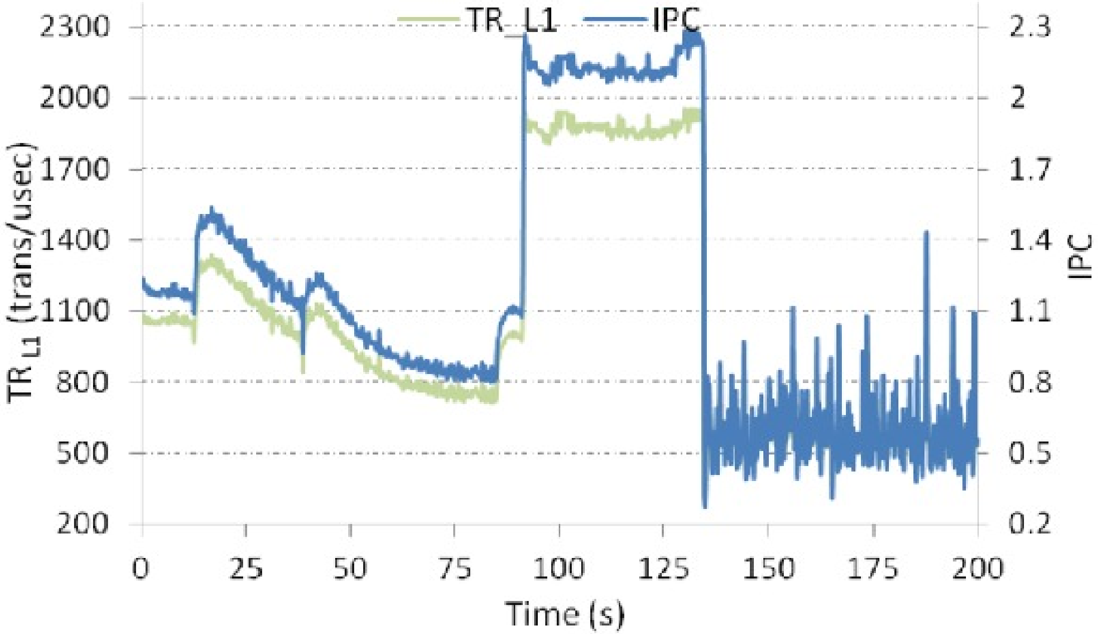


TR_{L1} and IPC evolution with time for ***cactusADM***

- The plot presents
  - L1 bandwidth
  - IPC
- Strong connection between L1 bandwidth and IPC dynamically
  - **Almost identical shape**
  - **Synchronized rises and drops with similar magnitude**

Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Dynamic values

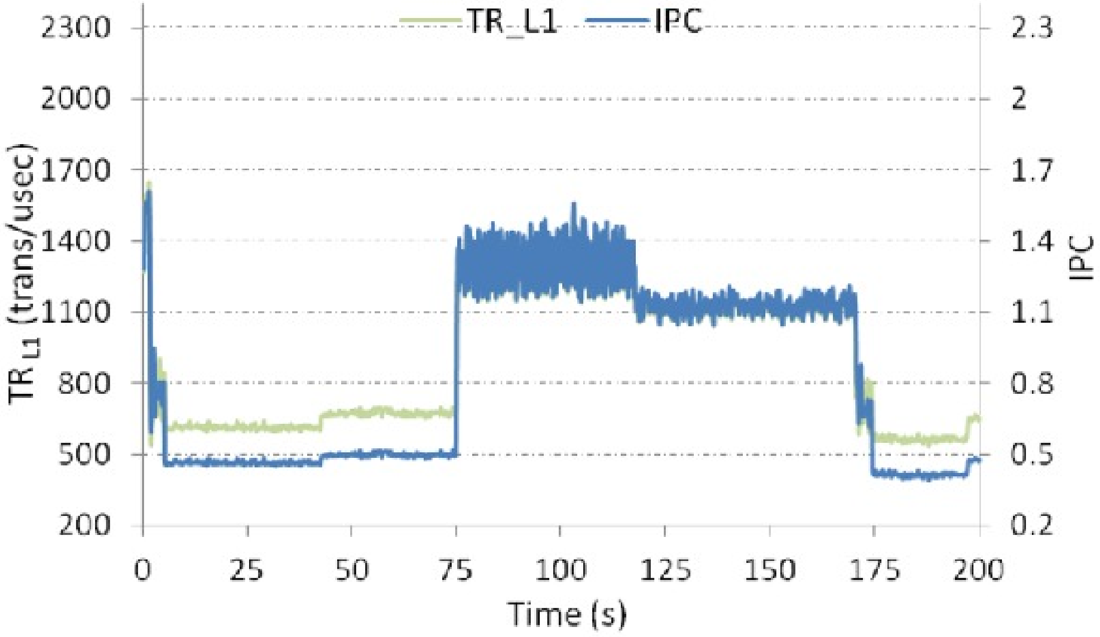


TR_{L1} and IPC evolution with time for ***xalancbmk***

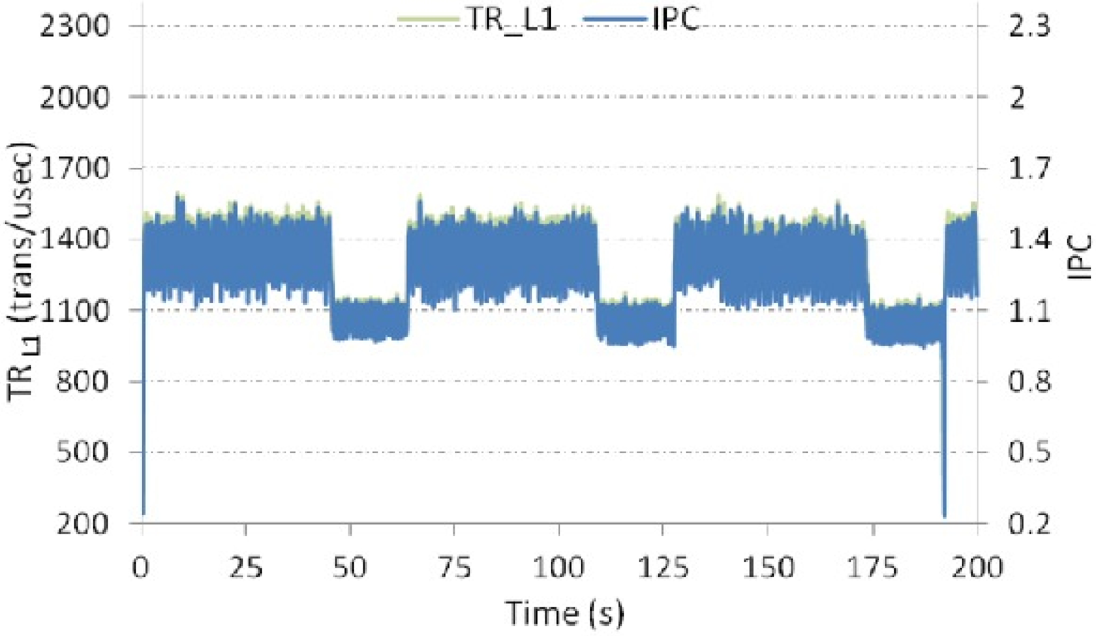
- The plot presents
  - L1 bandwidth
  - IPC
- Strong connection between L1 bandwidth and IPC dynamically
  - **Almost identical shape**
  - **Synchronized rises and drops with similar magnitude**
  - Even small peaks in L1 bandwidth trigger synchronized peaks in the IPC

Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Dynamic values



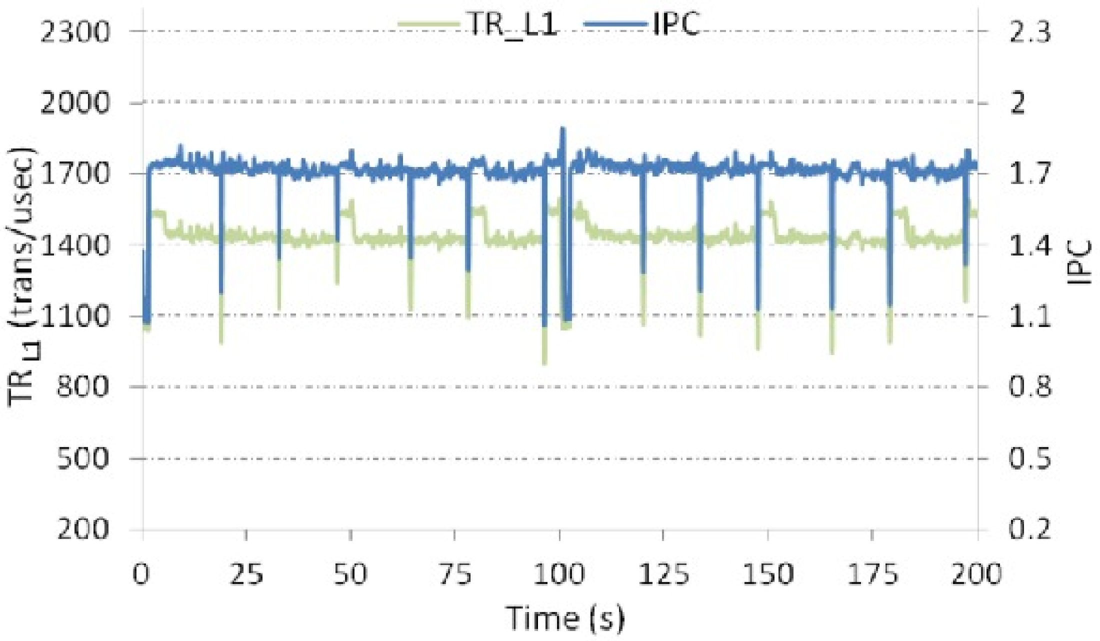
TR$_{L1}$ and IPC evolution with time for ***astar***



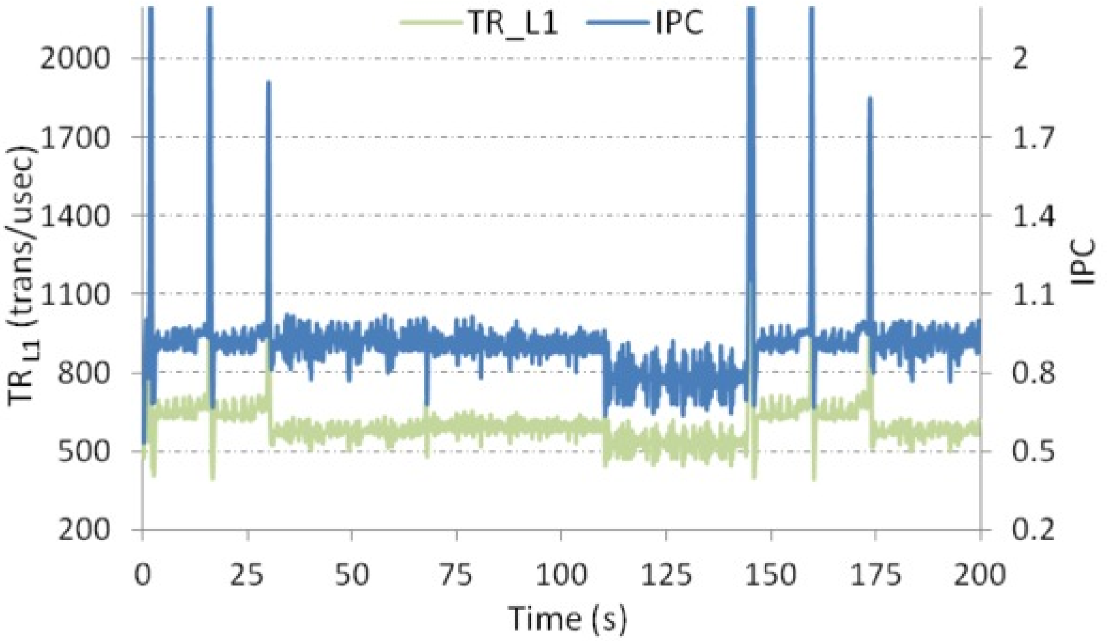
TR$_{L1}$ and IPC evolution with time for ***bzip2***

Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Dynamic values



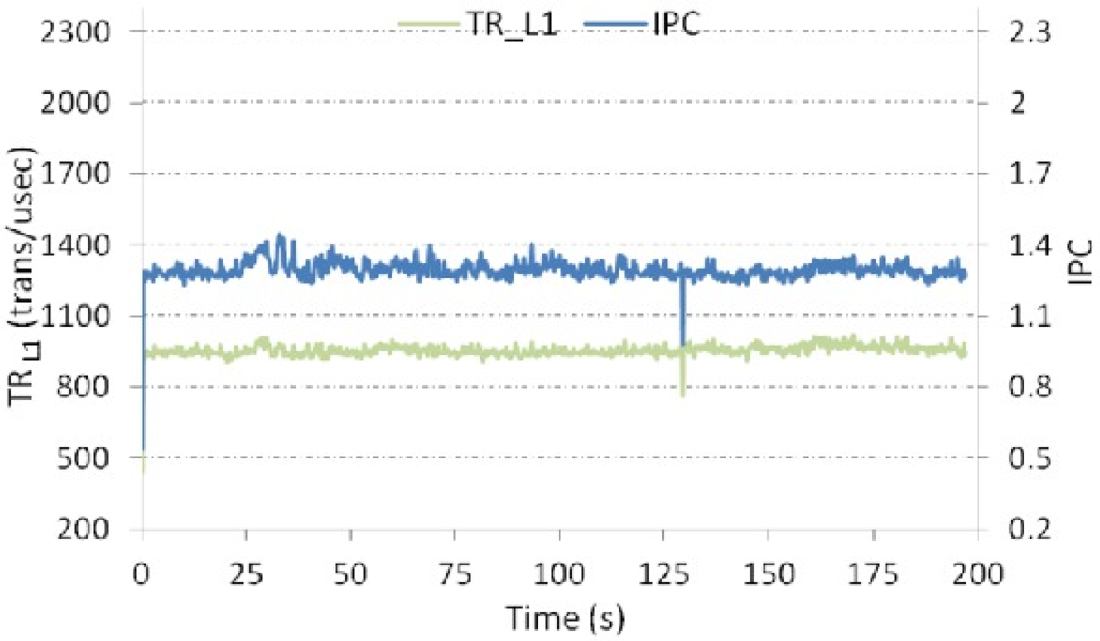
TR$_{L1}$ and IPC evolution with time for ***perlbench***



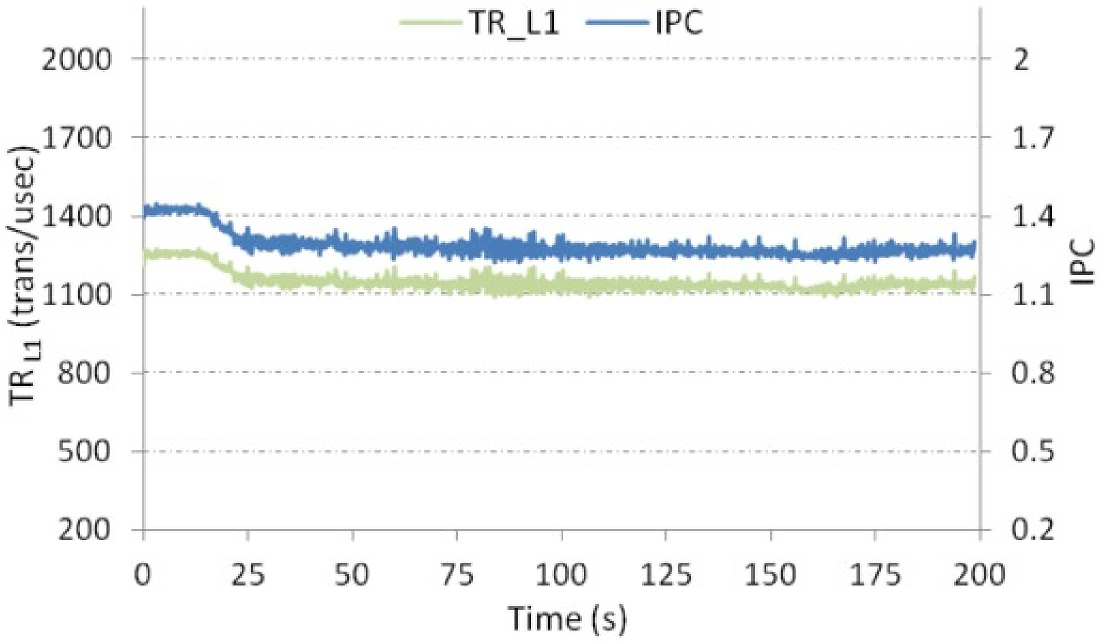
TR$_{L1}$ and IPC evolution with time for ***milc***

Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Dynamic values



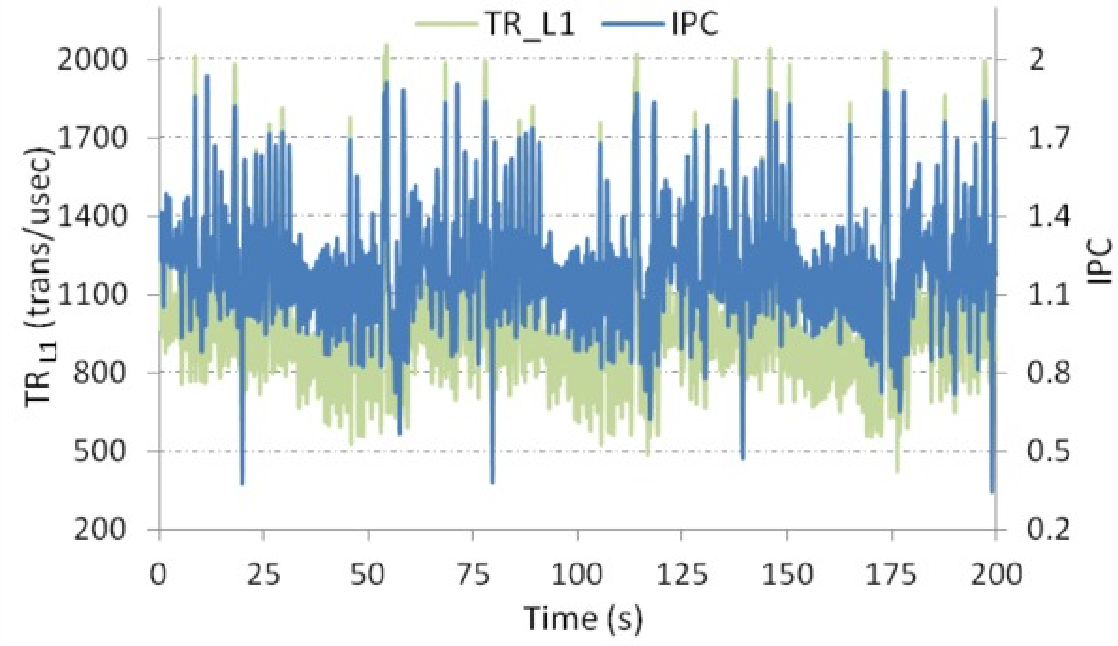
TR$_{L1}$ and IPC evolution with time for ***sjeng***



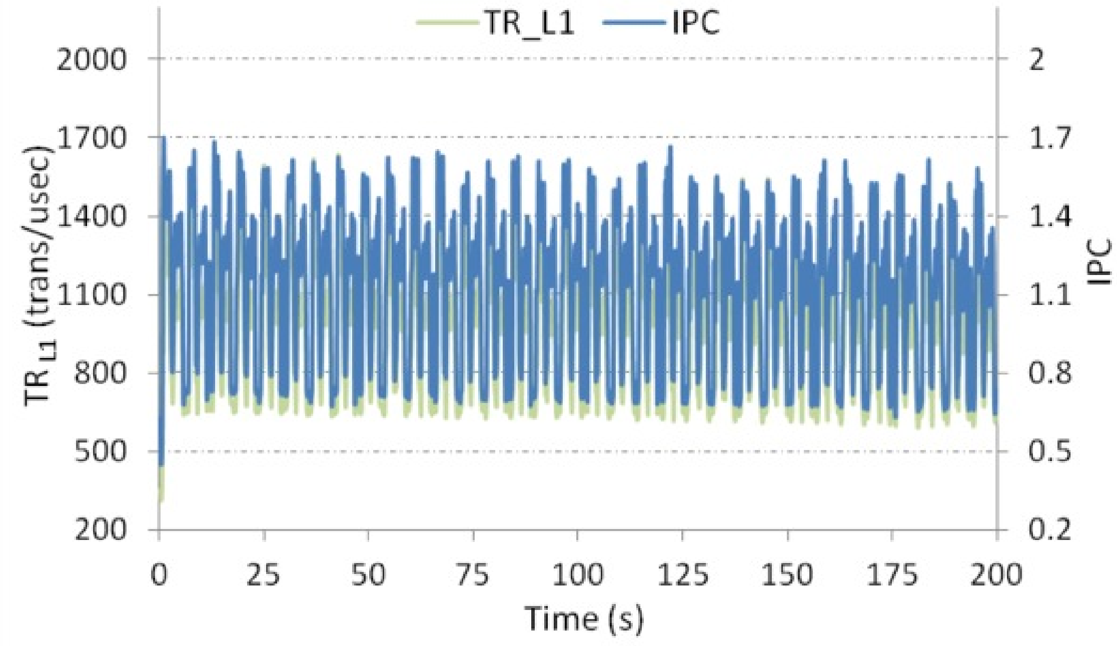
TR$_{L1}$ and IPC evolution with time for ***povray***

Effects of L1 bandwidth on performance of SMT processors

Stand-alone execution – Dynamic values



TR_{L1} and IPC evolution with time for ***gcc***



TR_{L1} and IPC evolution with time for ***zeusMP***

Effects of L1 bandwidth on performance of SMT processors

- Stand-alone execution
  - Average
    - ➢Certain similarities appear among average values of L1 bandwidth and IPC, but there is no clear evidence about the connection between them
  - Dynamic
    - ➢Synchronized and correlated trend between the L1 bandwidth of a thread and its performance
- Concurrent execution

Effects of L1 bandwidth on performance of SMT processors

- Stand-alone execution
  - Average
    - Certain similarities appear among average values of L1 bandwidth and IPC, but there is no clear evidence about the connection between them
  - Dynamic
    - Synchronized and correlated trend between the L1 bandwidth of a thread and its performance
- Concurrent execution
  - Two threads running simultaneously on a given core **share the L1 cache**

Effects of L1 bandwidth on performance of SMT processors

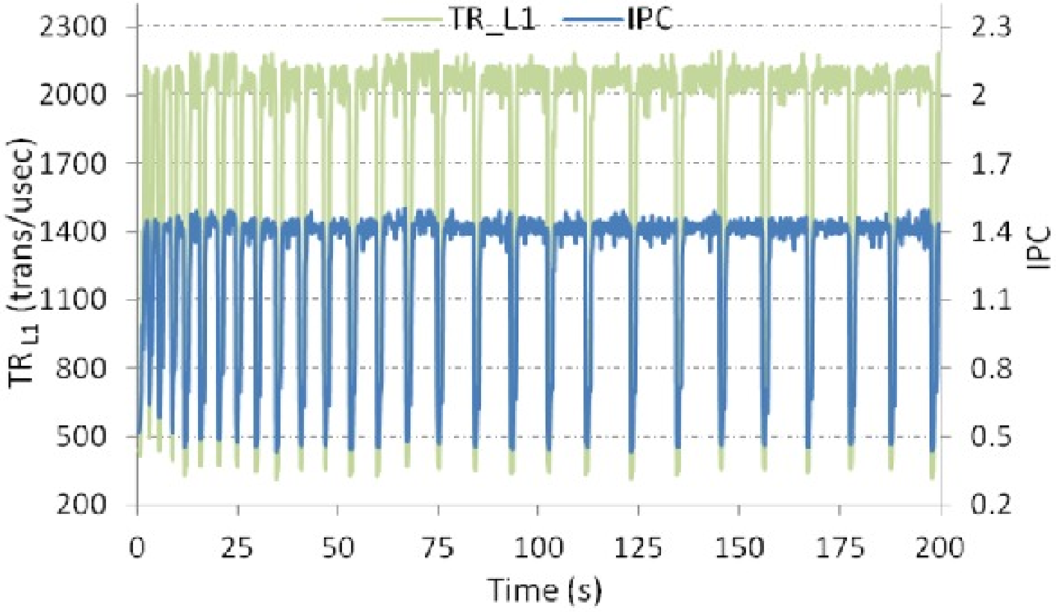
- Stand-alone execution
  - Average
    - ➢ Certain similarities appear among average values of L1 bandwidth and IPC, but there is no clear evidence about the connection between them
  - Dynamic
    - ➢ Synchronized and correlated trend between the L1 bandwidth of a thread and its performance
- Concurrent execution
  - Two threads running simultaneously on a given core **share the L1 cache**
  - Their **performance depend on the L1 bandwidth** they achieve

Effects of L1 bandwidth on performance of SMT processors

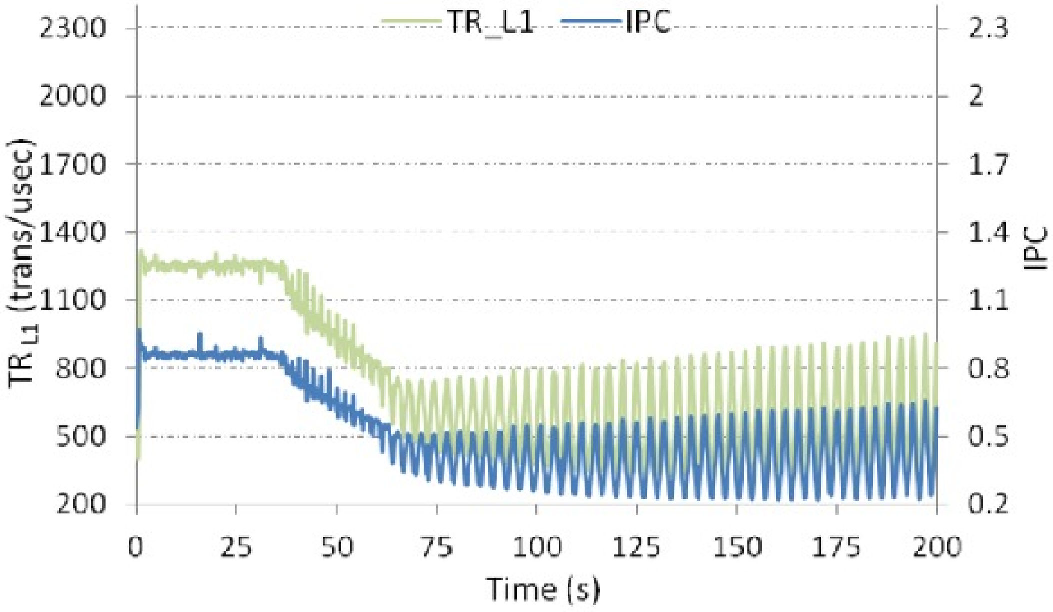
- Stand-alone execution
  - Average
    - Certain similarities appear among average values of L1 bandwidth and IPC, but there is no clear evidence about the connection between them
  - Dynamic
    - Synchronized and correlated trend between the L1 bandwidth of a thread and its performance
- Concurrent execution
  - Two threads running simultaneously on a given core **share the L1 cache**
  - Their **performance depend on the L1 bandwidth** they achieve
    - Competition for the L1 bandwidth will limit the performance

Effects of L1 bandwidth on performance of SMT processors

Interferences between co-runners



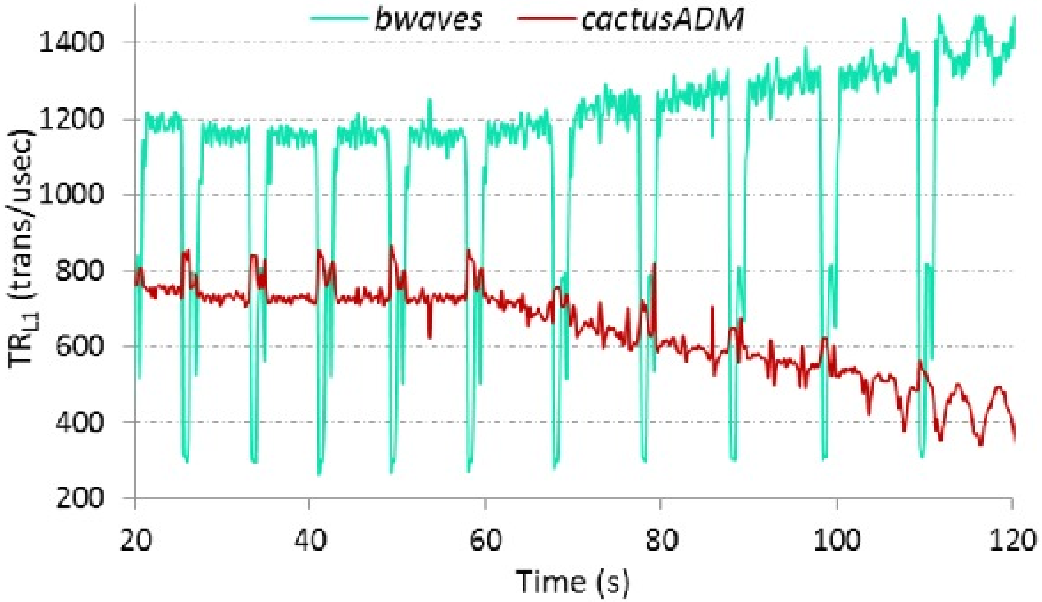
TR_{L1} and IPC evolution with time for ***bwaves***



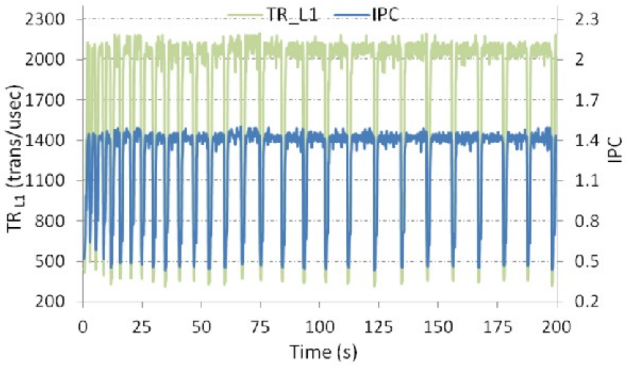
TR_{L1} and IPC evolution with time for **cactusADM**

Effects of L1 bandwidth on performance of SMT processors

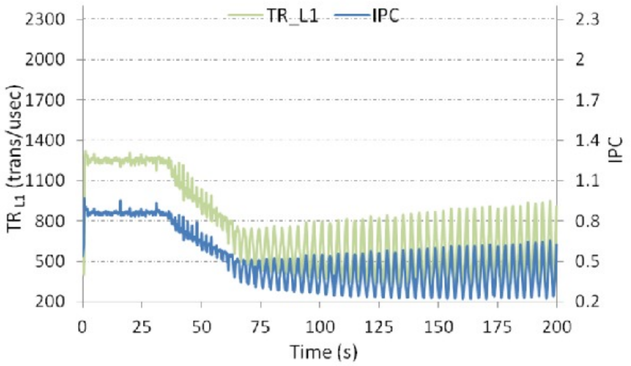
Interferences between co-runners



TR_{L1} of ***bwaves*** and ***cactusADM*** running on the same core



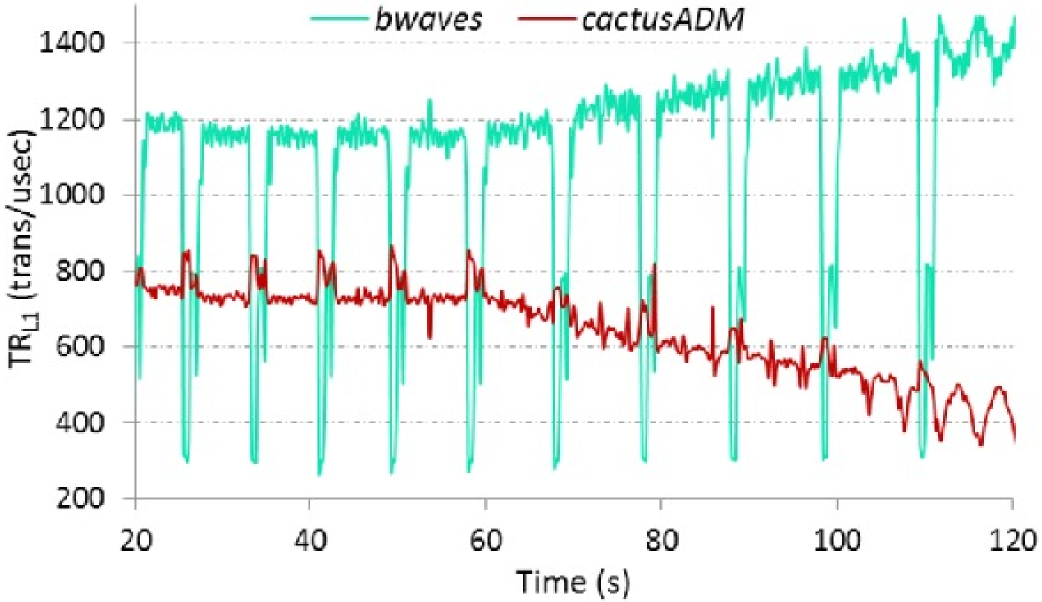
TR_{L1} and IPC evolution with time for ***bwaves***



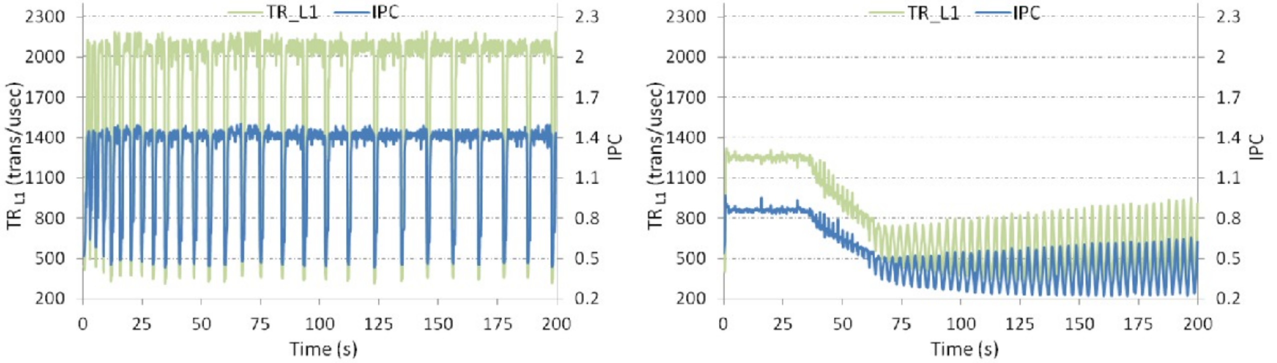
TR_{L1} and IPC evolution with time for ***cactusADM***

Effects of L1 bandwidth on performance of SMT processors

Interferences between co-runners



TR$_{L1}$ of ***bwaves*** and ***cactusADM*** running on the same core



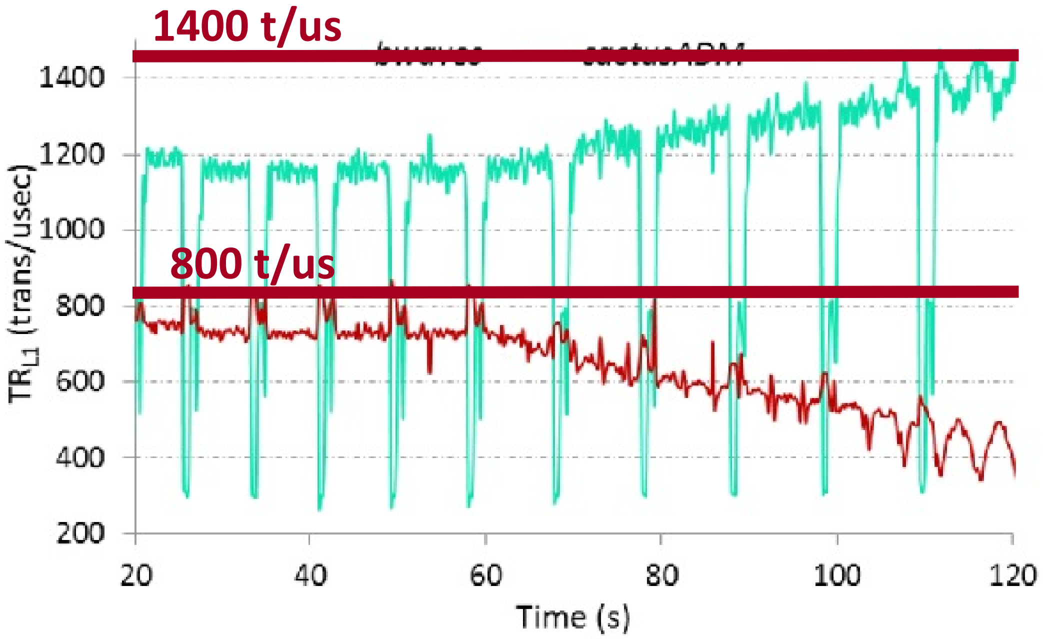
TR$_{L1}$ and IPC evolution with time for ***bwaves***

TR$_{L1}$ and IPC evolution with time for ***cactusADM***

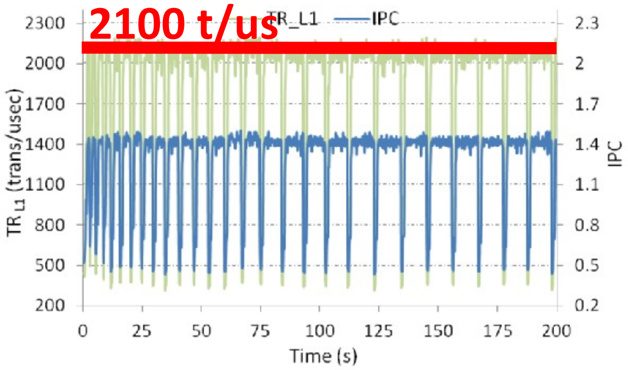
- The L1 bandwidth of the core cannot satisfy the requirements of both threads

Effects of L1 bandwidth on performance of SMT processors

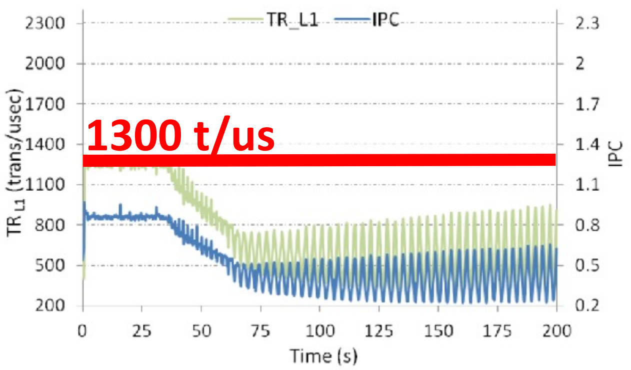
Interferences between co-runners



TR_{L1} of ***bwaves*** and ***cactusADM*** running on the same core



TR_{L1} and IPC evolution with time for ***bwaves***

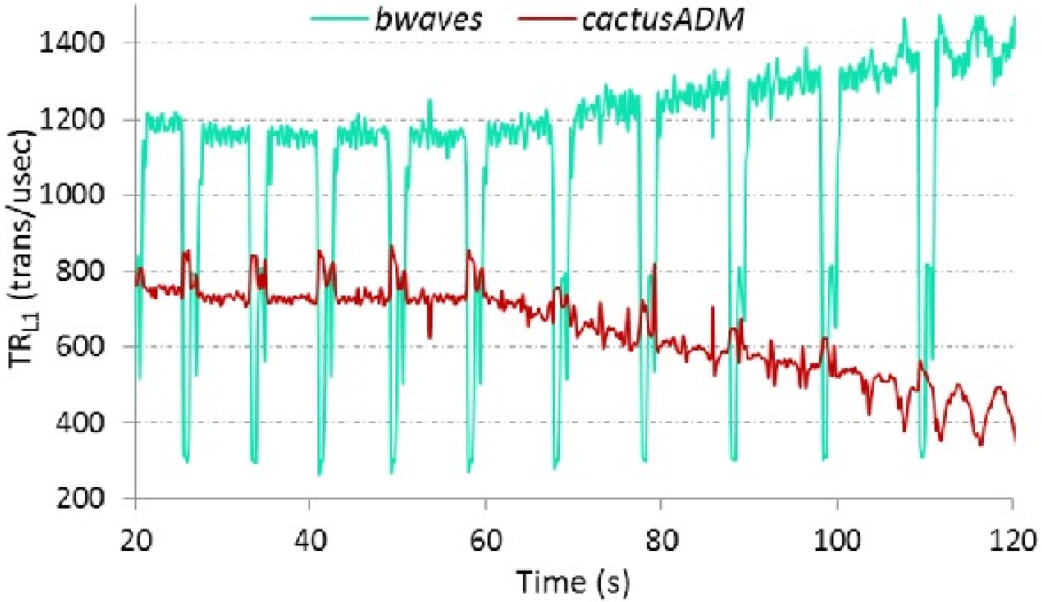


TR_{L1} and IPC evolution with time for ***cactusADM***

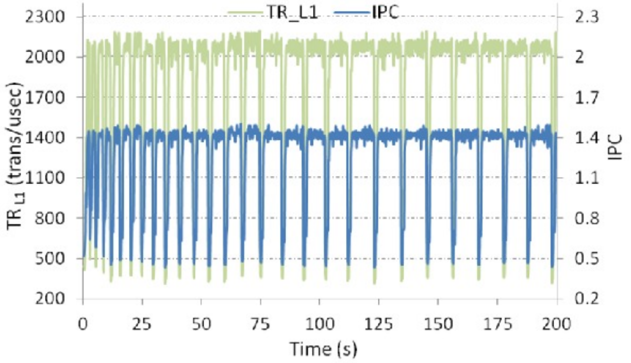
- The L1 bandwidth of the core cannot satisfy the requirements of both threads
  - *Bwaves* maximum is around 1400 t/usec (2100 t/usec in stand-alone execution)
  - *CactusADM* maximum is around 800 t/usec (1300 t/usec in standa-alone execution)

Effects of L1 bandwidth on performance of SMT processors

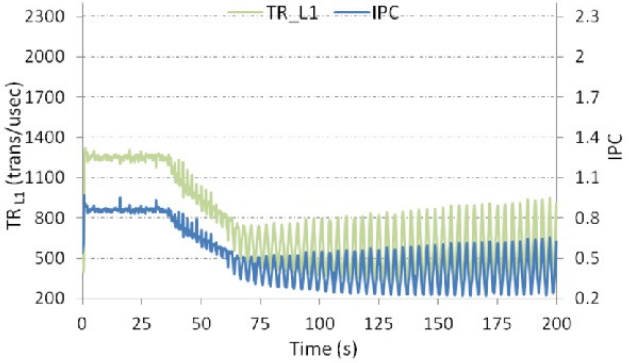
Interferences between co-runners



TR$_{L1}$ of ***bwaves*** and ***cactusADM*** running on the same core



TR$_{L1}$ and IPC evolution with time for ***bwaves***



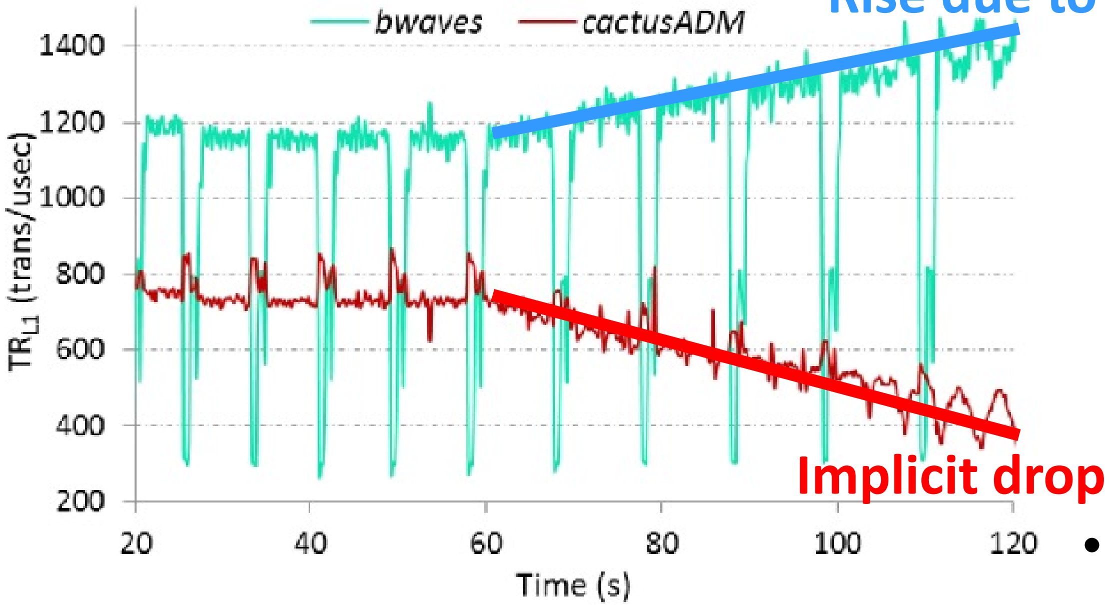
TR$_{L1}$ and IPC evolution with time for ***cactusADM***

- The L1 bandwidth of the core cannot satisfy the requirements of both threads

- The L1 bandwidth utilization of a thread depends on the L1 bandwidth utilization of the co-runner

Effects of L1 bandwidth on performance of SMT processors

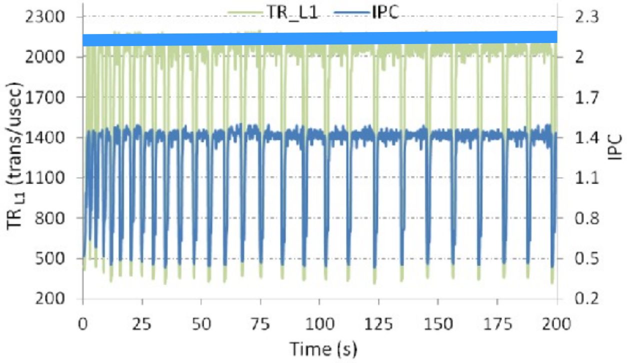
Interferences between co-runners



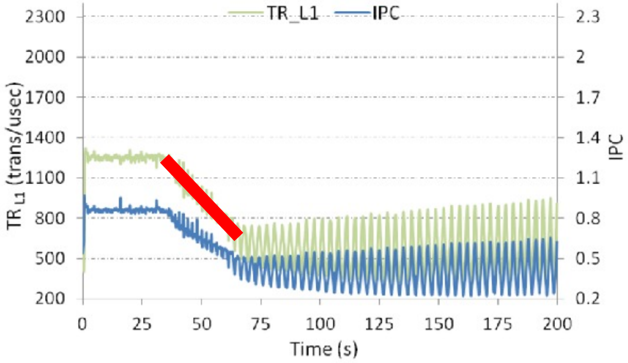
Rise due to the co-runner drop

Implicit drop

TR_{L1} of ***bwaves*** and ***cactusADM*** running on the same core



TR_{L1} and IPC evolution with time for ***bwaves***

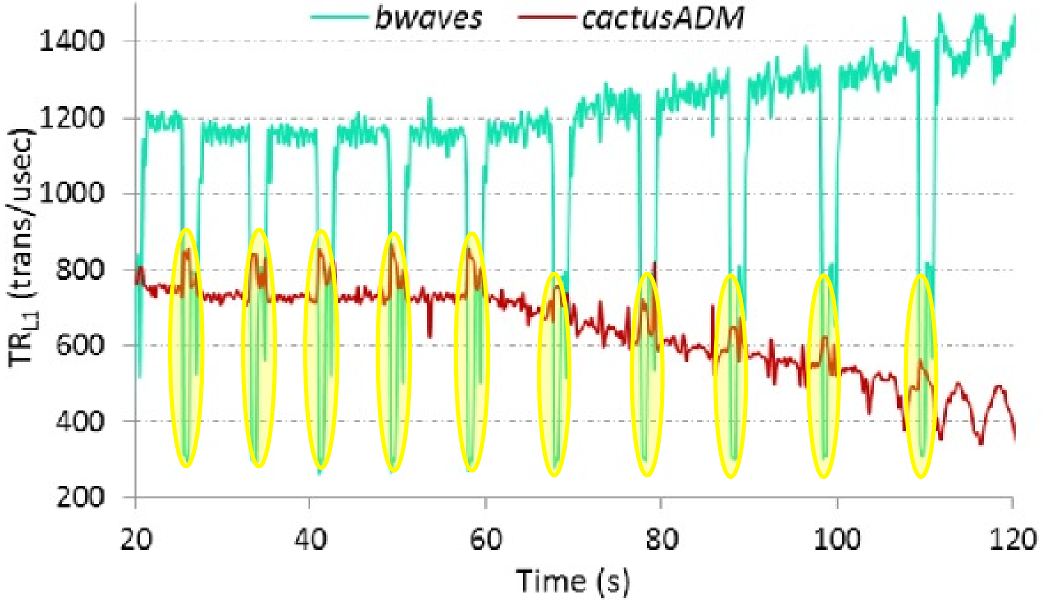


TR_{L1} and IPC evolution with time for ***cactusADM***

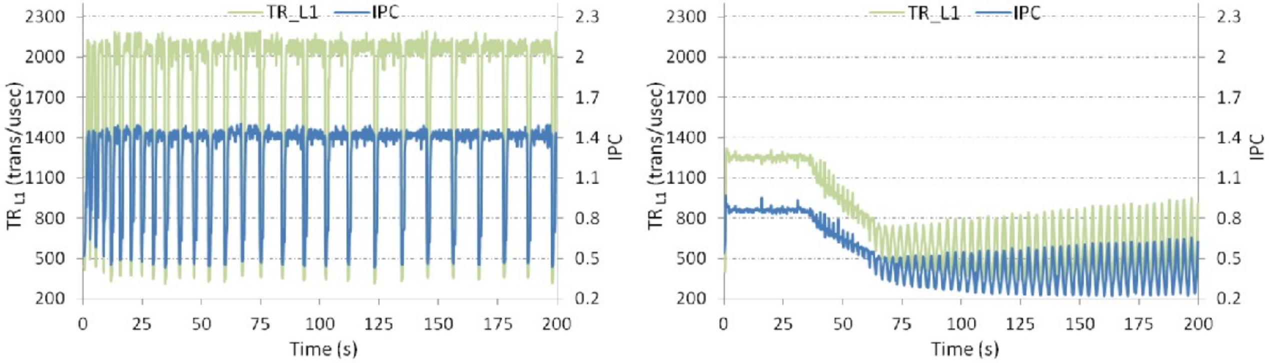
- The L1 bandwidth of the core cannot satisfy the requirements of both threads
- The L1 bandwidth utilization of a thread depends on the L1 bandwidth utilization of the co-runner
    - Increasing trend of the L1 bandwidth of *bwaves* due to L1 bandwidth utilization decrease of *cactusADM*

Effects of L1 bandwidth on performance of SMT processors

Interferences between co-runners



TR_{L1} of ***bwaves*** and ***cactusADM*** running on the same core



TR_{L1} and IPC evolution with time for ***bwaves***

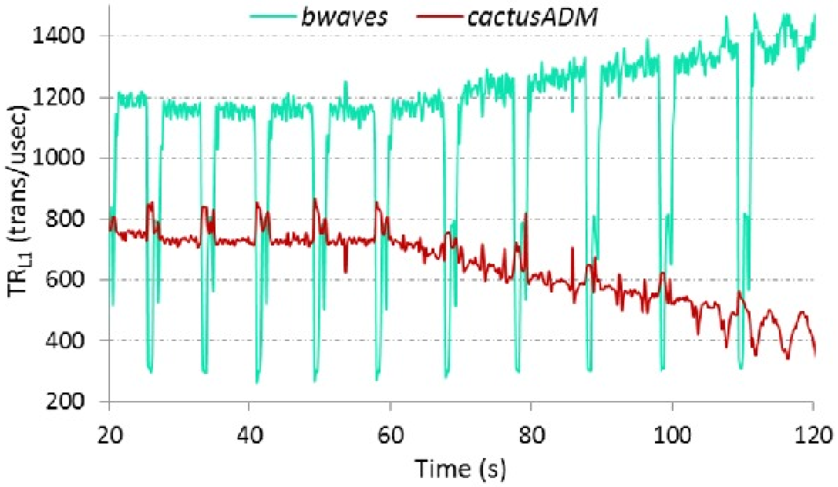
TR_{L1} and IPC evolution with time for ***cactusADM***

- The L1 bandwidth of the core cannot satisfy the requirements of both threads

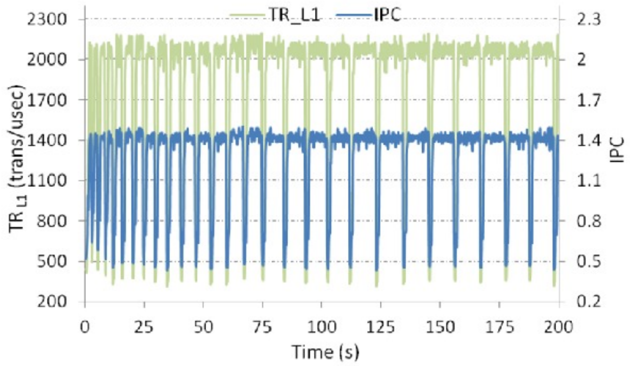
- The L1 bandwidth utilization of a thread depends on the L1 bandwidth utilization of the co-runner
    - Increasing trend of the L1 bandwidth of *bwaves* due to L1 bandwidth utilization decrease of *cactusADM*
    - Peaks on the L1 bandwidth of *cactusADM* due to L1 bandwidth drops of *bwaves*

Effects of L1 bandwidth on performance of SMT processors

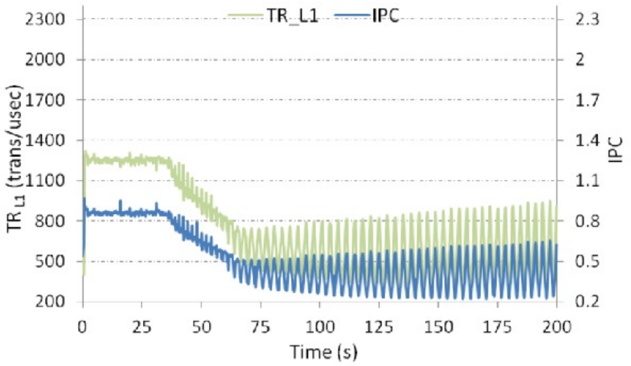
Interferences between co-runners



TR_{L1} of ***bwaves*** and ***cactusADM*** running on the same core

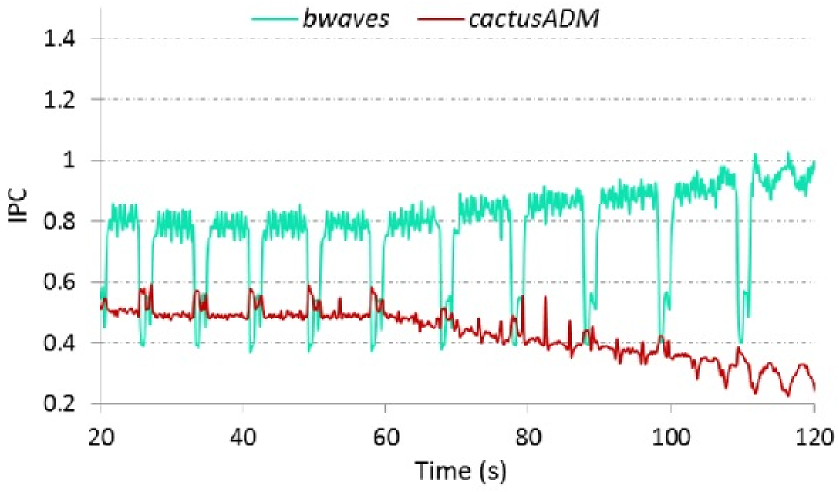


TR_{L1} and IPC evolution with time for ***bwaves***



TR_{L1} and IPC evolution with time for ***cactusADM***

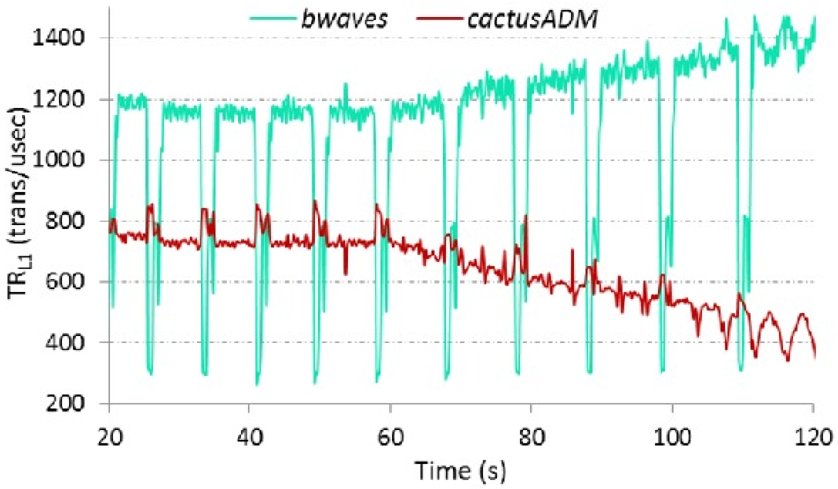
- The L1 bandwidth of the core cannot satisfy the requirements of both threads
- The L1 bandwidth utilization of a thread depends on the L1 bandwidth utilization of the co-runner
- **The observations of the L1 bandwidth are applicable to performance**



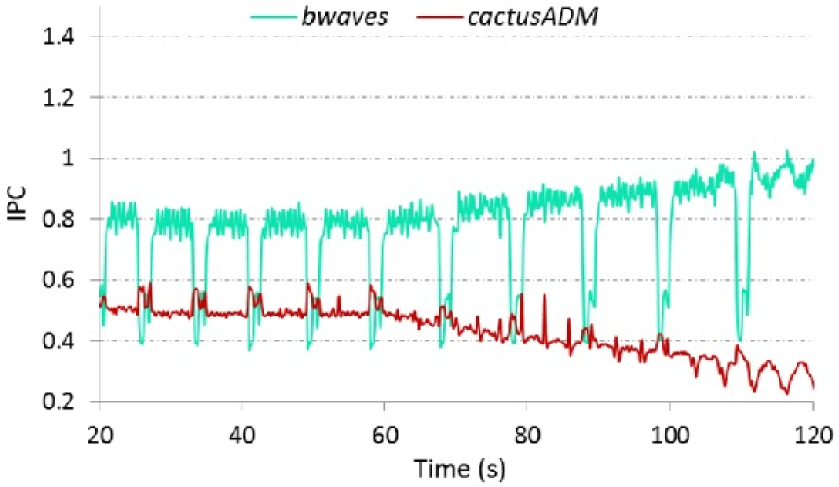
IPC of ***bwaves*** and ***cactusADM*** running on the same core

Effects of L1 bandwidth on performance of SMT processors

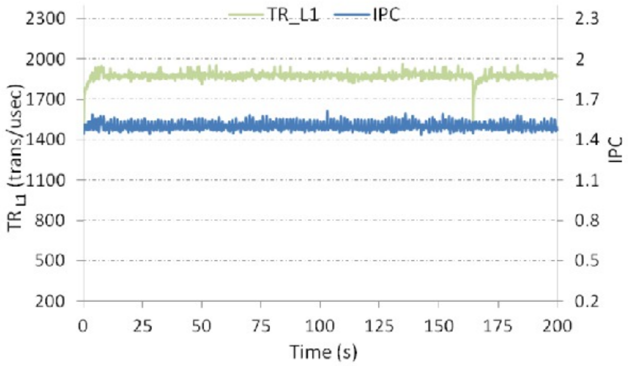
Interferences between co-runners



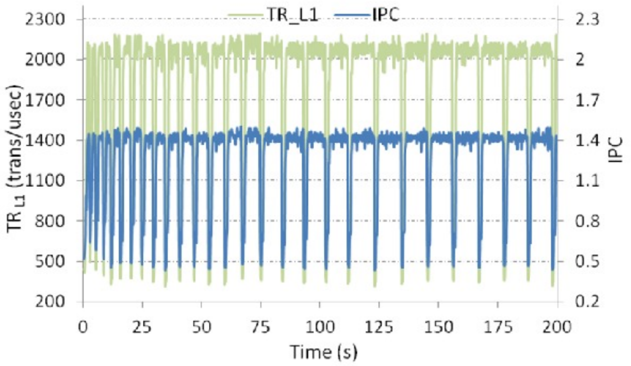
TR$_{L1}$ of ***h264ref*** and ***bwaves*** running on the same core



IPC of ***h264ref*** and ***bwaves*** running on the same core



TR$_{L1}$ and IPC evolution with time for ***h264ref***

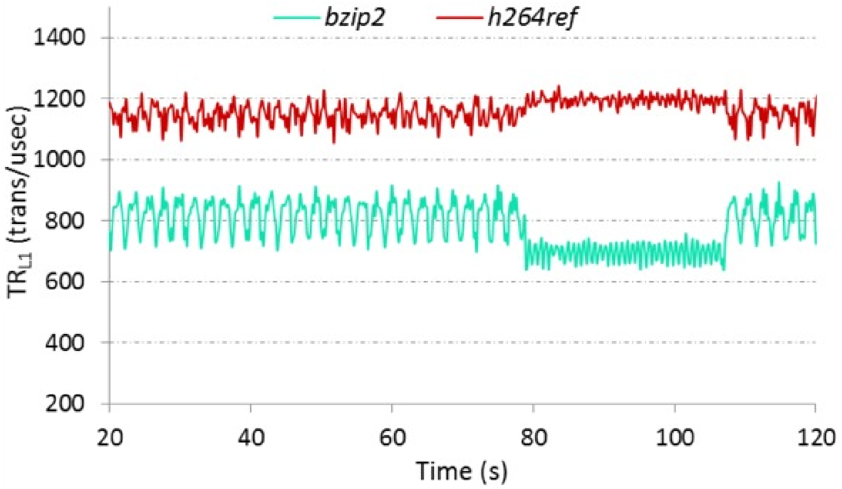


TR$_{L1}$ and IPC evolution with time for ***bwaves***

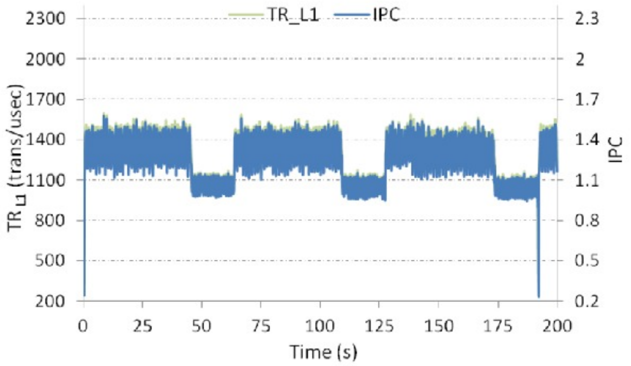
- The L1 bandwidth of the core cannot satisfy the requirements of both threads
- The L1 bandwidth utilization of a thread depends on the L1 bandwidth utilization of the co-runner
- The observations of the L1 bandwidth are applicable to performance

Effects of L1 bandwidth on performance of SMT processors

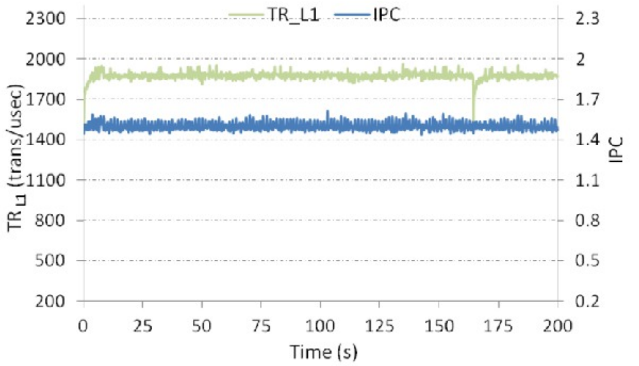
Interferences between co-runners



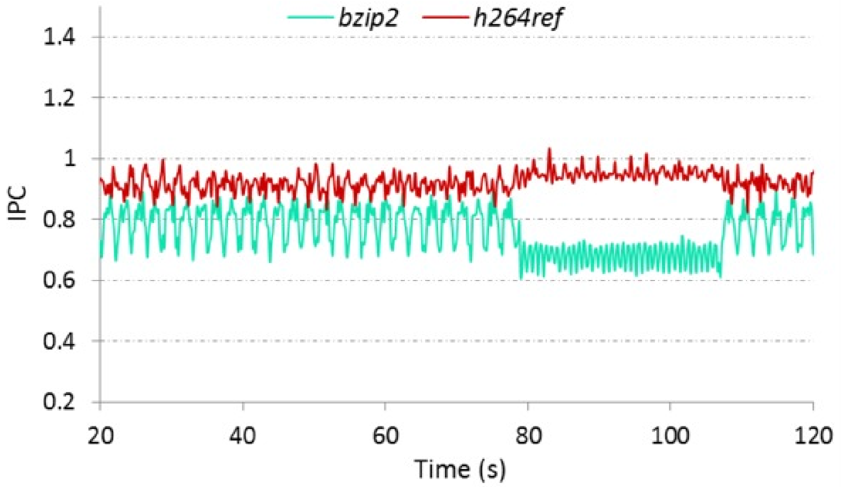
TR$_{L1}$ of ***bzip2*** and ***h264ref*** running on the same core



TR$_{L1}$ and IPC evolution with time for ***bzip2***



TR$_{L1}$ and IPC evolution with time for ***h264ref***

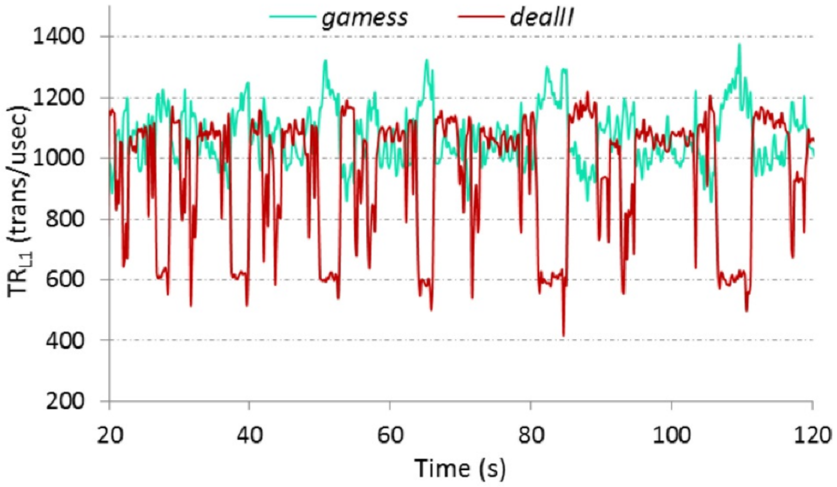


IPC of ***bzip2*** and ***h264ref*** running on the same core

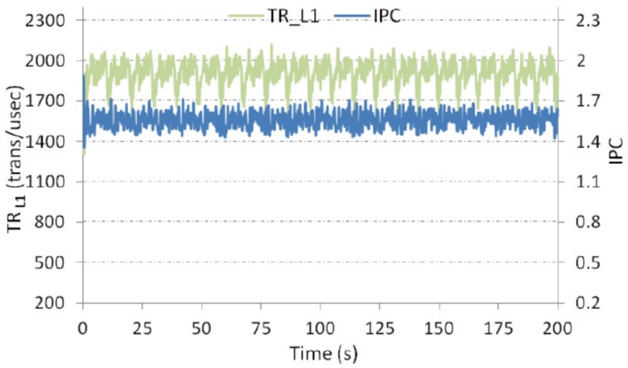
- The L1 bandwidth of the core cannot satisfy the requirements of both threads
- The L1 bandwidth utilization of a thread depends on the L1 bandwidth utilization of the co-runner
- The observations of the L1 bandwidth are applicable to performance

Effects of L1 bandwidth on performance of SMT processors

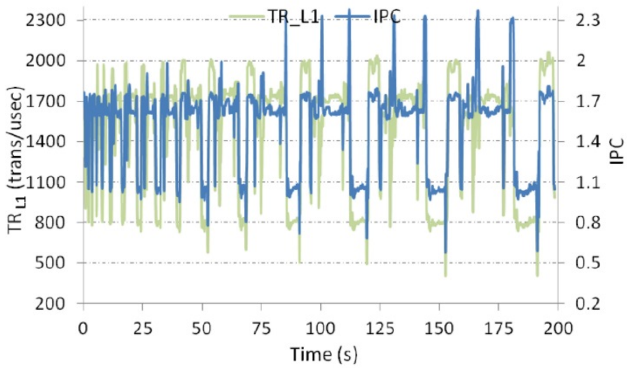
Interferences between co-runners



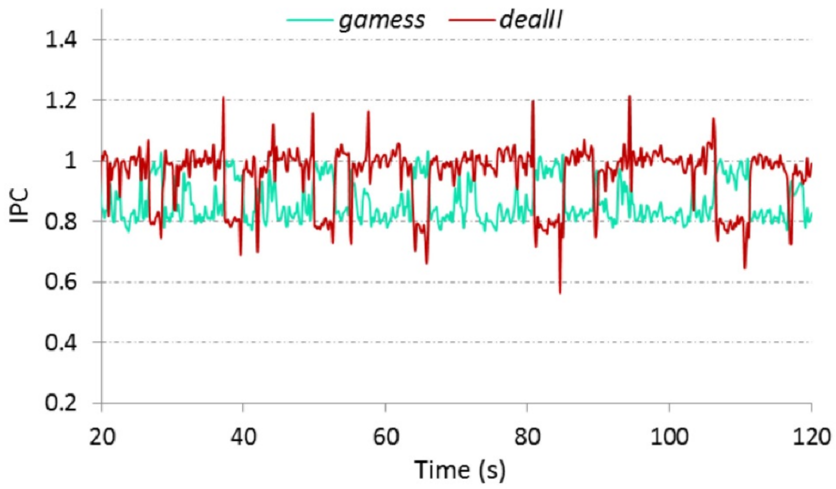
TR_{L1} of ***gamess*** and ***dealII*** running on the same core



TR_{L1} and IPC evolution with time for ***gamess***



TR_{L1} and IPC evolution with time for ***dealII***



IPC of ***gamess*** and ***dealII*** running on the same core

- The L1 bandwidth of the core cannot satisfy the requirements of both threads
- The L1 bandwidth utilization of a thread depends on the L1 bandwidth utilization of the co-runner
- The observations of the L1 bandwidth are applicable to performance

Effects of L1 bandwidth on performance of SMT processors

- Stand-alone execution
  - Average
    - ➢ Certain similarities appear among L1 bandwidth and IPC, but there is no clear evidence about the connection between them.
  - Dynamic
    - ➢ Synchronized and correlated trend between the L1 bandwidth of a thread and its performance.
- Concurrent execution
    - ➢ **Insufficient L1 bandwidth** to satisfy the requirements of two processes.

Effects of L1 bandwidth on performance of SMT processors

- Stand-alone execution
  - Average
    - ➢ Certain similarities appear among L1 bandwidth and IPC, but there is no clear evidence about the connection between them.
  - Dynamic
    - ➢ Synchronized and correlated trend between the L1 bandwidth of a thread and its performance.
- Concurrent execution
    - ➢ **Insufficient L1 bandwidth** to satisfy the requirements of two processes.
    - ➢ L1 bandwidth rises and drops on a the L1 bandwidth (or performance) of a process trigger the opposite behavior in the co-runner

Outline

- Introduction
- Experimental platform
- Effects of L1 bandwidth on performance of SMT processors
- L1-bandwidth aware thread allocation policies
- Evaluation methodology
- Performance evaluation results
- Conclusions

L1 bandwidth aware thread allocation policies

- We devise two policies
    - Static thread allocation policy (St2c)
    - Dynamic thread allocation policy (Dt2c)

L1 bandwidth aware thread allocation policies

- We devise two policies
  - Static thread allocation policy (St2c)
  - Dynamic thread allocation policy (Dt2c)

- The goal of both policies is to balance the overall L1 bandwidth of the running processes among the processor cores

L1 bandwidth aware thread allocation policies

- We devise two policies
  - Static thread allocation policy (St2c)
  - Dynamic thread allocation policy (Dt2c)

- The goal of both policies is to balance the overall L1 bandwidth of the running processes among the processor cores

- The policies rely on the L1 bandwidth requirements of the processes to guide the thread allocation
  - Differ in how L1 bandwidth is estimated

L1 bandwidth aware thread allocation policies

- We devise two policies
  - Static thread allocation policy (St2c)
  - Dynamic thread allocation policy (Dt2c)
- The goal of both policies is to balance the overall L1 bandwidth of the running processes among the processor cores
- The policies rely on the L1 bandwidth requirements of the processes to guide the thread allocation
  - Differ in how L1 bandwidth is estimated
- The t2c policies can work as a step of a global scheduler
  - No process selection performed in our policies

L1 bandwidth aware thread allocation policies

St2c: Static thread to core allocation policy

- Threads are allocated to cores based on the average L1 bandwidth requirement of each process

L1 bandwidth aware thread allocation policies

St2c: Static thread to core allocation policy

- Threads are allocated to cores based on the average L1 bandwidth requirement of each process
  - Requires a preliminary stand-alone execution of each process

L1 bandwidth aware thread allocation policies

St2c: Static thread to core allocation policy

- Threads are allocated to cores based on the average L1 bandwidth requirement of each process
  - Requires a preliminary stand-alone execution of each process
  - Thread to core mappings only update when the running processes change

L1 bandwidth aware thread allocation policies

St2c: Static thread to core allocation policy

- Threads are allocated to cores based on the average L1 bandwidth requirement of each process
  - Requires a preliminary stand-alone execution of each process
  - Thread to core mappings only update when the running processes change

- To balance L1 bandwidth
  - Threads are sorted in increasing L1 bandwidth
  - Reiteratively, the threads with maximum and minimum requirements are selected to share a given core

L1 bandwidth aware thread allocation policies

St2c: Static thread to core allocation policy

- Major benefits
  - Good estimation for benchmarks with uniform L1-bandwidth shape 🙂
    - 11 of 25 analyzed benchmarks

L1 bandwidth aware thread allocation policies

St2c: Static thread to core allocation policy

- Major benefits
    - Good estimation for benchmarks with uniform L1-bandwidth shape 🙂
    - Avoids interferences of co-runners in the L1 bandwidth estimations 🙂
        - L1 bandwidth measured in stand-alone execution

L1 bandwidth aware thread allocation policies

St2c: Static thread to core allocation policy

- Major benefits
  - Good estimation for benchmarks with uniform L1-bandwidth shape ☺
  - Avoids interferences of co-runners in the L1 bandwidth estimations ☺

- Major drawbacks
  - **Requires a preliminary run of processes** ☹
    - To estimate the L1 bandwidth

L1 bandwidth aware thread allocation policies

St2c: Static thread to core allocation policy

- Major benefits
  - Good estimation for benchmarks with uniform L1-bandwidth shape 🙂
  - Avoids interferences of co-runners in the L1 bandwidth estimations 🙂
- Major drawbacks
  - **Requires a preliminary run of processes** ☹
  - **Poor L1 bandwidth estimation** for processes with non-uniform shape ☹
    - 14 of the 25 analyzed benchmarks

L1 bandwidth aware thread allocation policies

Dt2c: Dynamic thread to core allocation policy

- Dt2c tackles both mentioned shortcomings of the St2c

L1 bandwidth aware thread allocation policies

Dt2c: Dynamic thread to core allocation policy

- Dt2c tackles both mentioned shortcoming of the St2c
  - The L1 bandwidth requirements of each process is dynamically updated at run-time using performance counters

L1 bandwidth aware thread allocation policies

Dt2c: Dynamic thread to core allocation policy

- Dt2c tackles both mentioned shortcoming of the St2c
  - The L1 bandwidth requirements of each process is dynamically updated at run-time using performance counters
    - Does not require any previous information of the processes 🙂

L1 bandwidth aware thread allocation policies

Dt2c: Dynamic thread to core allocation policy

- Dt2c tackles both mentioned shortcoming of the St2c
  - The L1 bandwidth requirements of each process is dynamically updated at run-time using performance counters
    - Does not require any previous information of the processes ☺
    - Captures the L1 bandwidth requirements of benchmarks with non-uniform shapes ☺

L1 bandwidth aware thread allocation policies

Dt2c: Dynamic thread to core allocation policy

- Dt2c tackles both mentioned shortcoming of the St2c
  - The L1 bandwidth requirements of each process is dynamically updated at run-time using performance counters
    - Does not require any previous information of the processes
    - Captures the L1 bandwidth requirements of benchmarks with non-uniform shapes

- Balancing L1 bandwidth can be performed as stated in the St2c policy
- L1 bandwidth updated every OS quantum

  →Thread to core mappings are updated after each OS quantum

Outline

- Introduction
- Experimental platform
- Effects of L1 bandwidth on performance of SMT processors
- L1-bandwidth aware thread allocation policies
- Evaluation methodology
- Performance evaluation results
- Conclusions

Evaluation methodology

- Four thread allocation policies are compared
    - St2c
    - Dt2c
    - Naïve t2c

    - Linux t2c

Evaluation methodology

- Four thread allocation policies are compared
  - St2c
  - Dt2c
  - Naïve t2c
    - Random, allowing unbalancing of L1 bandwidth among cores
  - Linux t2c

Evaluation methodology

- Four thread allocation policies are compared
  - St2c
  - Dt2c
  - Naïve t2c
    - Random, allowing unbalancing of L1 bandwidth among cores
  - Linux t2c
    - The affinity of all the threads is set to all the cores. Thus, Linux decides the thread allocation

Evaluation methodology

- Four thread allocation policies are compared
  - St2c
  - Dt2c
  - Naïve t2c
    - Random, allowing unbalancing of L1 bandwidth among cores
  - Linux t2c
    - The affinity of all the threads is set to all the cores. Thus, Linux decides the thread allocation

- All the policies are implemented in a user-level scheduler
  - Sharing the main code, and any possible overhead

Evaluation methodology

- Focuses on avoiding performance differences to be caused by early finalization of the execution of some threads

Evaluation methodology

- Focuses on avoiding performance differences to be caused by early finalization of the execution of some threads
  - The workloads run with half the number of cores than the number of benchmarks
    - All the hardware threads are used

Evaluation methodology

- Focuses on avoiding performance differences to be caused by early finalization of the execution of some threads
  - The workloads run with half the number of cores than the number of benchmarks
    - All the hardware threads are used
  - Target number of instructions for each benchmarks
    - Instructions required to run 200 seconds in stand-alone execution

Evaluation methodology

- Focuses on avoiding performance differences to be caused by early finalization of the execution of some threads
  - The workloads run with half the number of cores than the number of benchmarks
    - All the hardware threads are used
  - Target number of instructions for each benchmarks
    - Instructions required to run 200 seconds in stand-alone execution
  - All the processes of the mix run until the last processes completes its target number of instructions

Evaluation methodology

- Focuses on avoiding performance differences to be caused by early finalization of the execution of some threads
  - The workloads run with half the number of cores than the number of benchmarks
    - All the hardware threads are used
  - Target number of instructions for each benchmarks
    - Instructions required to run 200 seconds in stand-alone execution
  - All the processes of the mix run until the last processes completes its target number of instructions
    - The performance metric (IPC) of each process is obtained at the point it completes its target number of instructions

Evaluation methodology

- Focuses on avoiding performance differences to be caused by early finalization of the execution of some threads
  - The workloads run with half the number of cores than the number of benchmarks
    - All the hardware threads are used
  - Target number of instructions for each benchmarks
    - Instructions required to run 200 seconds in stand-alone execution
  - All the processes of the mix run until the last processes completes its target number of instructions
    - The performance metric (IPC) of each process is obtained at the point it completes its target number of instructions
    - We compare the same part of the execution of each benchmarks across different runs of the mix

Evaluation methodology

- Focuses on avoiding performance differences to be caused by early finalization of the execution of some threads
  - The workloads run with half the number of cores than the number of benchmarks
    - All the hardware threads are used
  - Target number of instructions for each benchmarks
    - Instructions required to run 200 seconds in stand-alone execution
  - All the processes of the mix run until the last processes completes its target number of instructions
    - The performance metric (IPC) of each process is obtained at the point it completes its target number of instructions
    - We compare the same part of the execution of each benchmarks across different runs of the mix
    - The contribution of all the processes to the mix metrics is equalized

Evaluation methodology

Metrics

- Average IPC
  - Plain metric to measure throughput improvements

Evaluation methodology

Metrics

- Average IPC
    - Plain metric to measure throughput improvements
    - Can favor unfair scheduling ☹
        - Threads with higher IPC could receive higher weight to improve the average IPC

Evaluation methodology

Metrics

- Average IPC
  - Plain metric to measure throughput improvements
  - Can favor unfair scheduling
    - Threads with higher IPC could receive higher weight to improve the average IPC
    - **These situations are avoided in our methodology**, because the set of running processes is fixed and kept the entire execution 🙂

Evaluation methodology

Metrics

- Average IPC
  - Plain metric to measure throughput improvements
  - Can favor unfair scheduling
    - Threads with higher IPC could receive higher weight to improve the average IPC
    - **These situations are avoided in our methodology**, because the set of running processes is fixed and kept the entire execution

- Harmonic mean of weighted IPC
  - Encapsulates fairness additionally to performance
    - The harmonic mean tends to be low if any thread presents much lower speedup than the others

Evaluation methodology

Mix design

- The higher the bandwidth requirements
  - The higher L1 bandwidth contention can induce
  - The higher performance degradation can suffer

Evaluation methodology

Mix design

- The higher the bandwidth requirements
  - The higher L1 bandwidth contention can induce
  - The higher performance degradation can suffer

→The more critical its allocation is

Evaluation methodology

Mix design

- The higher the bandwidth requirements
  - The higher L1 bandwidth contention can induce
  - The higher performance degradation can suffer

→The more critical its allocation is

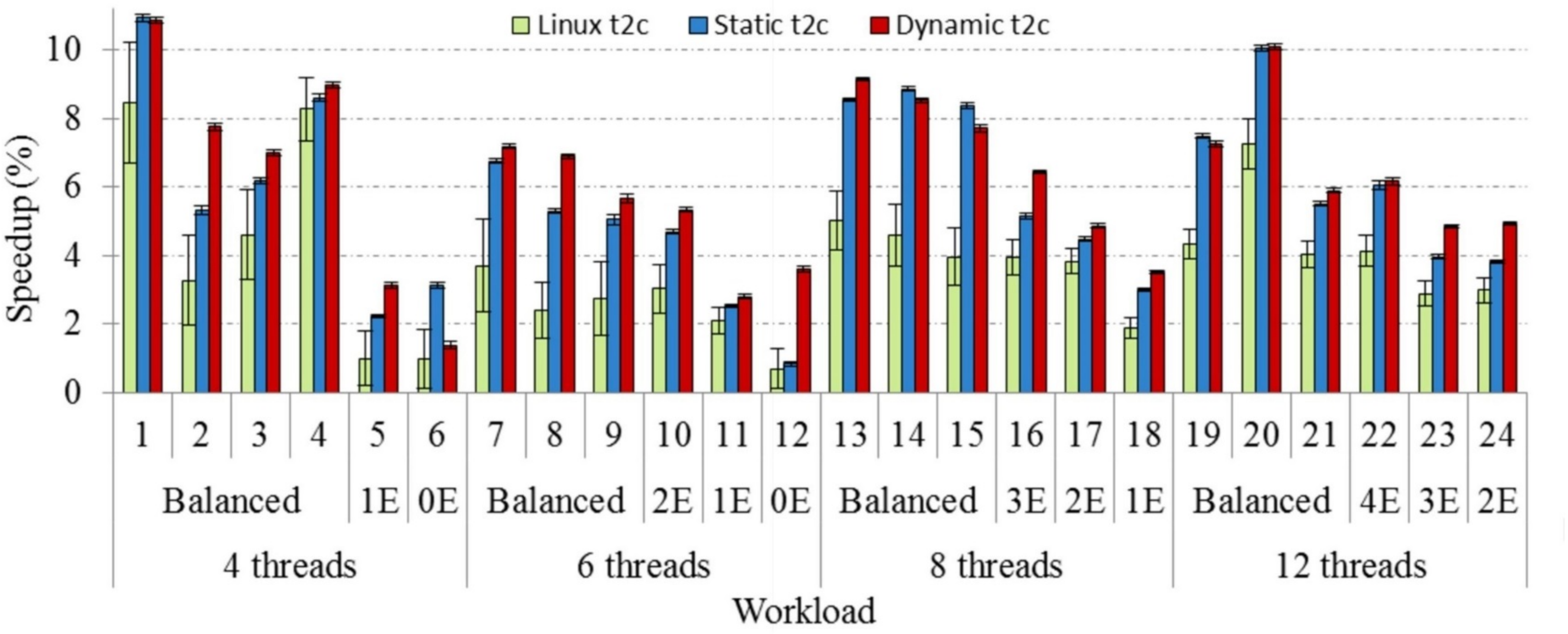
- Mixes are classified according to their number of benchmarks with extreme L1 bandwidth requirements (more than 1700 trans/usec)
  - Balanced mixes: half of the benchmarks with extreme L1 bandwidth
  - Non-balanced mixes: fewer number of benchmarks with extreme L1 bandwidth requirements than with lower requirements

Outline

- Introduction
- Experimental platform
- Effects of L1 bandwidth on performance of SMT processors
- L1-bandwidth aware thread allocation policies
- Evaluation methodology
- Performance evaluation results
- Conclusions

Performance evaluation results

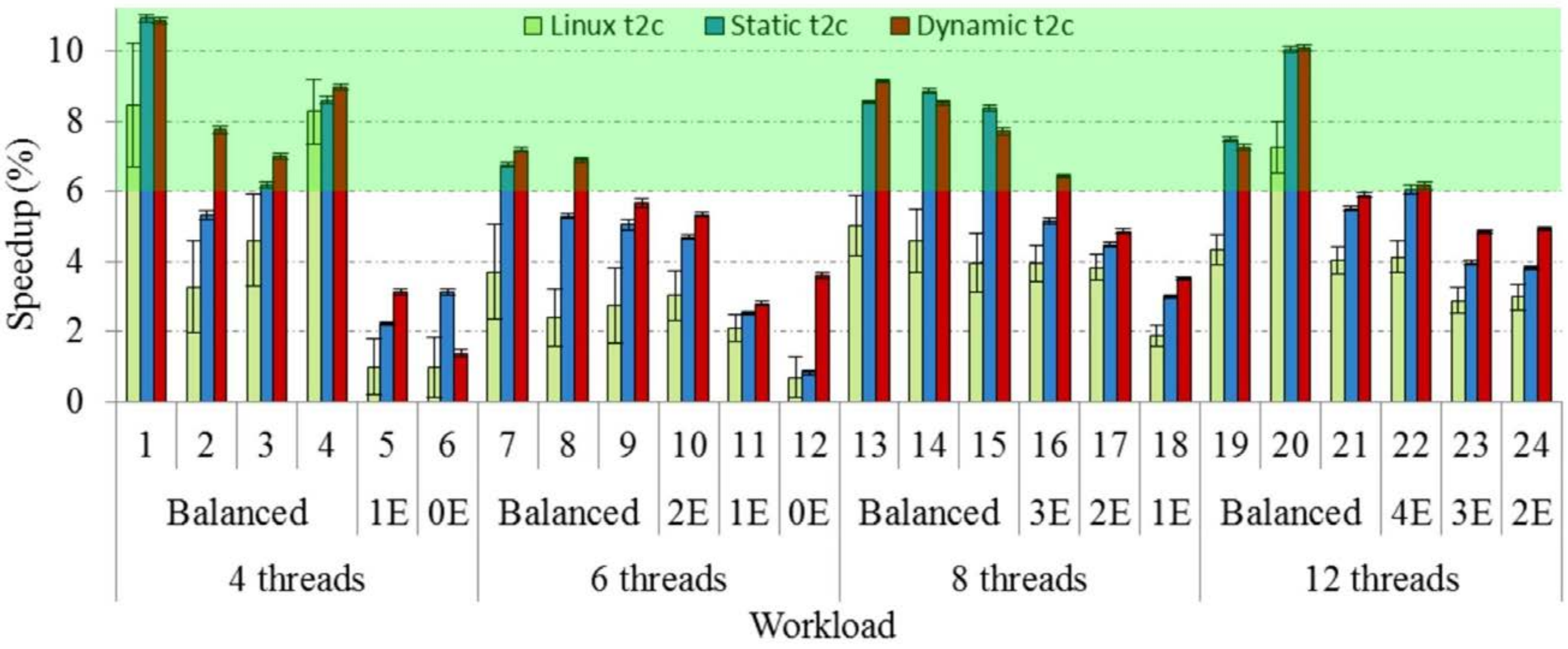
Speedup of the average IPC w.r.t. naïve t2c



- Average speedups of 20 executions of each mix and 95% confidence intervals
- Balanced mixes include half of the mixes with *extreme L1 bandwidth*
- Non balanced mixes nomenclature
  - ❑ XE refers to a mix with *X extreme L1 bandwidth* benchmarks

Performance evaluation results

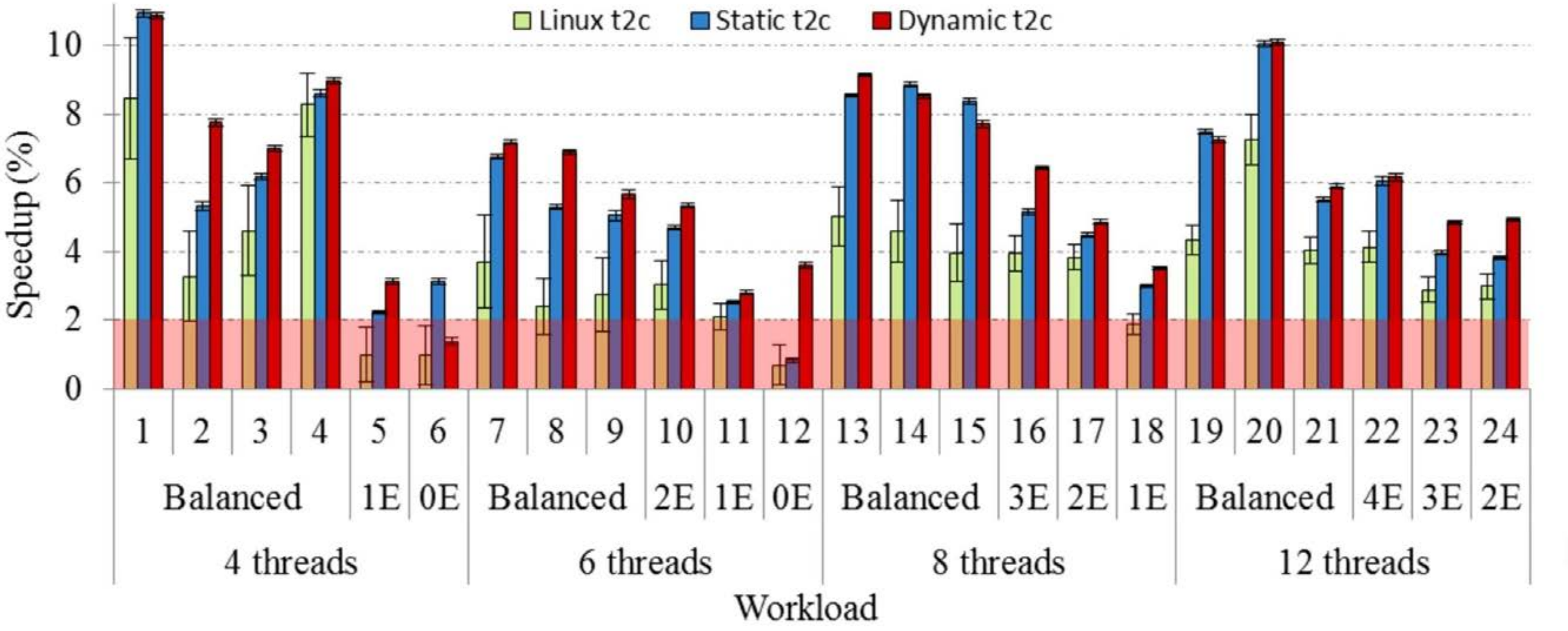
Speedup of the average IPC w.r.t. naïve t2c



- Speedups around or above 6%
  - 14 mixes with the Dt2c
  - 10 mixes with the St2c
  - 3 mixes with the Linux t2c

Performance evaluation results

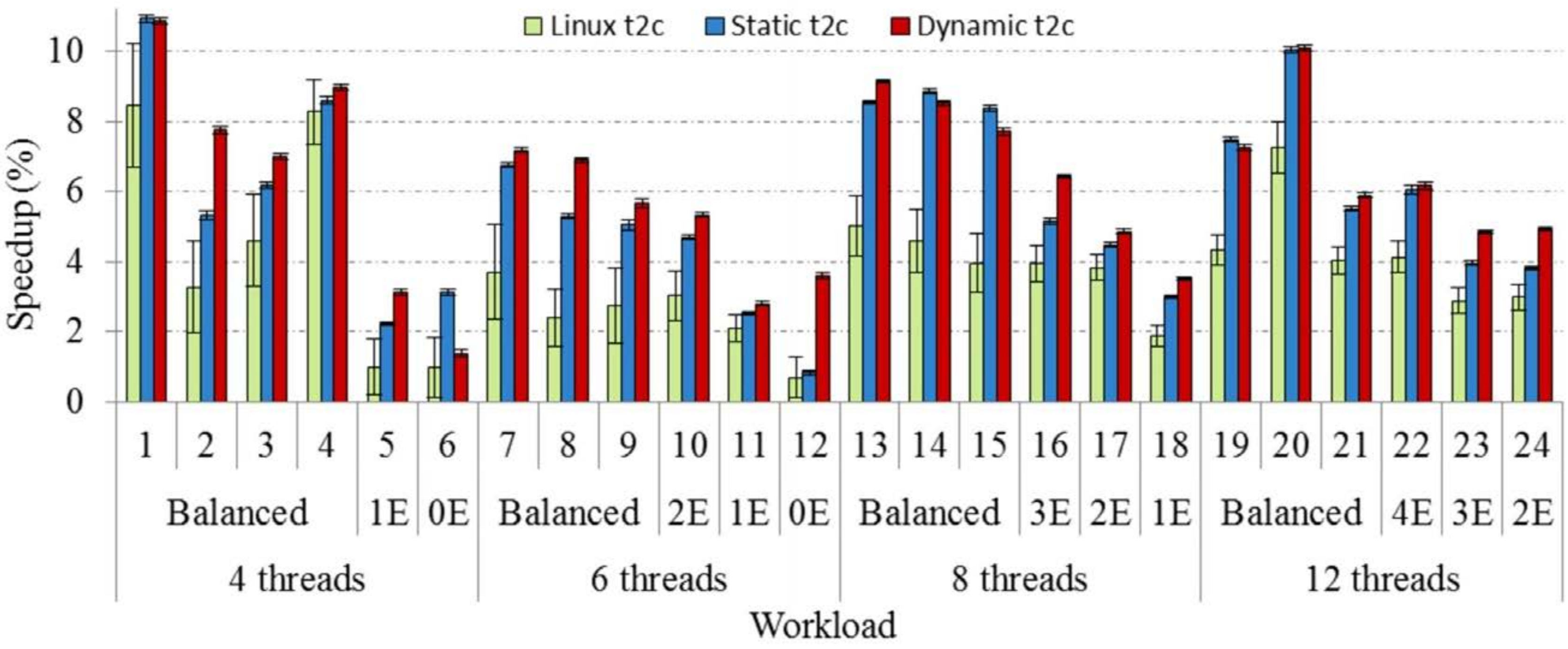
Speedup of the average IPC w.r.t. naïve t2c



- Speedups below 2%
    - 5 mixes with the Linux t2c
    - 1 mix with the St2c
    - 1 mix with the Dt2c

Performance evaluation results

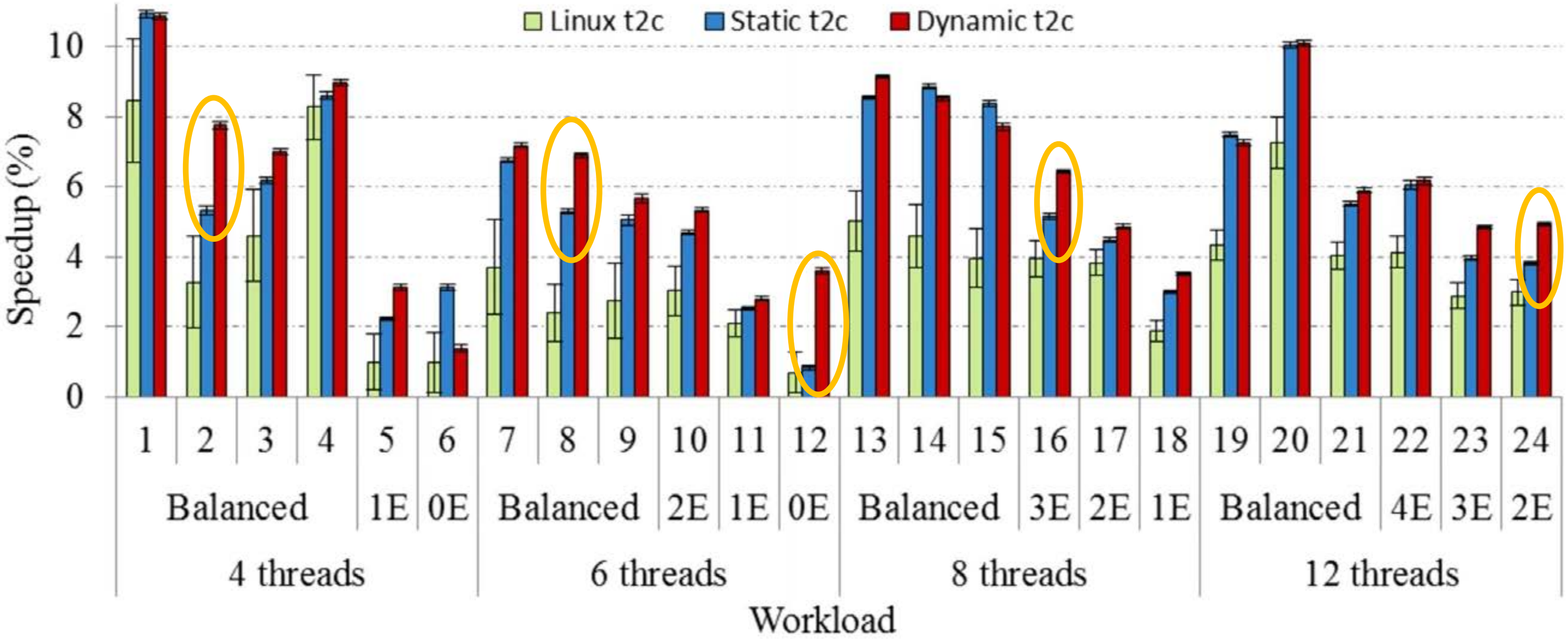
Speedup of the average IPC w.r.t. naïve t2c



- Dt2c performs better on average than St2c
  - Best performance with Dt2c in 19 mixes

Performance evaluation results

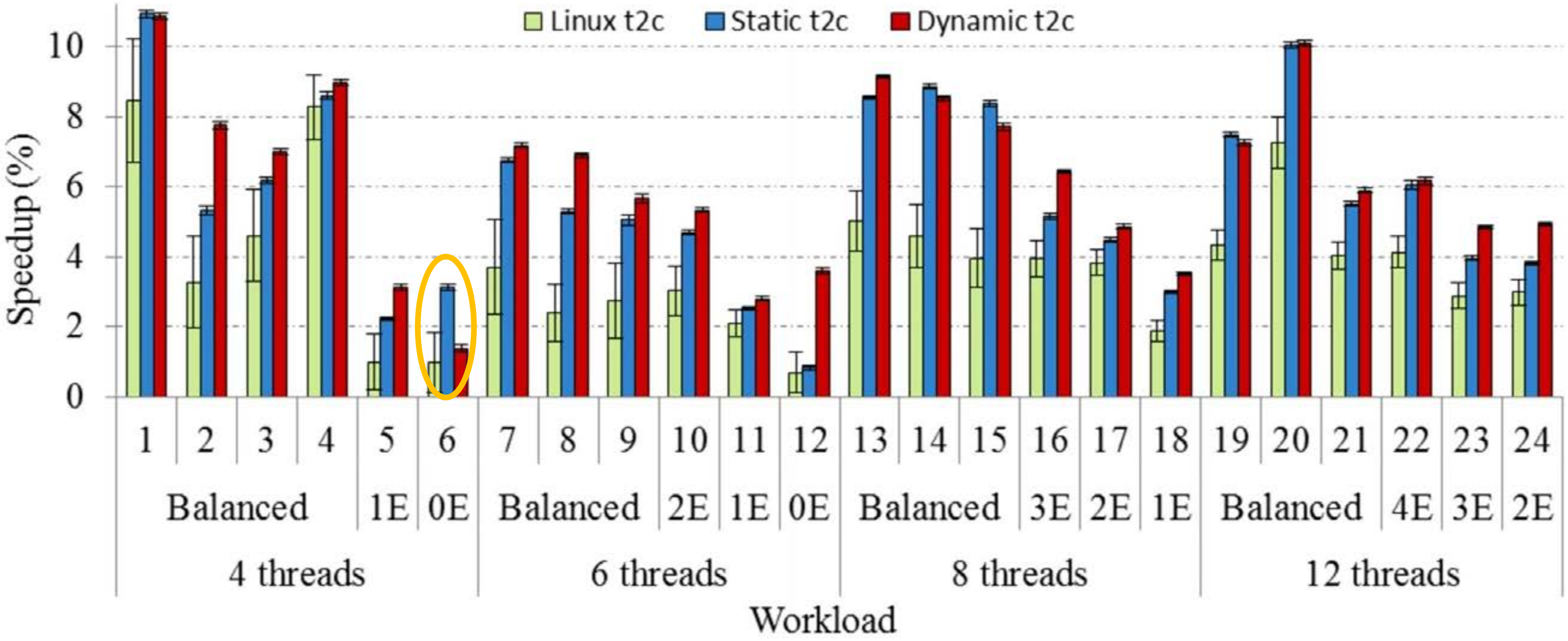
Speedup of the average IPC w.r.t. naïve t2c



- Dt2c performs better on average than St2c
  - Best performance with Dt2c in 19 mixes
- Major differences when the mix includes a wider set of benchmarks with non-uniform shape in the L1 bandwidth

Performance evaluation results

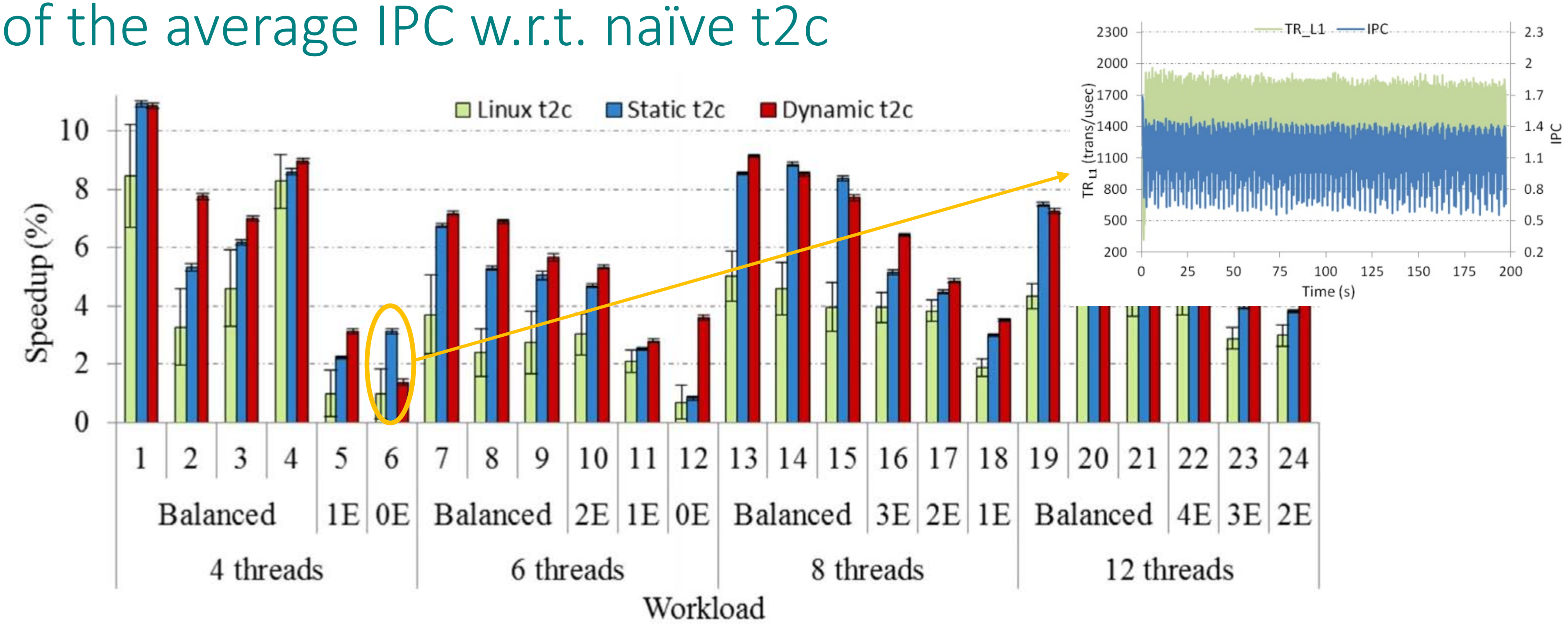
Speedup of the average IPC w.r.t. naïve t2c



- Dt2c performs better on average than St2c
    - Best performance with Dt2c in 19 mixes
- Major differences when the mix includes a wider set of benchmarks with non-uniform shape in the L1 bandwidth
- Major benefit of St2c relative to Dt2c in mix 6

Performance evaluation results

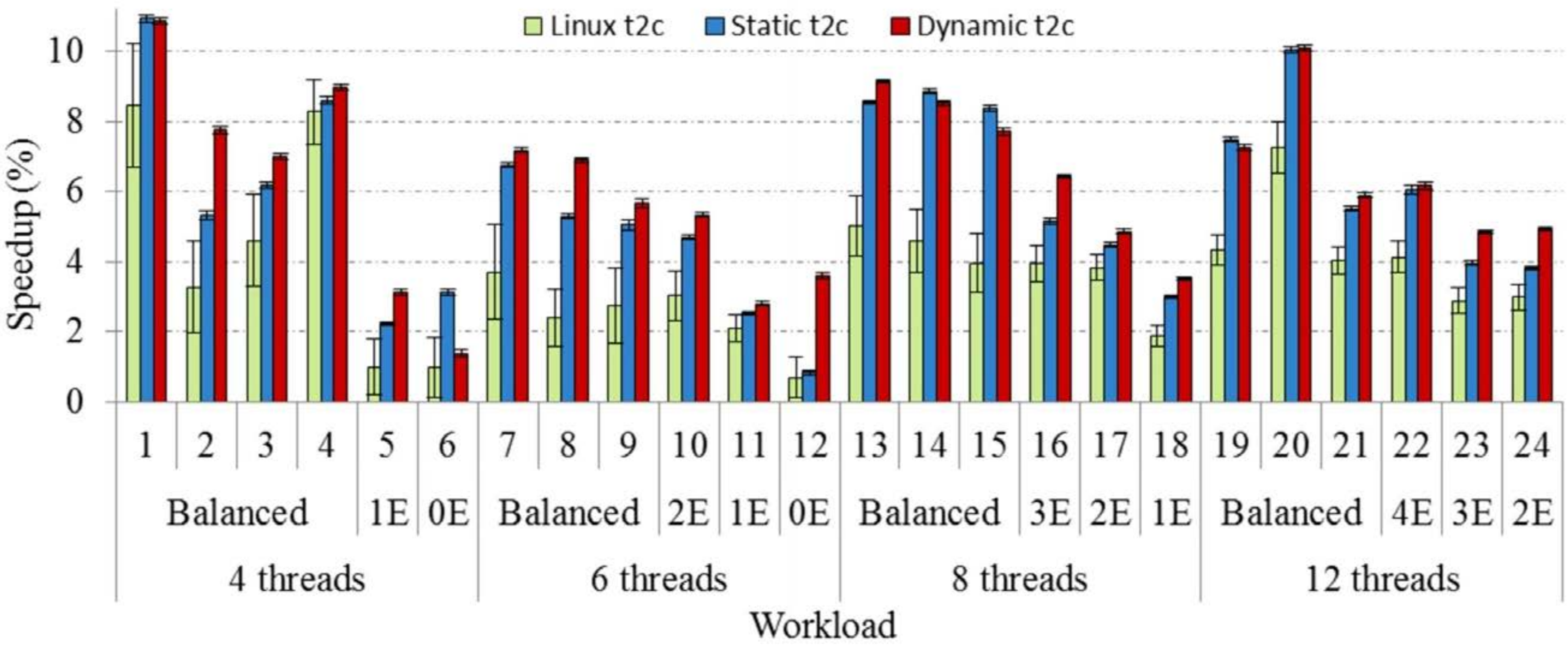
Speedup of the average IPC w.r.t. naïve t2c



- Dt2c performs better on average than St2c
  - Best performance with Dt2c in 19 mixes
- Major differences when the mix includes a wider set of benchmarks with non-uniform shape in the L1 bandwidth
- Major benefit of St2c relative to Dt2c in mix 6
  - Includes *GemsFDTD* benchmarks whose L1 bandwidth demand varies too fast

Performance evaluation results

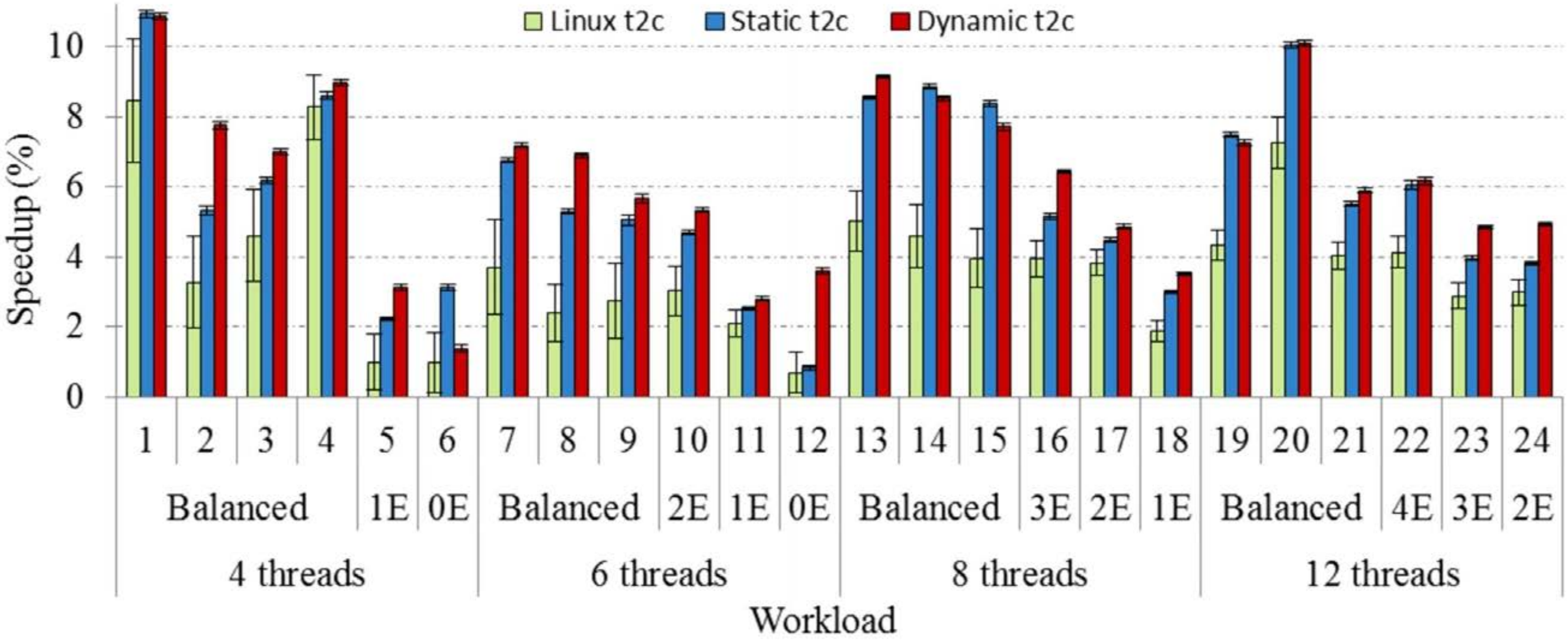
Speedup of the average IPC w.r.t. naïve t2c



- Higher speedups are achieved with balanced mixes

Performance evaluation results

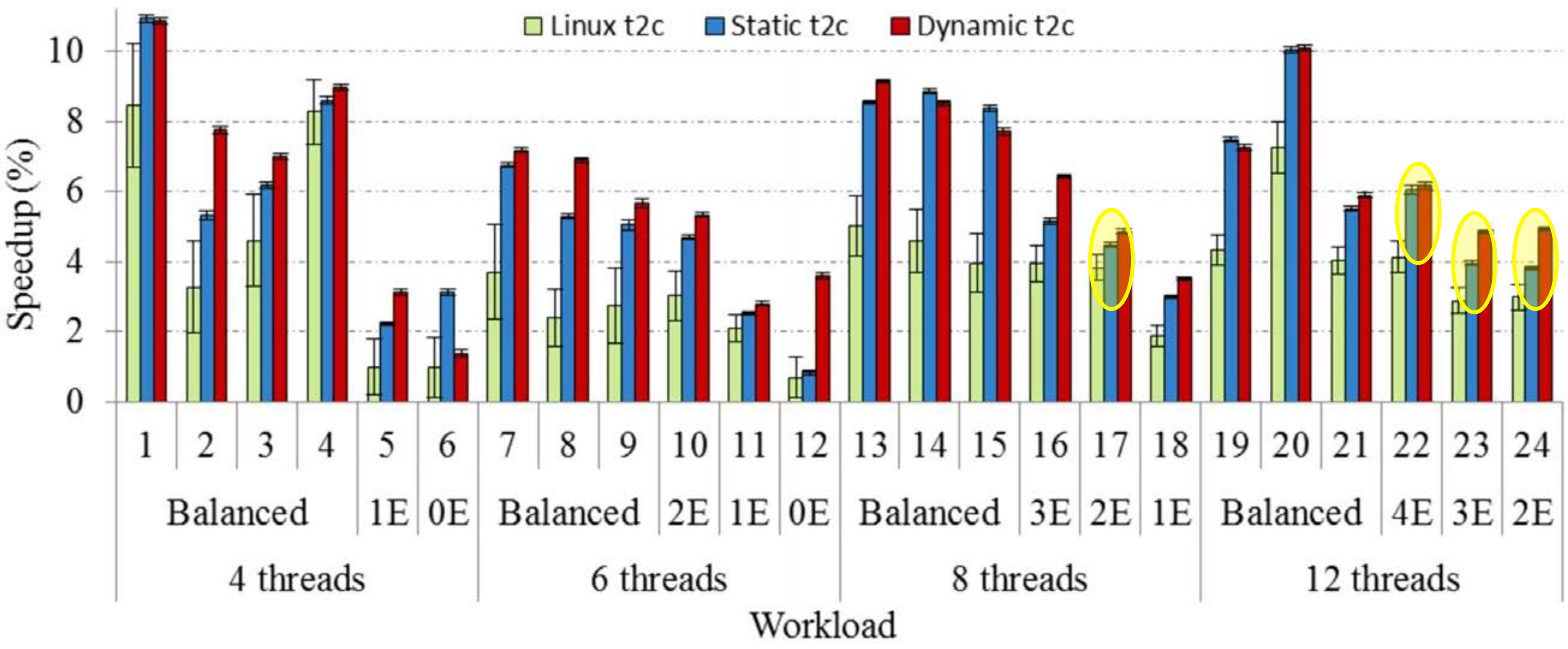
Speedup of the average IPC w.r.t. naïve t2c



- Higher speedups are achieved with balanced mixes
- As the number of benchmarks with extreme L1-bandwidth utilization decrease, the speedups tend to decrease

Performance evaluation results

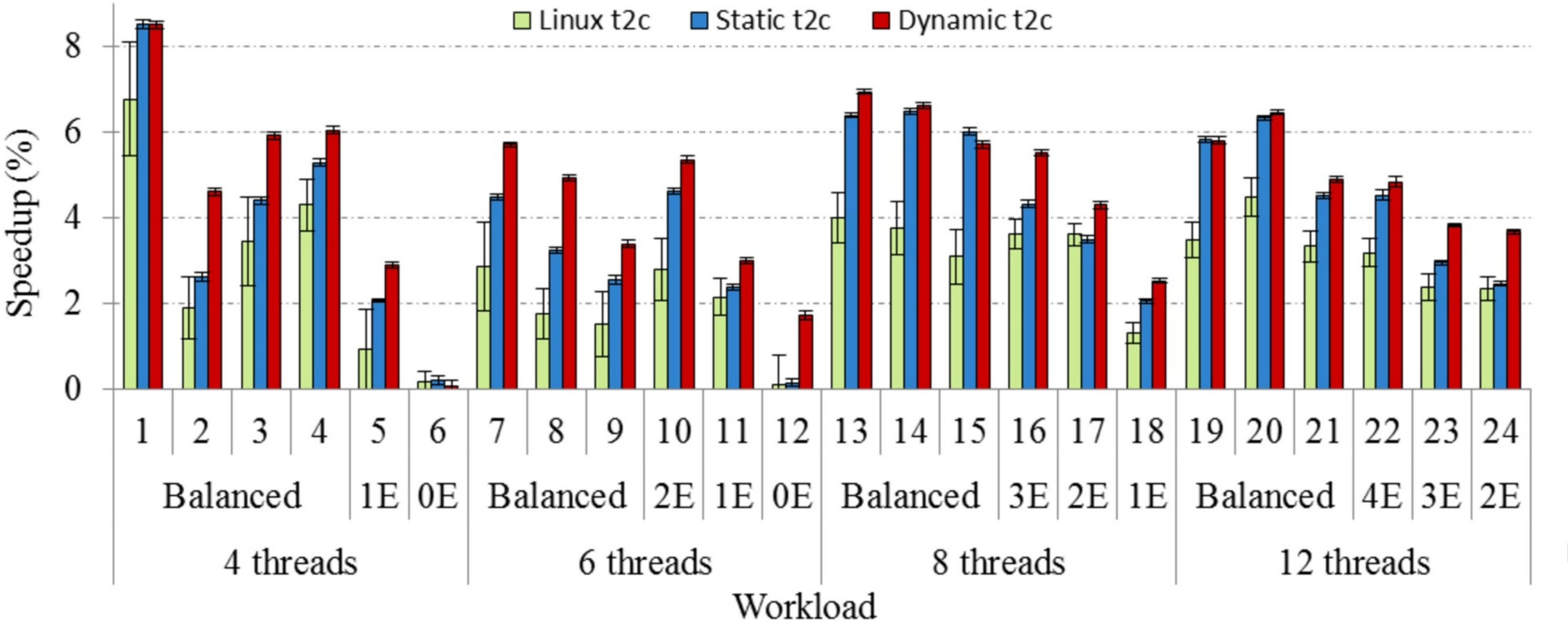
Speedup of the average IPC w.r.t. naïve t2c



- Higher speedups are achieved with balanced mixes
- As the number of benchmarks with extreme L1-bandwidth utilization decrease, the speedups tend to decrease
- Interesting speedups around 5% are observed with 2 benchmarks with extreme L1-bandwidth utilization

Performance evaluation results

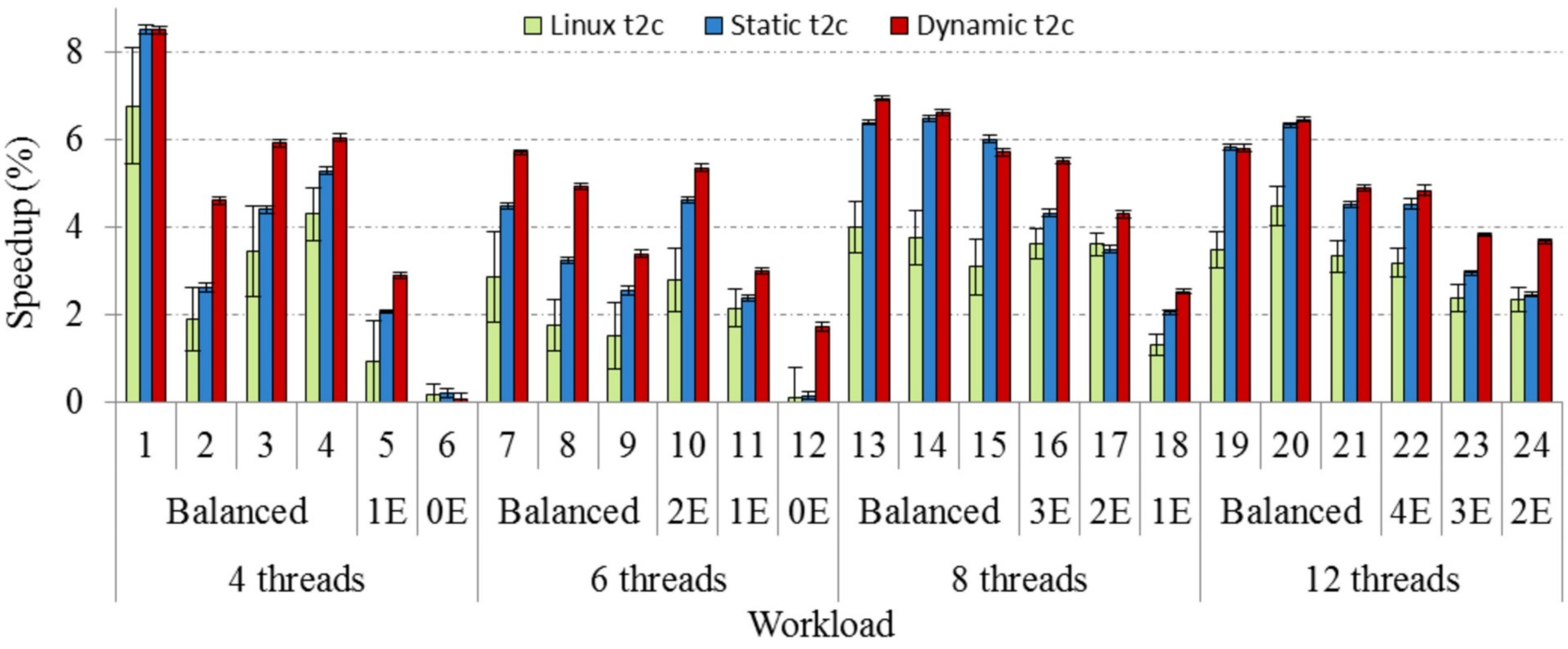
Speedup of the harmonic mean of weighted IPC w.r.t. naïve t2c



- Similar conclusions with the harmonic mean of weighted IPC metric

Performance evaluation results

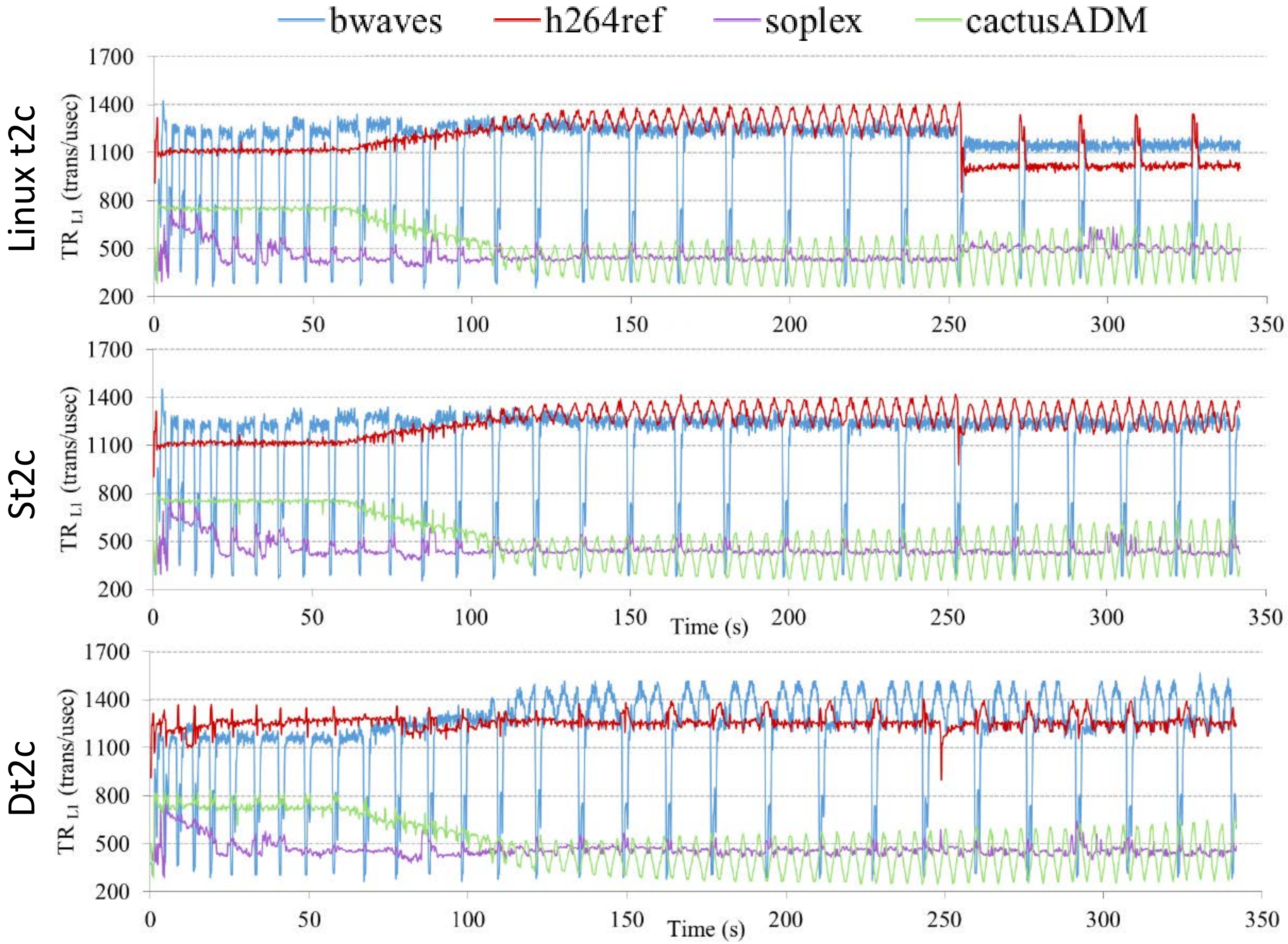
Speedup of the harmonic mean of weighted IPC w.r.t. naïve t2c



- Similar conclusions with the harmonic mean of weighted IPC metric
  - The speedups of the policies are slightly reduced relative to the naïve policy
  - The differences between the Dt2c policy and St2c policy are increased
- The Dt2c policy is the best one, since it improves the other policies in performance and fairness

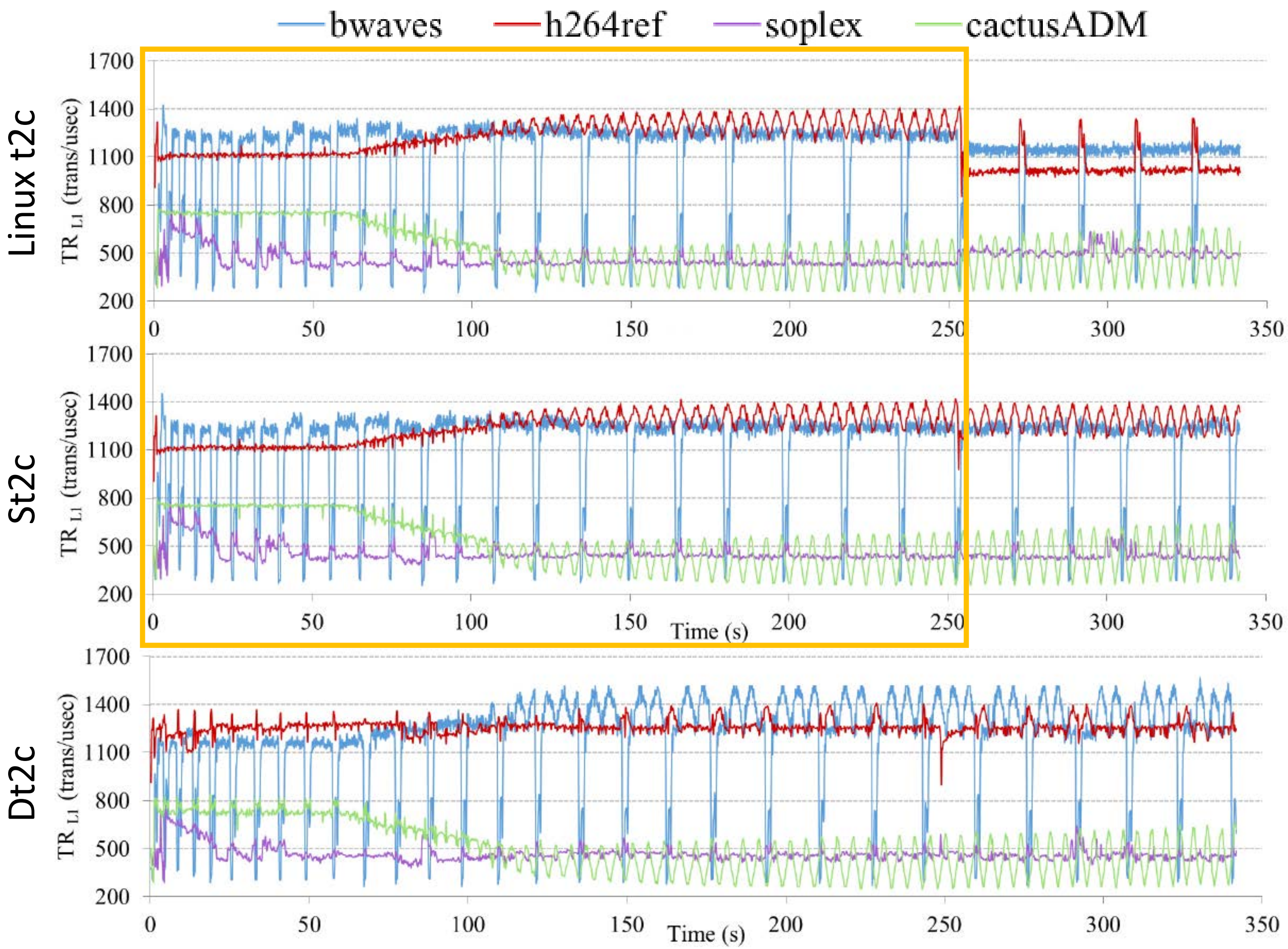
Performance evaluation results

Dynamic L1 bandwidth on mix 2



Performance evaluation results

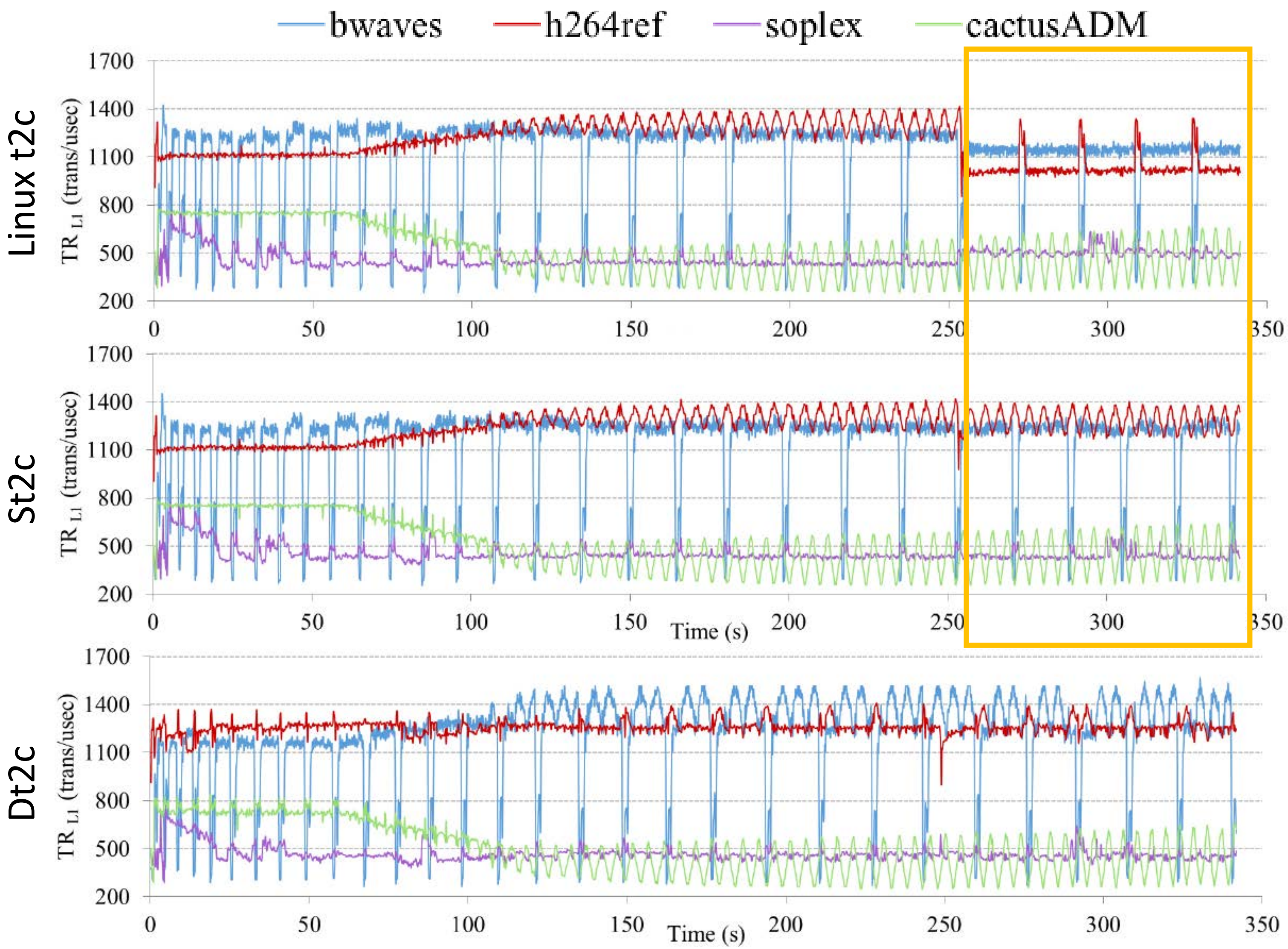
Dynamic L1 bandwidth on mix 2



- Similar plots for Linux and St2c during the first 250 seconds
- Thread to core mapping
  - *h264ref* and *cactusADM*
  - *bwaves* and *soplex*

Performance evaluation results

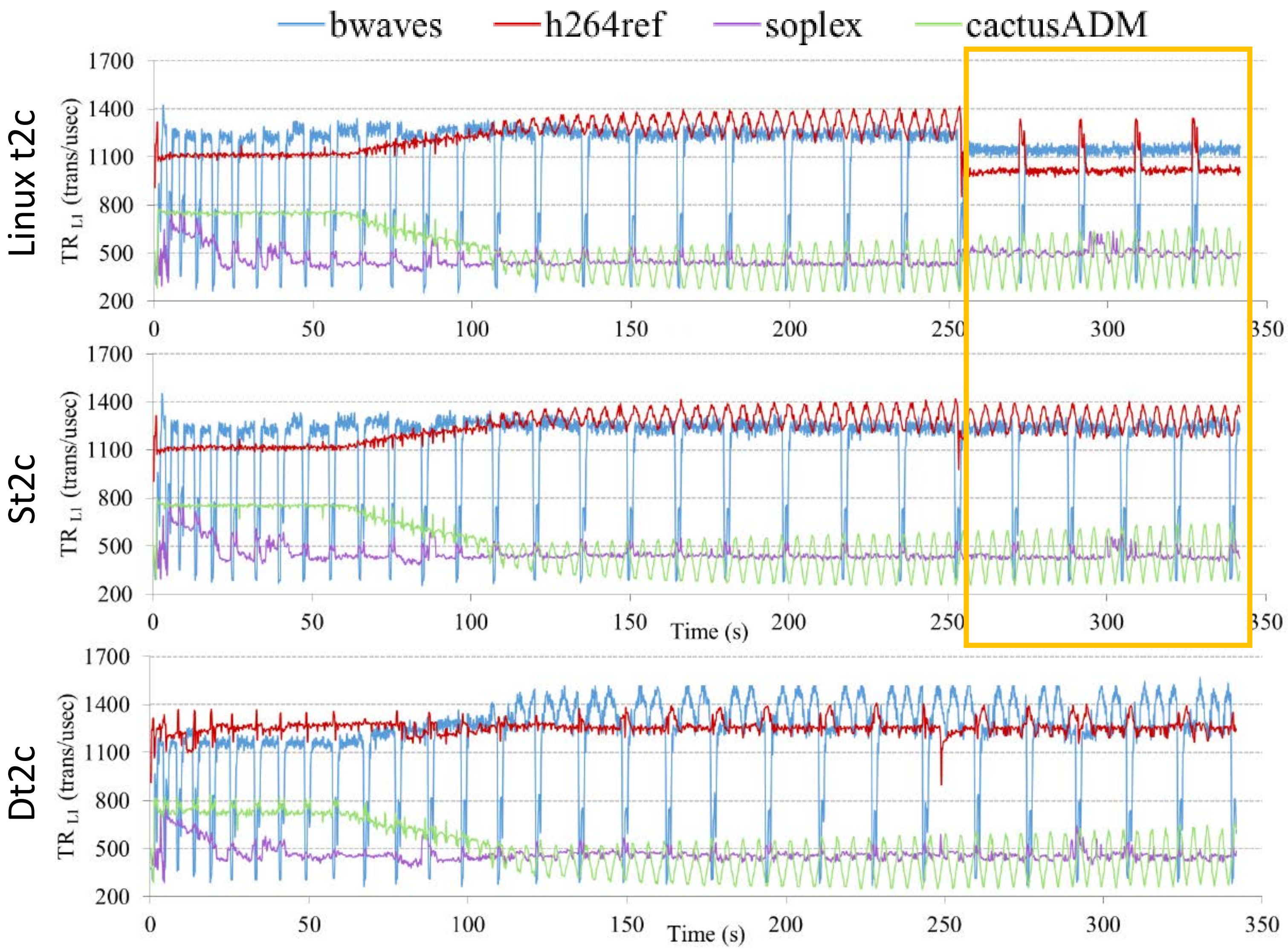
Dynamic L1 bandwidth on mix 2



- Similar plots for Linux and St2c during the first 250 seconds
- Thread to core mapping
  - *h264ref* and *cactusADM*
  - *bwaves* and *soplex*
- Around second 250, Linux t2c policy changes the t2c mapping
  - *bwaves* and *h264ref* share a core

Performance evaluation results

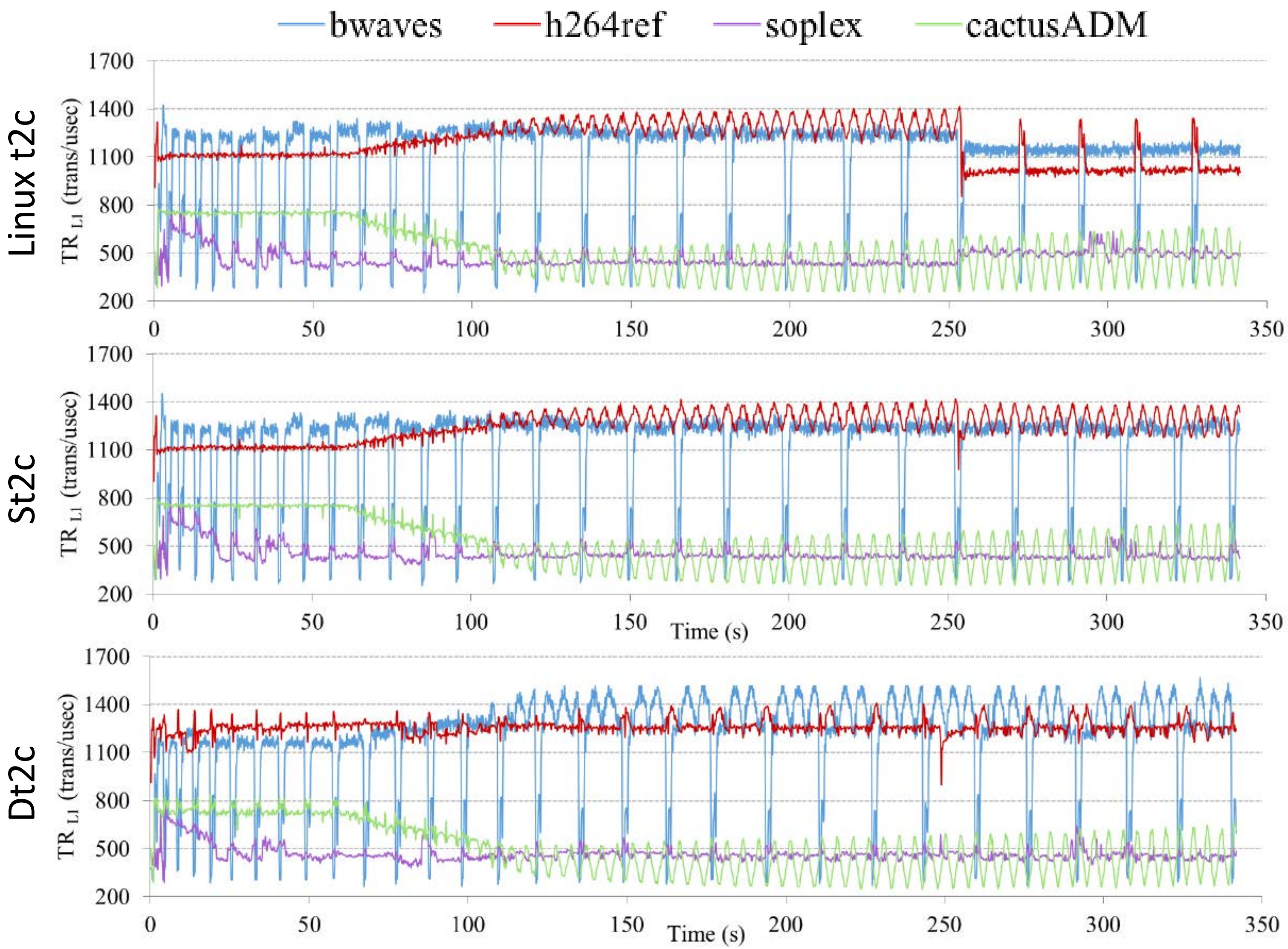
Dynamic L1 bandwidth on mix 2



- Similar plots for Linux and St2c during the first 250 seconds
- Thread to core mapping
  - *h264ref* and *cactusADM*
  - *bwaves* and *soplex*
- Around second 250, Linux t2c policy changes the t2c mapping
  - *bwaves* and *h264ref* share a core
  - It achieves lower performance, but Linux keeps using it until the end of the execution

Performance evaluation results

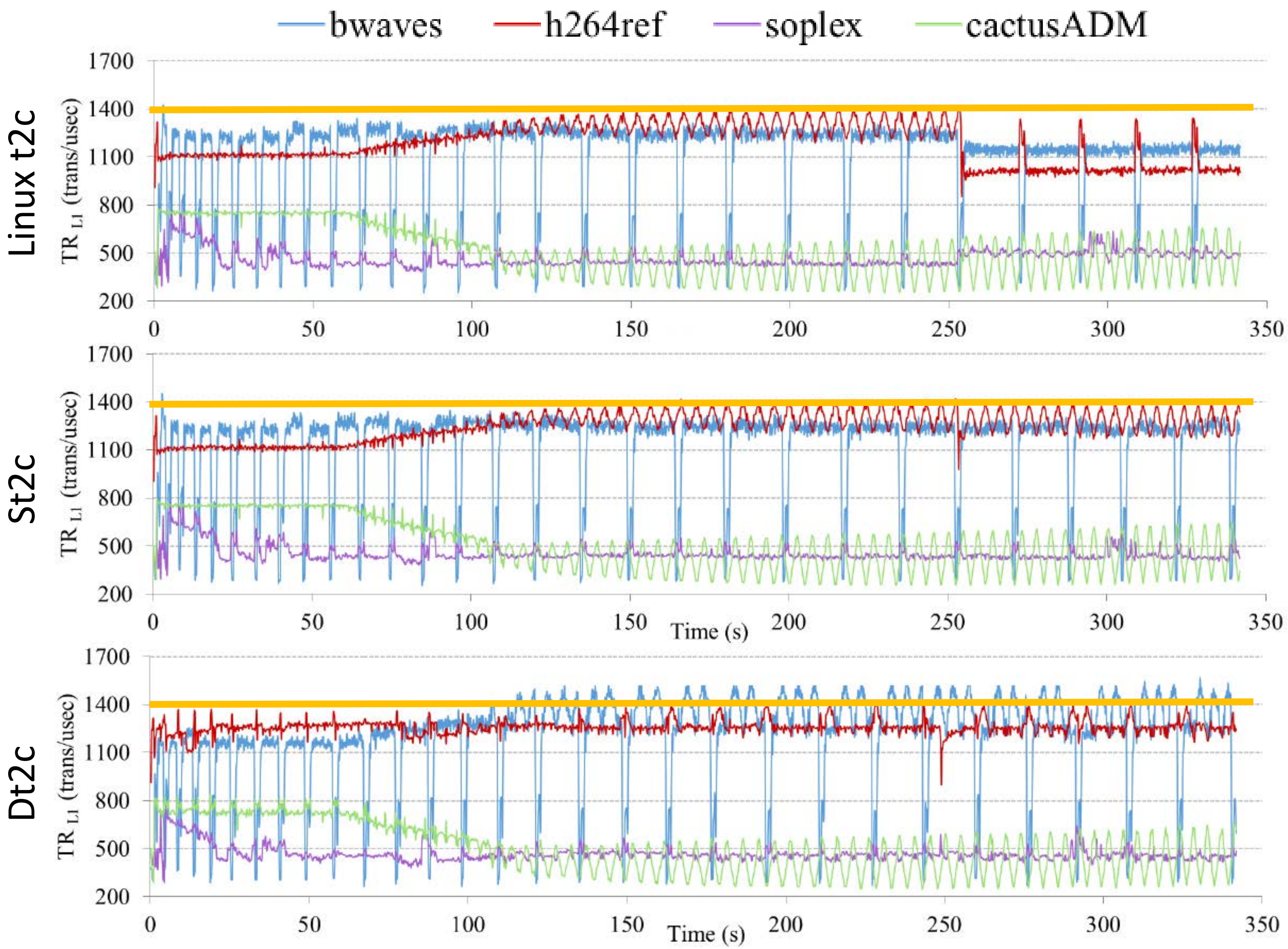
Dynamic L1 bandwidth on mix 2



- Similar plots for Linux and St2c during the first 250 seconds
- Thread to core mapping
  - *h264ref* and *cactusADM*
  - *bwaves* and *soplex*
- Around second 250, Linux t2c policy changes the t2c mapping
  - *bwaves* and *h264ref* share a core
  - It achieves lower performance, but Linux keeps using it until the end of the execution
- Dt2c usually allocates
  - *bwaves* and *cactusADM* on the same core

Performance evaluation results

Dynamic L1 bandwidth on mix 2



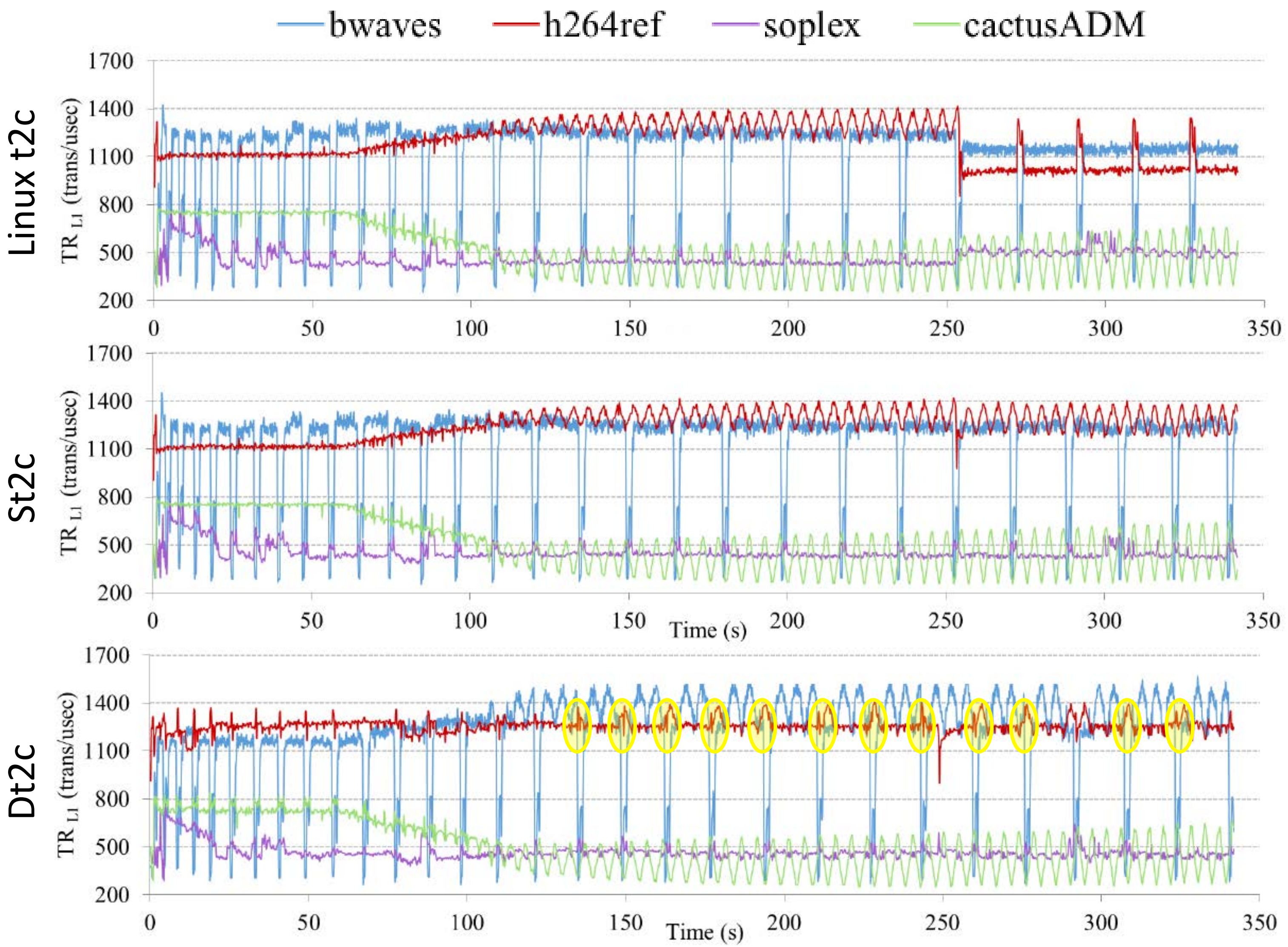
- Similar plots for Linux and St2c during the first 250 seconds
- Thread to core mapping
  - *h264ref* and *cactusADM*
  - *bwaves* and *soplex*

- Around second 250, Linux t2c policy changes the t2c mapping
  - *bwaves* and *h264ref* share a core
  - It achieves lower performance, but Linux keeps using it until the end of the execution

- Dt2c usually allocates
  - *bwaves* and *cactusADM* on the same core, what brings better performance

Performance evaluation results

Dynamic L1 bandwidth on mix 2



- Similar plots for Linux and St2c during the first 250 seconds
- Thread to core mapping
  - *h264ref* and *cactusADM*
  - *bwaves* and *soplex*
- Around second 250, Linux t2c policy changes the t2c mapping
  - *bwaves* and *h264ref* share a core
  - It achieves lower performance, but Linux keeps using it until the end of the execution
- Dt2c usually allocates
  - *bwaves* and *cactusADM* on the same core, what brings better performance
  - *h264ref* runs with *cactusADM* in the drops of *bwaves*, showing peaks in its L1 bandwidth

Outline

- Introduction
- Experimental platform
- Effects of L1 bandwidth on performance of SMT processors
- L1-bandwidth aware thread allocation policies
- Evaluation methodology
- Performance evaluation results
- Conclusions

Conclusions

- L1 bandwidth contention in current multithreaded CMPS has been addressed

Conclusions

- L1 bandwidth contention in current multithreaded CMPS has been addressed
- Our work has shown:
  - Strong connection between L1 bandwidth and performance

Conclusions

- L1 bandwidth contention in current multithreaded CMPS has been addressed
- Our work has shown:
  - Strong connection between L1 bandwidth and performance
  - L1 bandwidth insufficient to satisfy the requirements of two threads

Conclusions

- L1 bandwidth contention in current multithreaded CMPS has been addressed

- Our work has shown:
  - Strong connection between L1 bandwidth and performance
  - L1 bandwidth insufficient to satisfy the requirements of two threads
  - Rises and drops in the L1 bandwidth (and performance) of a thread trigger opposite behavior in the co-runner

Conclusions

- L1 bandwidth contention in current multithreaded CMPS has been addressed

- Our work has shown:
  - Strong connection between L1 bandwidth and performance
  - L1 bandwidth insufficient to satisfy the requirements of two threads
  - Rises and drops in the L1 bandwidth (and performance) of a thread trigger opposite behavior in the co-runner

- To leverage the finding, two thread allocation strategies have been proposed
  - Goal: balancing the L1 bandwidth among all the L1 caches

Conclusions

- L1 bandwidth contention in current multithreaded CMPS has been addressed
- Our work has shown:
  - Strong connection between L1 bandwidth and performance
  - L1 bandwidth insufficient to satisfy the requirements of two threads
  - Rises and drops in the L1 bandwidth (and performance) of a thread trigger opposite behavior in the co-runner
- To leverage the finding, two thread allocation strategies have been proposed
  - Goal: balancing the L1 bandwidth among all the L1 caches
- The proposed policies outperform the Linux thread allocation policy in both performance and fairness

Thank you
&
¿Questions?

PACT 2013
The 22nd International Conference on Parallel Architectures and Compilation Techniques

L1-Bandwidth Aware Thread Allocation in Multicore SMT Processors

J. Feliu, J. Sahuquillo, S. Petit and J. Duato

Universitat Politècnica de València

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

9 September 2013
Edinburgh, United Kingdom

Backup slices

Evaluation methodology

Benchmark classification

- Benchmarks are classified in four categories
  - According to their L1 bandwidth

Classification	Benchmarks
Extreme L1 bandwidth	h264ref, bwaves, gamess
High L1 bandwidth	perlbench, bzip2, hmmer, libquantum, leslie3d, namd, dealII, gemsFDTD
Medium L1 bandwidth	gcc, gobmk, sjeng, astar, xalancbmk, zeusMP, povray, lbm
Low L1 bandwidth	mcf, omnetpp, milc, gromacs, cactusADM, soplex

Table 1. Mix classification

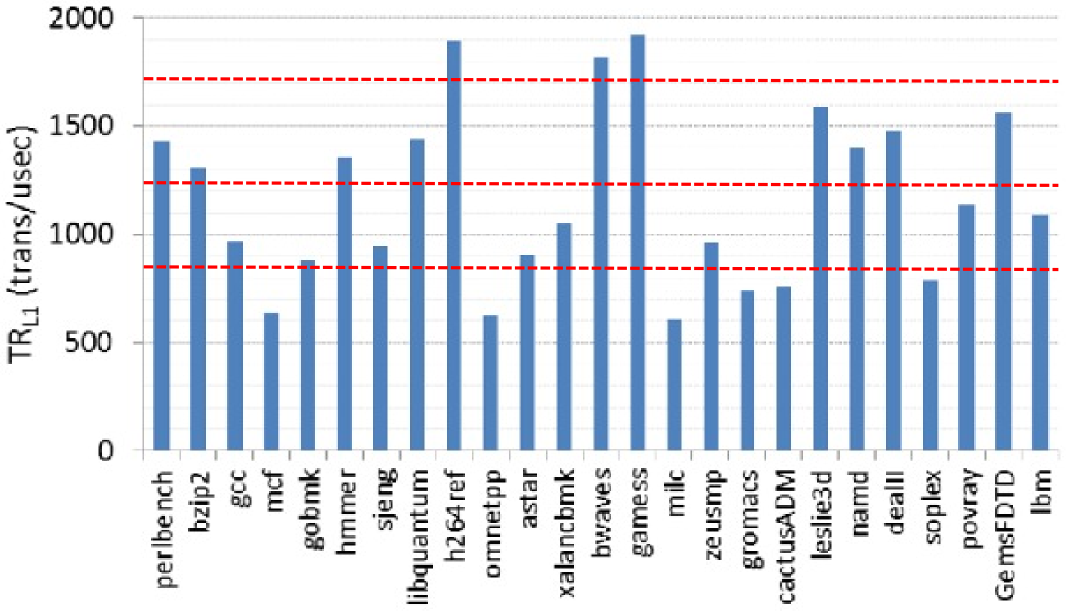
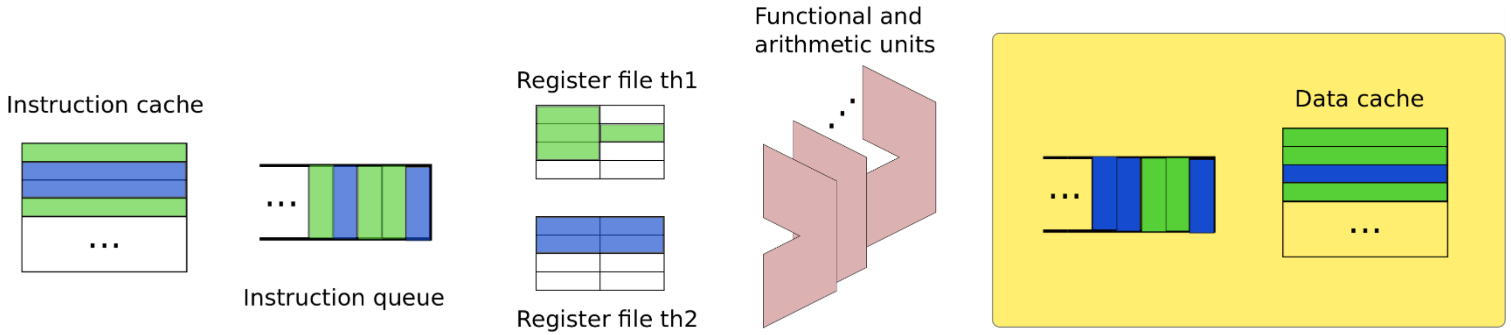


Fig 1. Average TR$_{L1}$ for SPEC CPU 2006 benchmarks

Introduction

- A critical shared resource in any CMP is the memory bandwidth
  - Main memory bandwidth
  - LLC bandwidth
  - Bandwidth at any shared cache
- Addressed with bandwidth-aware schedulers
- L1 caches are private to cores, but shared among threads in SMT cores
  - ➢ **L1 bandwidth contention may impact the performance**


Instruction cache
…
…
Instruction queue
Register file th1
Register file th2
Functional and arithmetic units
…
Data cache
…