A Parallel Data Distribution Management Algorithm

Gabriele D’Angelo

<g.dangelo@unibo.it>

http://www.cs.unibo.it/gdangelo/

joint work with:

Moreno Marzolla** and **Marco Mandrioli

Delft, Netherlands

Distributed **S**imulation and **R**eal **T**ime Applications (**DS-RT**), 2013

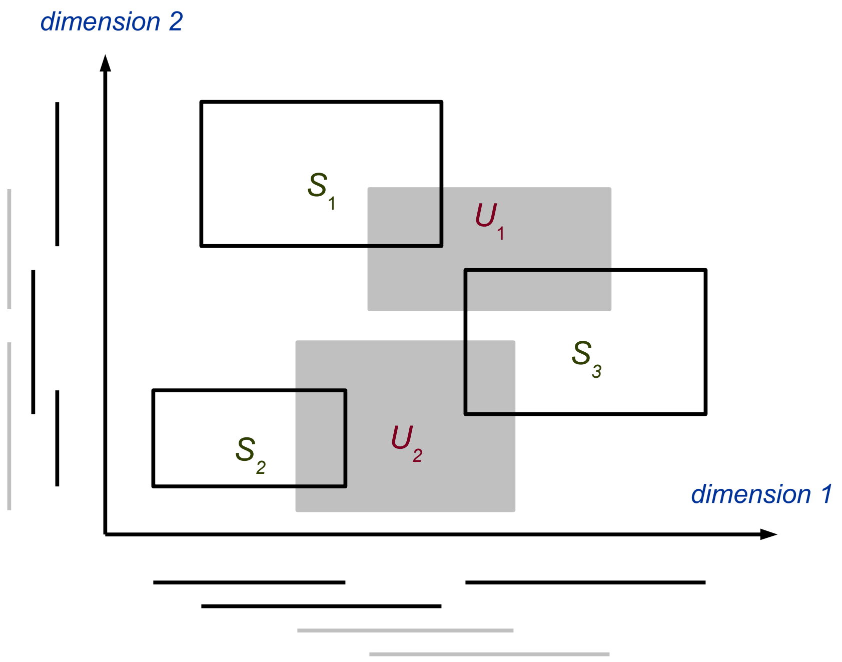
Presentation **outline**

- **D**ata **D**istribution **M**anagement (**DDM**)
- **DDM** algorithms
- **Parallel** Data Distribution Management
- Parallel Matching Algorithms
- **I**nterval **T**ree **M**atching (**ITM**)
- Experimental Evaluation
- **Conclusions**

Data **D**istribution **M**anagement (**DDM**)

- **DDM** services are part of the **IEEE 1516** "*High Level Architecture*" (HLA) specification
- **DDM** is about forwarding events generated on ***update* regions** to a set of ***subscription* regions**
- A **region** is a *d*-dimensional rectangle (*d*-rectangle) in a *d*-dimensional routing space
- The goal is to find all the couples of update and subscription regions **that overlap**
- We assume that each **region** corresponds to a single **extent**

DDM: simple example in **2 dimensions**



- The problem of identifying whether two *d*-rectangles intersect can be reduced to ***d*** independent intersection problems among one-dimensional segments

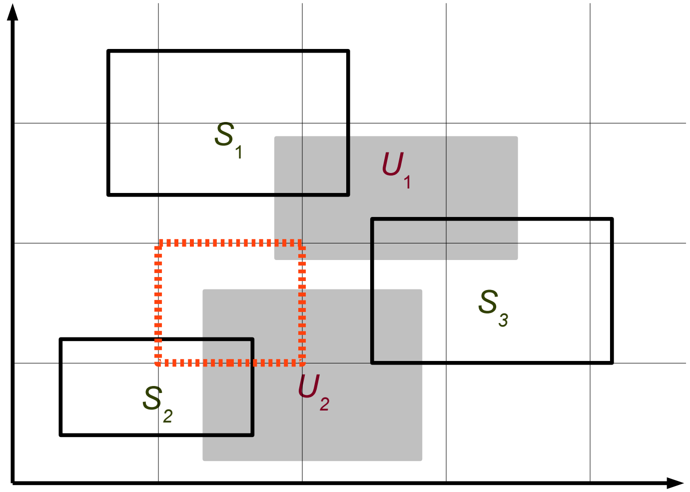
- Two d-rectangles intersect **if and only if** they **intersect among all their projections**

DDM algorithms: from simple to complex

- **No DDM**: all updates are broadcasted, the filtering is on receivers. **CONS:** communication inefficiency

- **Region-based matching** (**Brute Force**, **BF**): it tests all the subscription-update pairs. **CONS:** computational inefficiency

- **Grid-based matching**: it partitions the routing space into a grid of *d*-dimensional cells and finds matches



CONS:

- false positives
- which cell size is the best?

DDM algorithms: from simple to complex

- **Sort-Based Matching** (**SBM**): before matching, the extents are sorted. In this way, the overlaps can be found efficiently. In many cases **SBM** does better than **grid-based** [Raczy et al. 2005]. **CONS:** are there drawbacks?

- **Binary partition-based**: recursive binary partitioning of the extents. **CONS:** (more than) quadratic worst case cost

DDM algorithms: from simple to complex

- **Sort-Based Matching** (**SBM**): before matching, the extents are sorted. In this way, the overlaps can be found efficiently. In many cases **SBM** does better than **grid-based** [Raczy et al. 2005]. **CONS:** are there drawbacks?

- **Binary partition-based**: recursive binary partitioning of the extents. **CONS:** (more than) quadratic worst case cost

- … and so, **what's the problem?**

DDM algorithms: from simple to complex

- **Sort-Based Matching** (**SBM**): before matching, the extents are sorted. In this way, the overlaps can be found efficiently. In many cases **SBM** does better than **grid-based** [Raczy et al. 2005]. **CONS:** are there drawbacks?

- **Binary partition-based**: recursive binary partitioning of the extents. **CONS:** (more than) quadratic worst case cost

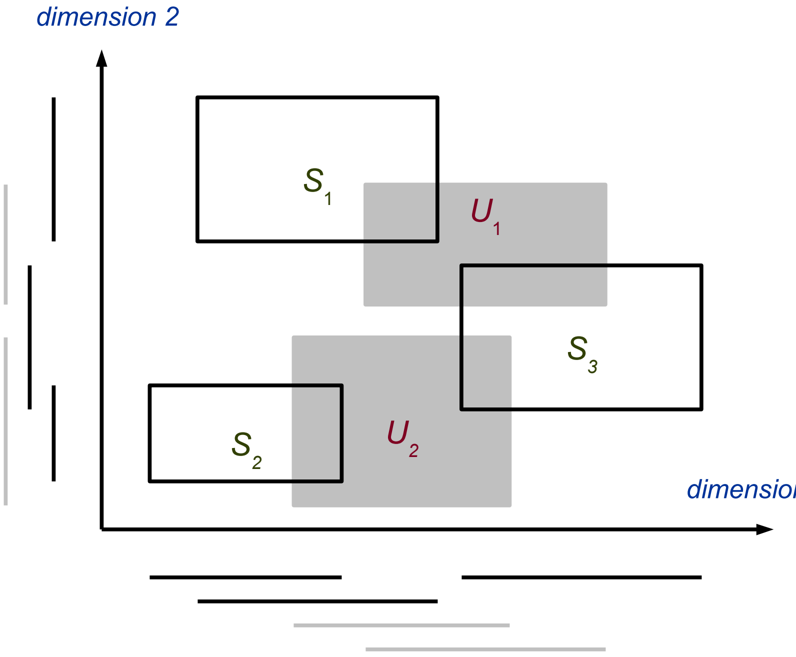
- … and so, **what's the problem?**

- Even smartphones are **multi-core** and we're still dealing with **sequential algorithms**

- The future of computing is very likely to be **many-core**

Parallel Data Distribution Management

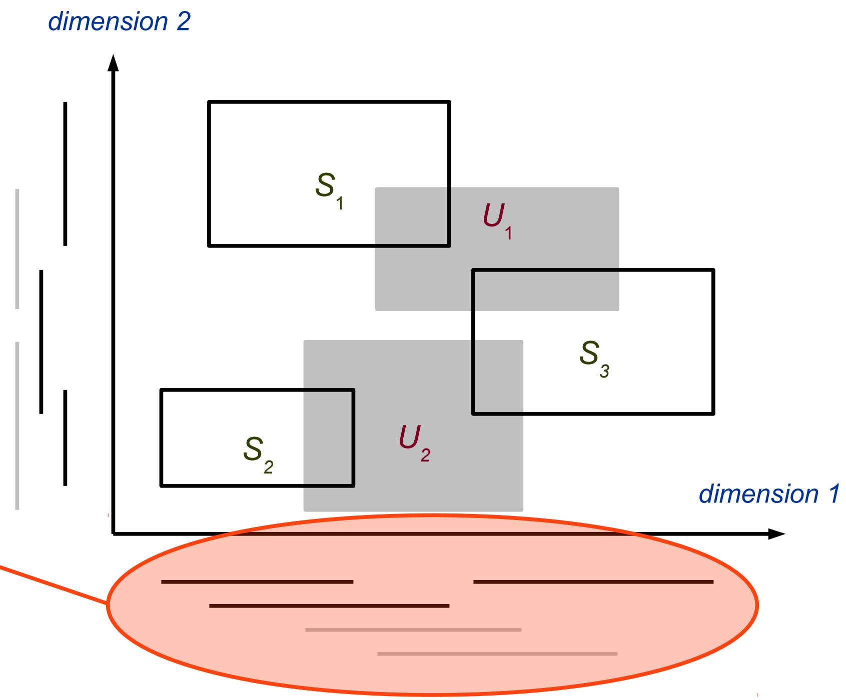
- **Easy solution**: each core (or CPU) is responsible of a specific dimension. The "master" core computes the final result



Parallel Data Distribution Management

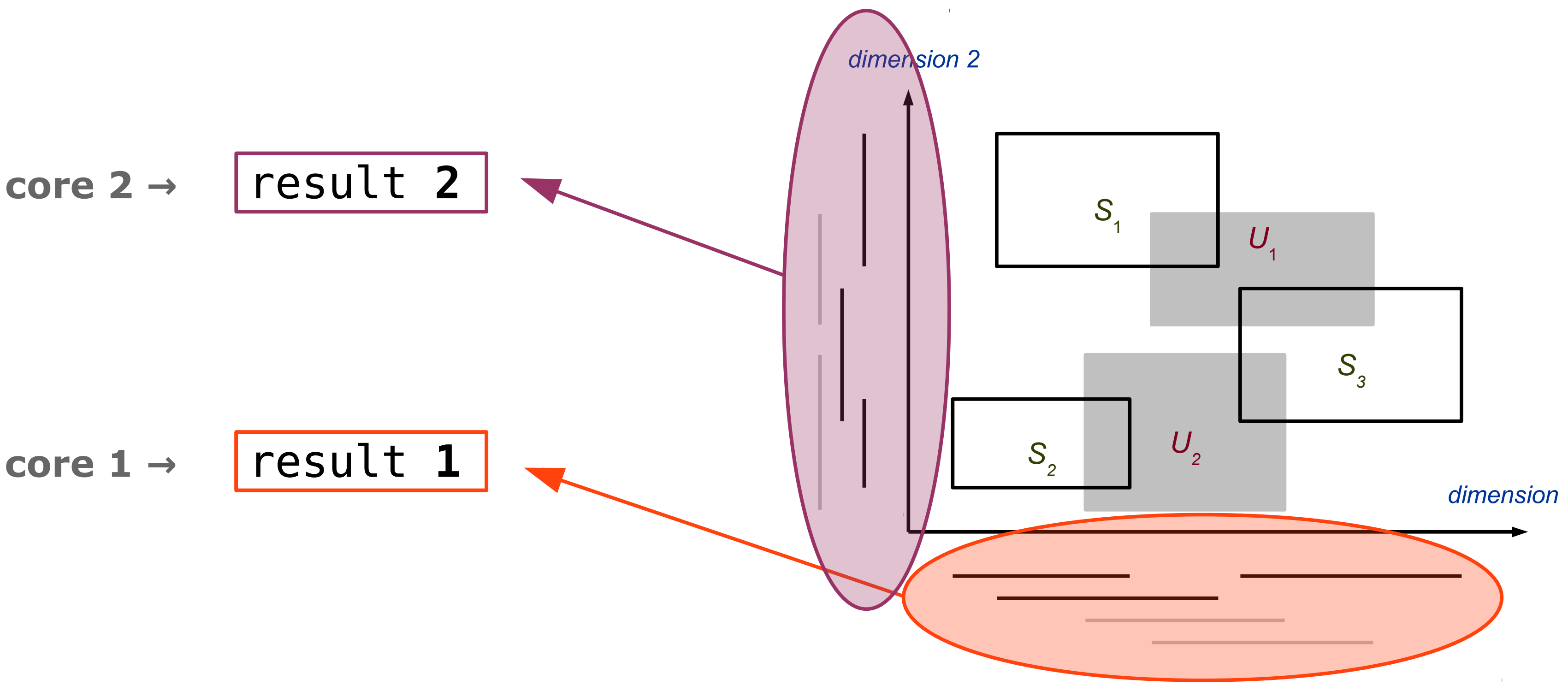
- **Easy solution**: each core (or CPU) is responsible of a specific dimension. The "master" core computes the final result

core 1 → $\boxed{\texttt{result 1}}$



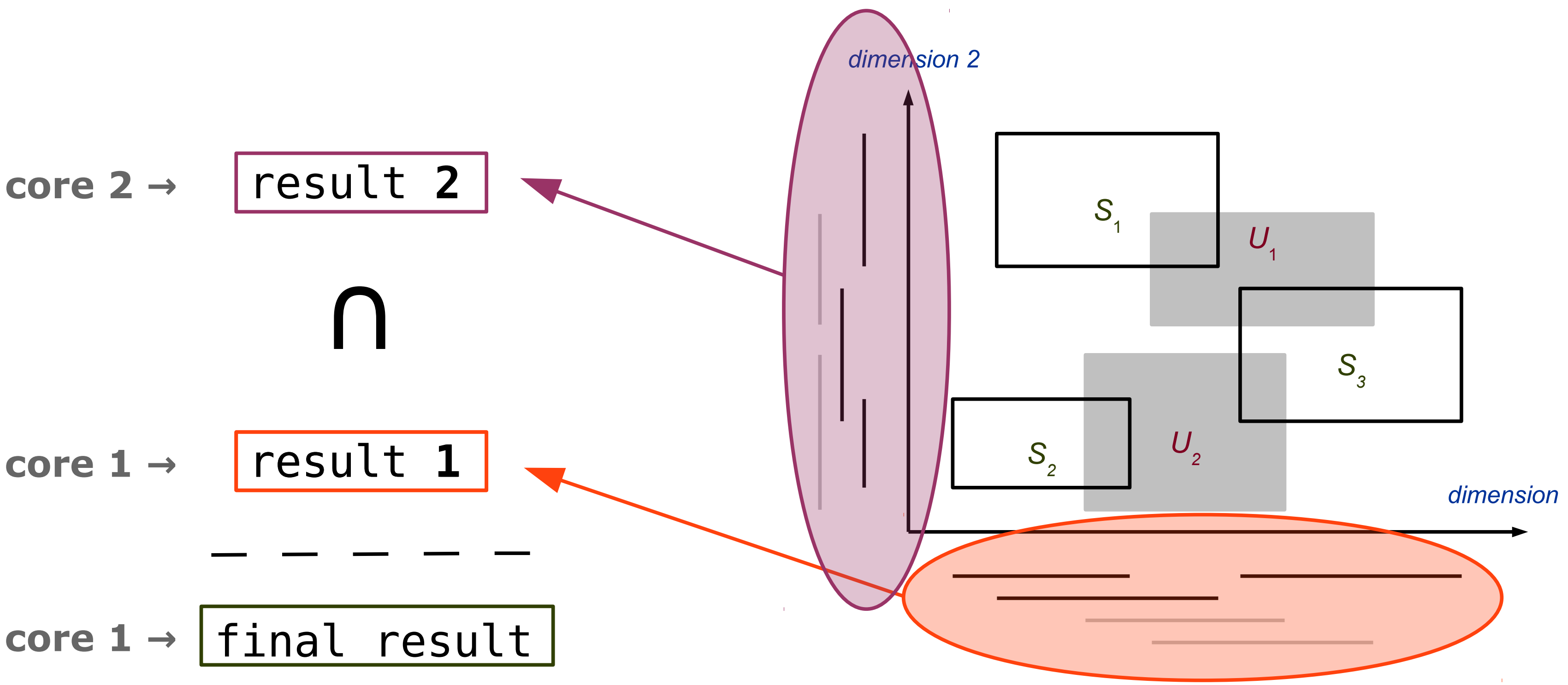
Parallel Data Distribution Management

- **Easy solution**: each core (or CPU) is responsible of a specific dimension. The "master" core computes the final result



Parallel Data Distribution Management

- **Easy solution**: each core (or CPU) is responsible of a specific dimension. The "master" core computes the final result



Parallel **D**ata **D**istribution **M**anagement

- **Easy solution**: each core (or CPU) is responsible of a specific dimension. The "master" core computes the final result
  - **PRO**: it can be applied to every DDM algorithm
  - **CONS**: what if the **number of cores**/CPUs is **larger** than the **number of dimensions**?

This solution is **not adequate** for "many core" environments

We need **parallel matching algorithms**

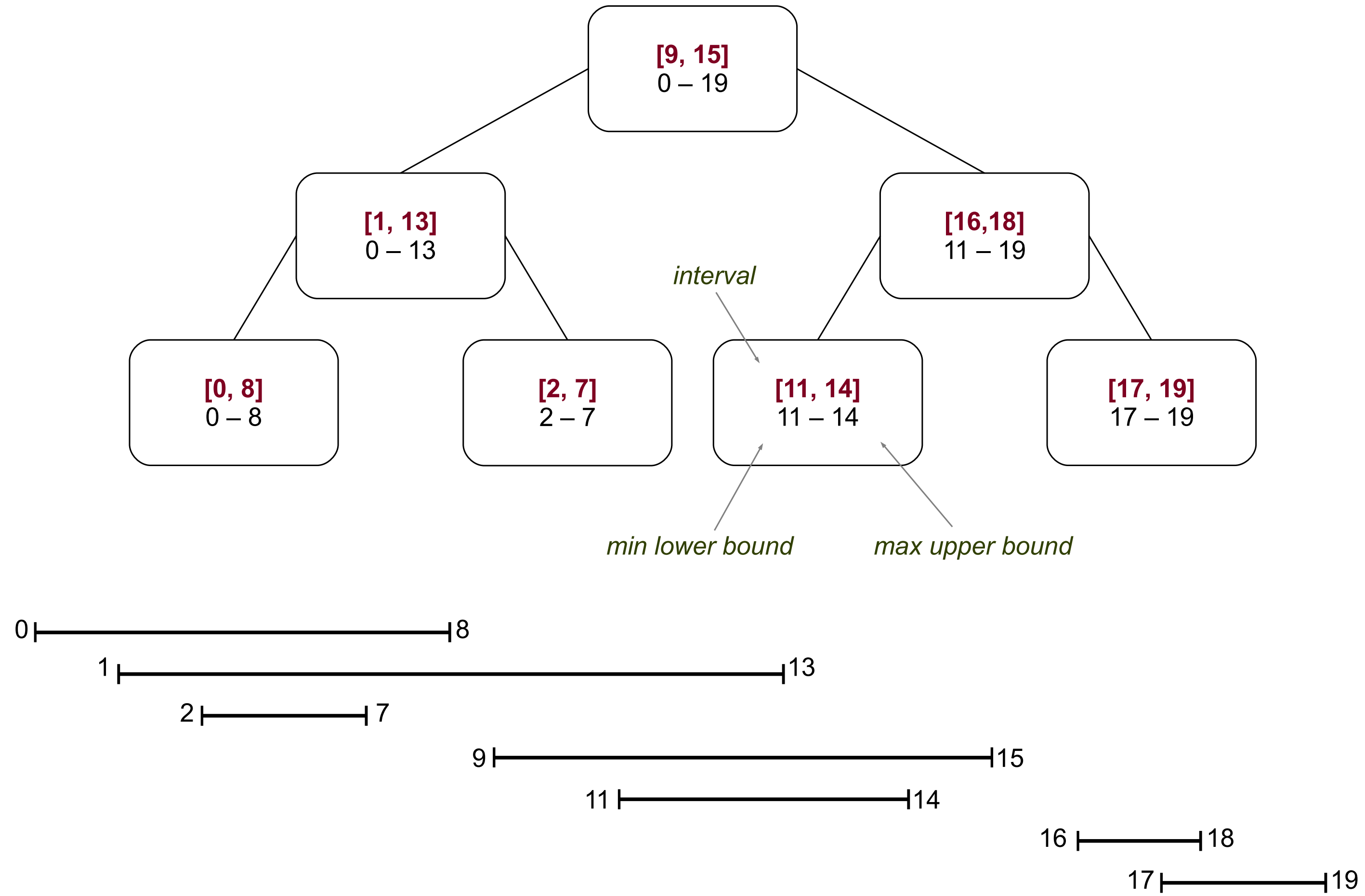
Parallel matching algorithms

- **Brute Force** (**BF**): inefficient, embarrassingly parallel
- **Sort-Based Matching** (**SBM**): efficient, not easy to parallelize
- **DESIDERATA**:
  - **efficient**
  - **easy** to parallelize
  - based on a **simple** data structure
  - that allows the efficient update/add/delete of extents

    (**dynamic interval management**)

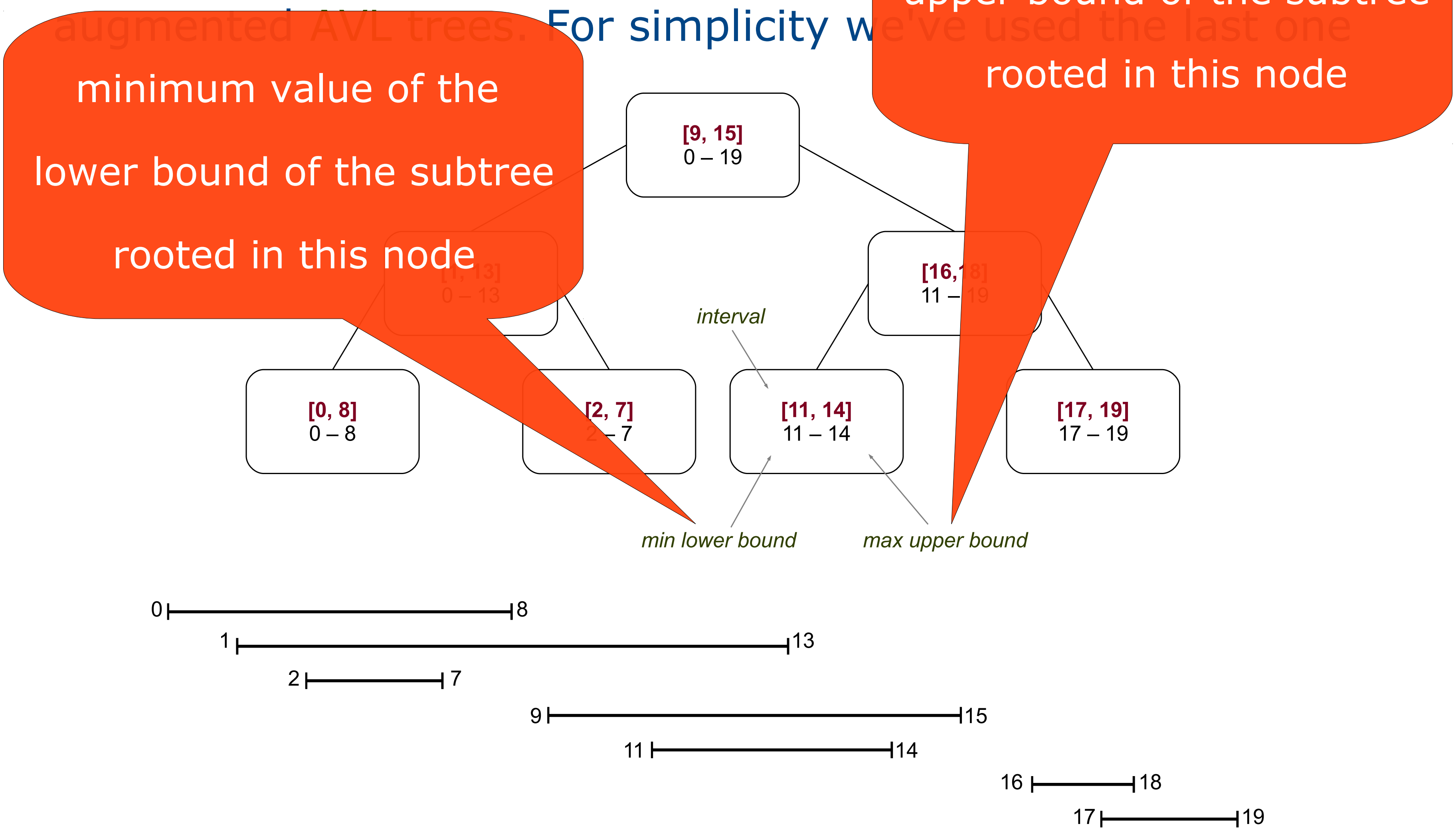
Our proposal: **Interval Tree Matching (ITM)**

- Based on the simple **Interval Tree** data structure
- It can be implemented using priority search trees or augmented AVL trees. For simplicity we've used the last one



Our proposal: **Interval Tree Matching (ITM)**

- Based on the simple **Interval Tree** data structure
- It can be implemented using priority search trees or augmented AVL trees. For simplicity we've used the last one

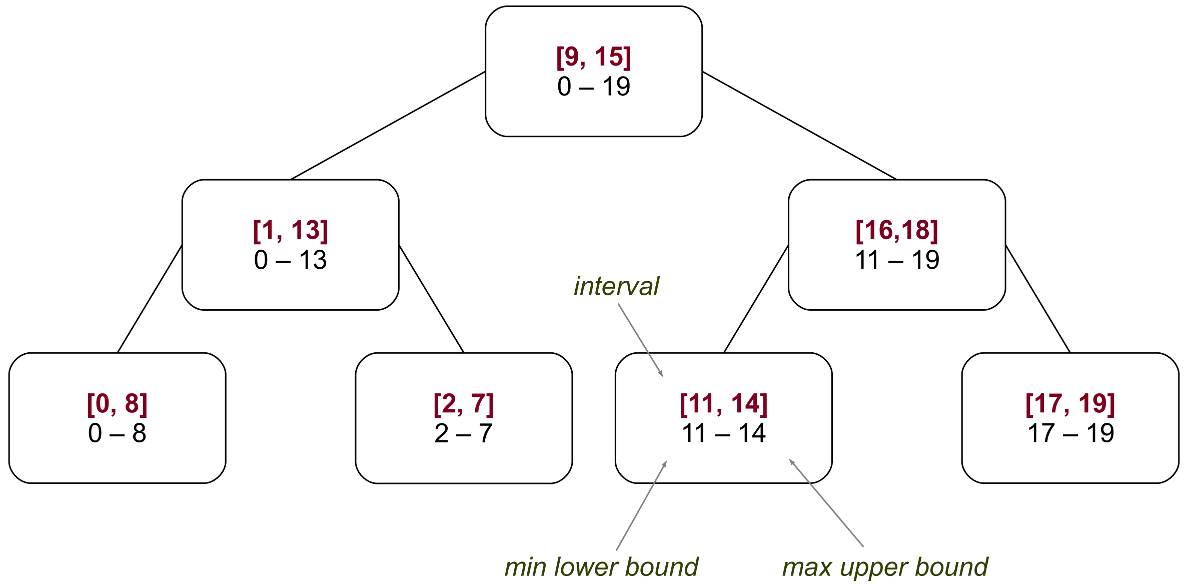


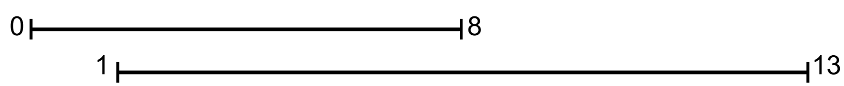
Our proposal: **Interval Tree Matching (ITM)**

- Based on the simple **Interval Tree** data structure
- It can be implemented using priority search trees or augmented AVL trees. For simplicity we've used the last one

balanced tree

insertion and delete are efficient





it contains all the subscription (or the update) regions

Interval Tree Matching (ITM): algorithm description

- Support function **INTERVAL-QUERY**:

  returns the list of intervals intersecting a given extent

- **Main function**:

  1) ***create*** *the Interval Tree with all the subscription extents*

  2) ***for every*** *update extent perform an* ***INTERVAL-QUERY***

Interval Tree Matching (ITM): comparison

- Support function **INTERVAL-QUERY**:
  returns the list of intervals intersecting a given extent

- **Main function**:
  1) ***create*** *the Interval Tree with all the subscription extents*
  2) ***for every*** *update extent perform an* ***INTERVAL-QUERY***

Algorithm	Computational cost	Additional space
Brute force	O(n·m) with **n** subscription extents, **m** update extents	none
Sort-based	O((n+m)log(n+m)+n·m)	O(n+m)
Interval tree	O(min{n·m, (K+1)log n}) with **K ≤ n·m** that is the total number of intersections	O(n)

Interval Tree Matching (ITM): comparison

- Support function **INTERVAL-QUERY**:

  returns the list of intervals intersecting a given extent

- **Main function**:

  1) ***create*** *the Interval Tree with all the subscription extents*

  2) ***for every*** *update extent perform an* ***INTERVAL-QUERY***

Algorithm	Computational cost	Additional space
Brute force	O(n·m) with **n** subscription extents, **m** update extents	none
Sort-based	O((n+m)log(n+m)+n·m)	O(n+m)
Interval tree	O(min{n·m, (K+1)log n}) with **K ≤ n·m** that is the total number of intersections	O(n)

Parallelization: the queries at point 2) are independent → it is trivial to parallelize

Interval Tree Matching (ITM): comparison

- Support function **INTERVAL-QUERY**:
  returns the list of intervals intersecting a given extent

- **Main function**:

  1) *create the Interval Tree with all the subscription extents*

  2) *for every update extent perform an* ***INTERVAL-QUERY***

Dynamic interval management: updates are efficient

Algorithm	Computational cost	Additional space
Brute force	O(n·m) with **n** subscription extents, **m** update extents	none
Sort-based	O((n+m)log(n+m)+n·m)	O(n+m)
Interval tree	O(min{n·m, (K+1)log n}) with **K ≤ n·m** that is the total number of intersections	O(n)

Parallelization: the queries at point 2) are independent → it is trivial to parallelize

Experimental evaluation

- The very same methodology used in [Raczy et al. 2005]

- Instances with a single dimension are considered

- **Main parameters**:

  - N = *number of extents* = $[50 \times 10^3, 500 \times 10^3]$ with 50% of update extents

  - α = *overlapping degree* = $\dfrac{\sum \text{area of extents}}{\text{area of the routing space}} = \{0.01, 1, 100\}$

- **Execution platform**: Intel® Core© i7-2600 3.40 GHz CPU with **4** physical cores and Hyper-Threading (HT), 16 GB RAM, Ubuntu 11.04

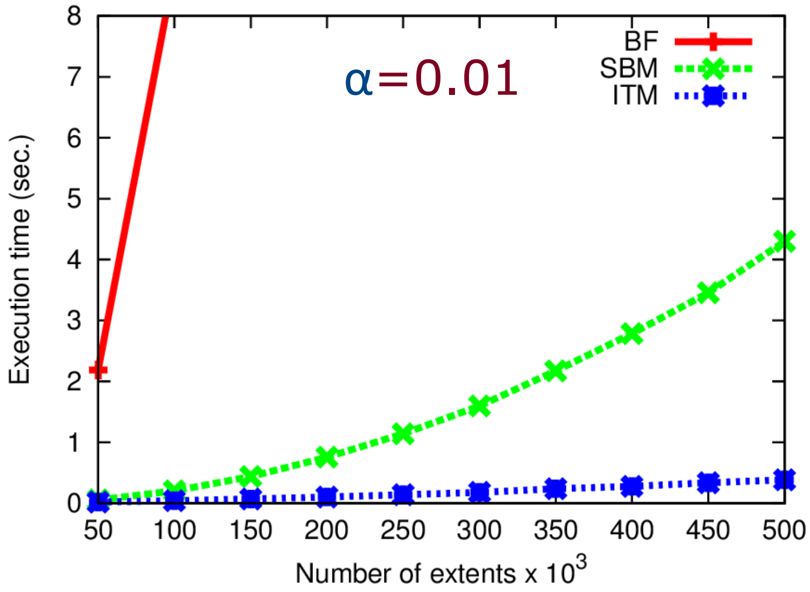
Experimental evaluation

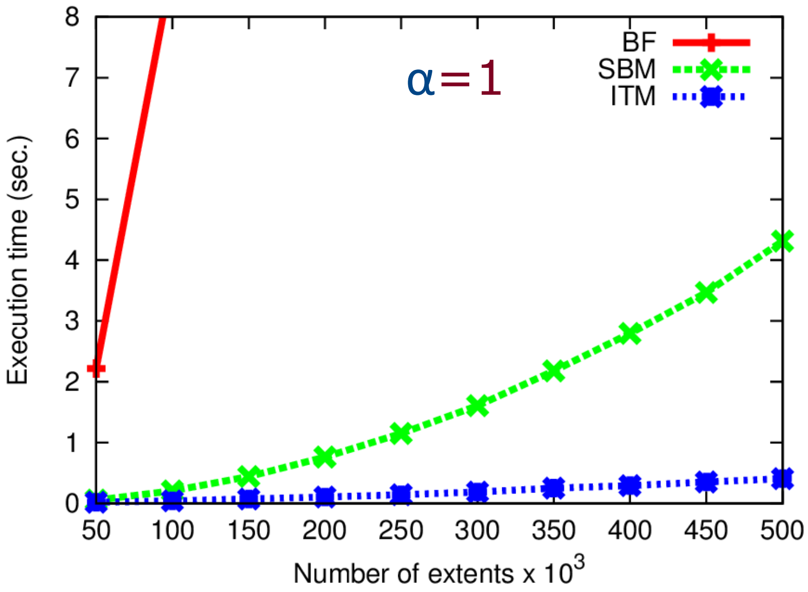
- The very same methodology used in [Raczy et al. 2005]
- Instances with a single dimension are considered
- **Main parameters**:
  - N = *number of extents* = $[50\text{x}10^3, 500\text{x}10^3]$ with 50% of update extents
  - α = *overlapping degree* = $\dfrac{\sum \text{area of extents}}{\text{area of the routing space}}$ = {0.01, 1, 100}
- **Execution platform**: Intel® Core© i7-2600 3.40 GHz CPU with **4** physical cores and Hyper-Threading (HT), 16 GB RAM, Ubuntu 11.04

All the source code and scripts used for this performance evaluation are available with a Free Software license from:

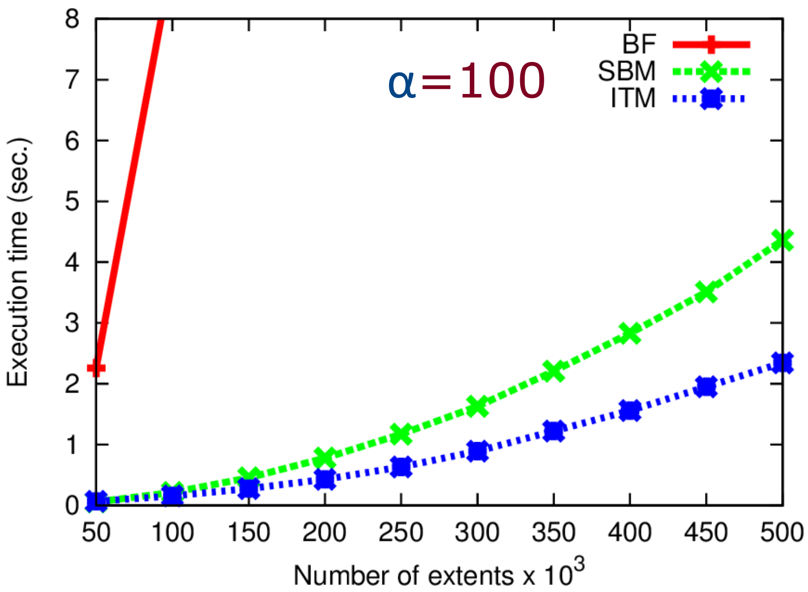
http://pads.cs.unibo.it

Experimental evaluation: sequential execution

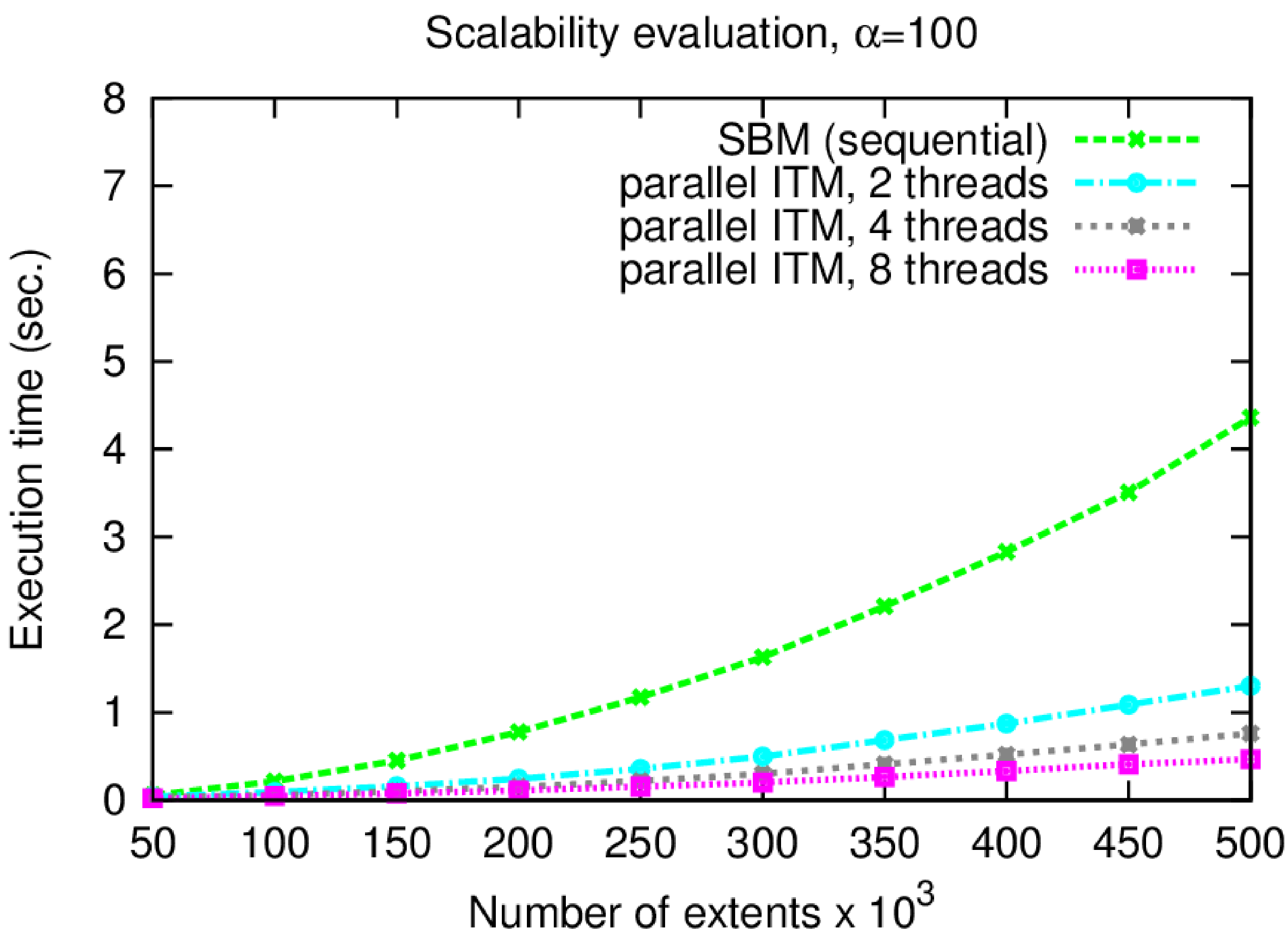




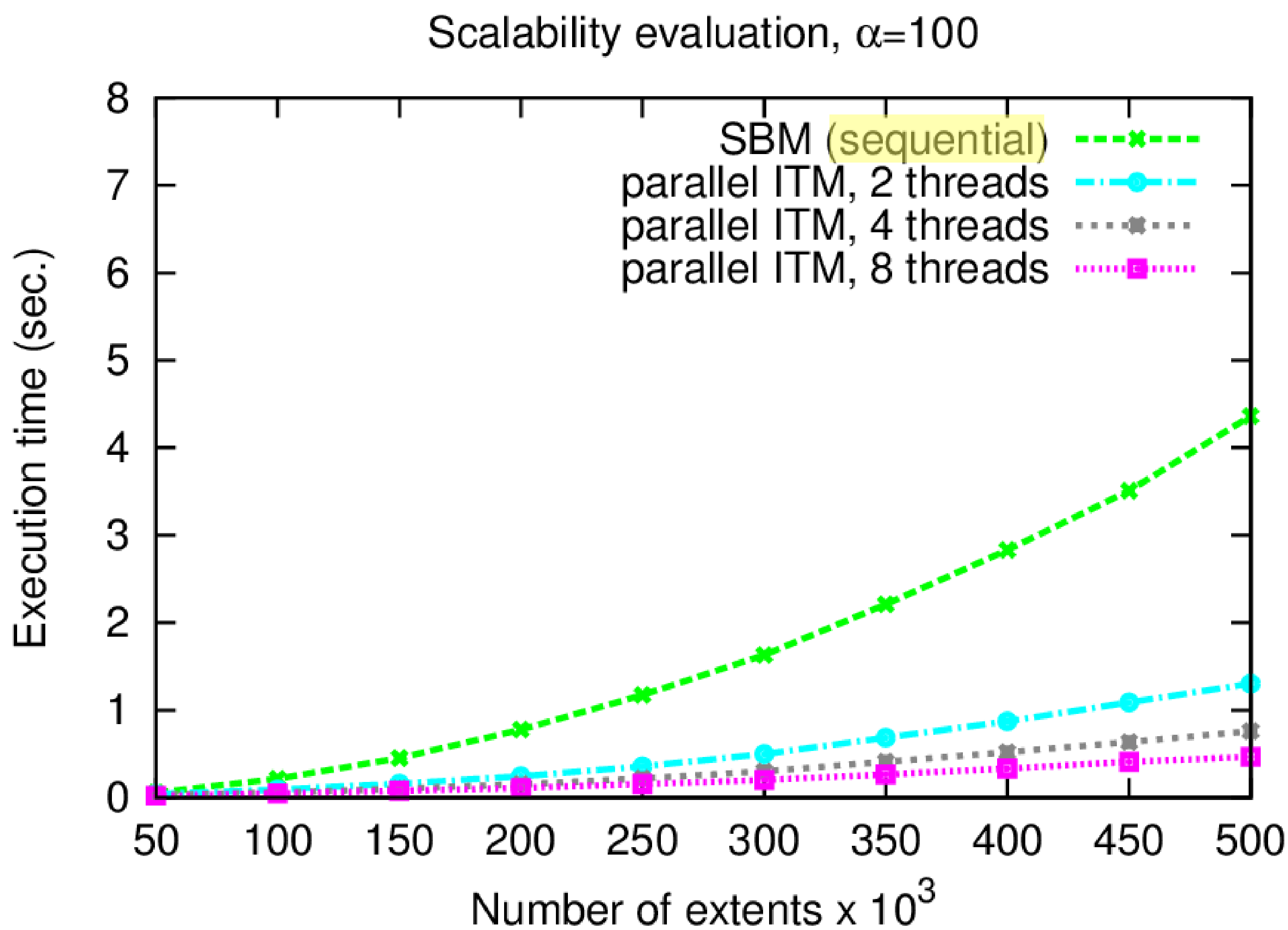
- **BF** is very slow
- **SBM** is unaffected by α
- **ITM** has the best performance
- Increasing α, the gap between **SBM** and **ITM** decreases
- Note that **α = 100** is a very high overlay degree



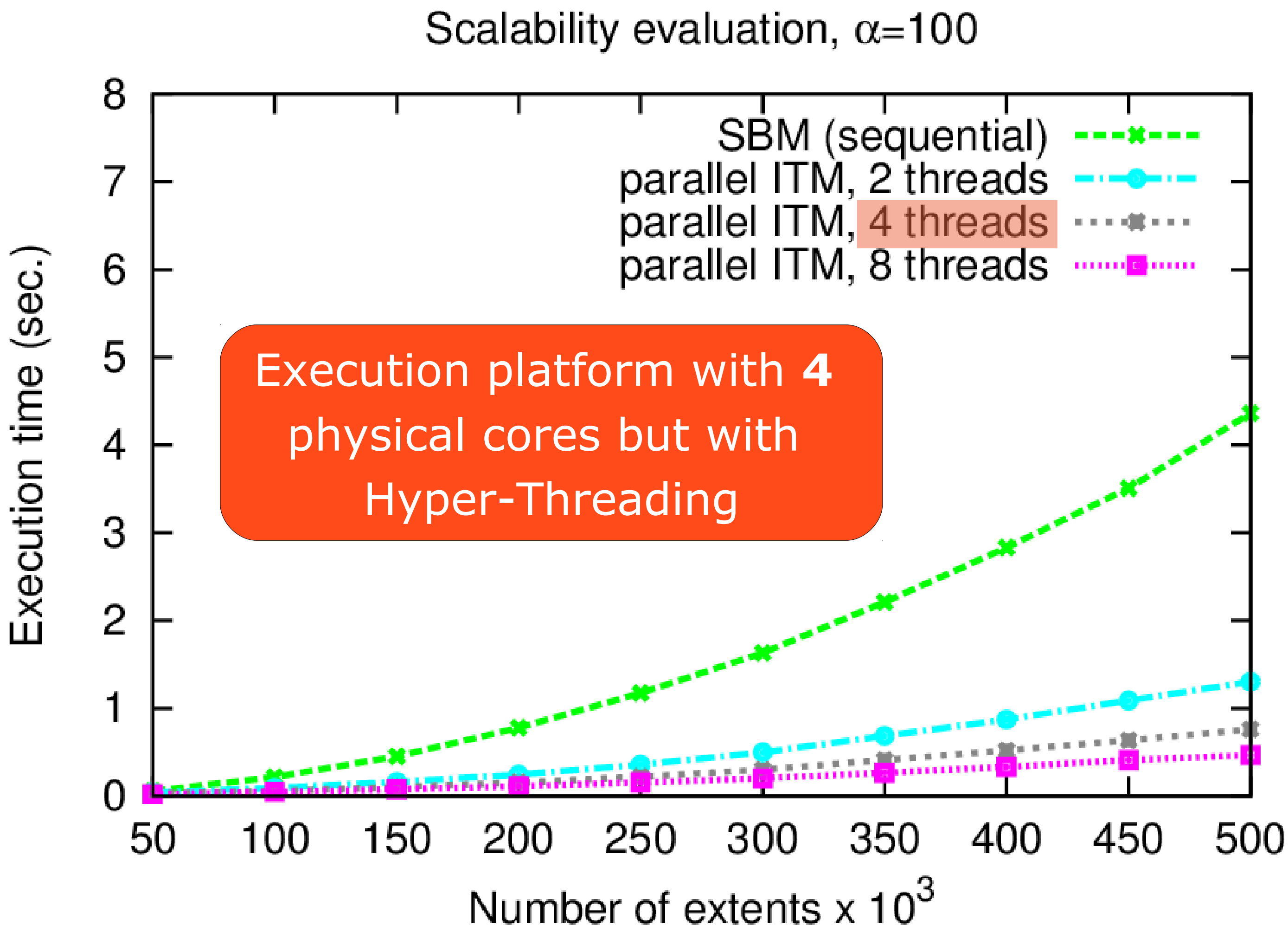
Experimental evaluation: parallel execution



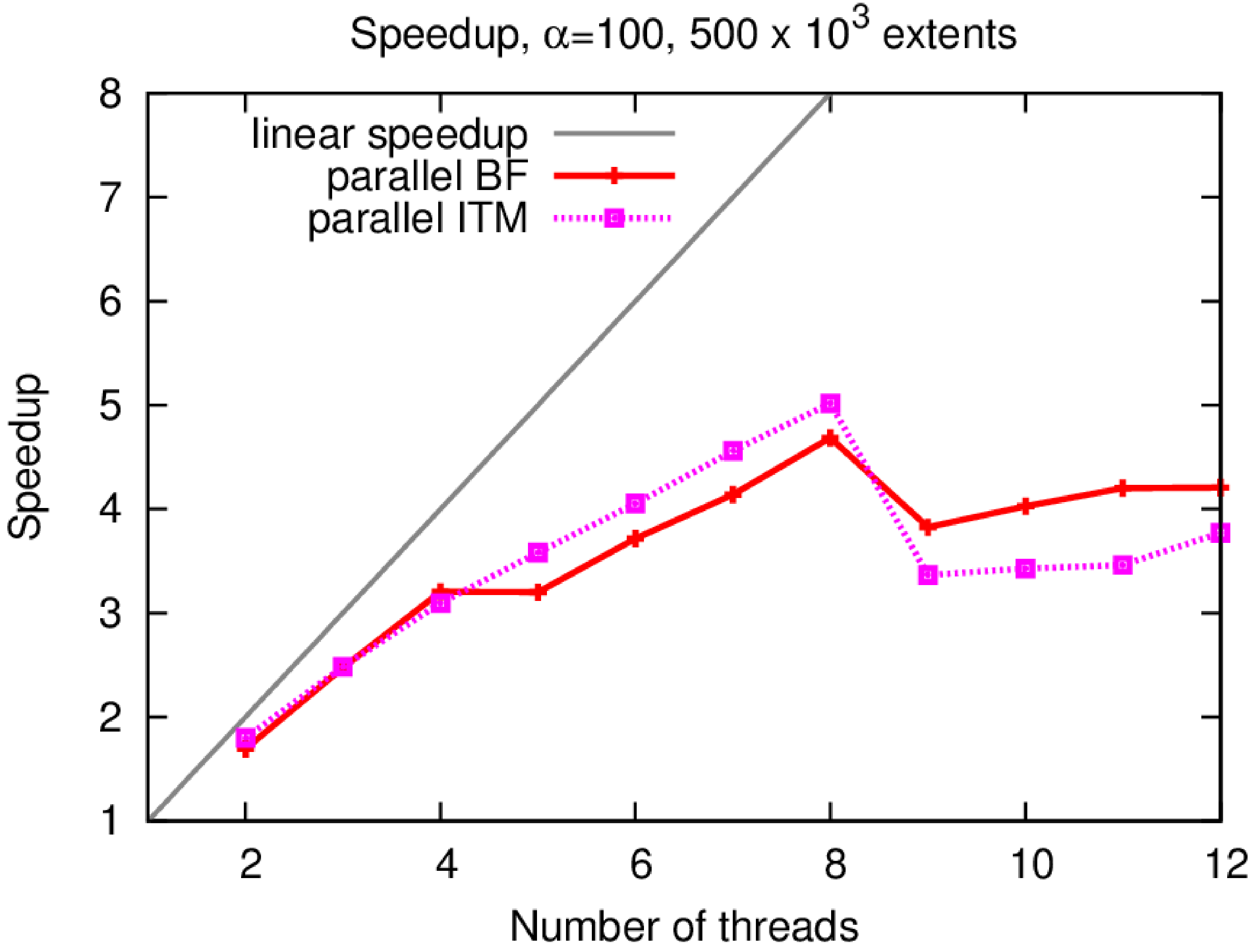
Experimental evaluation: parallel execution



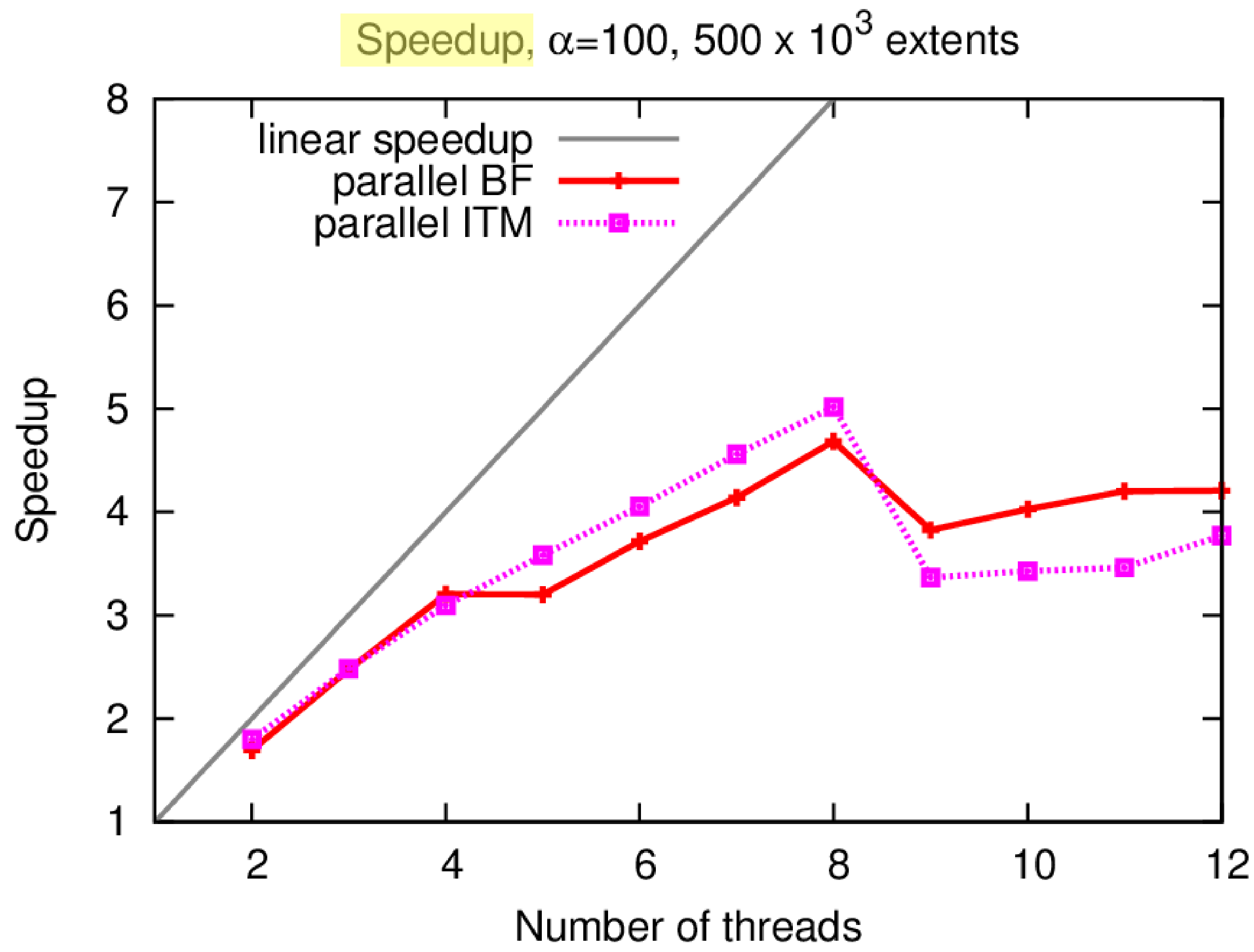
Experimental evaluation: parallel execution



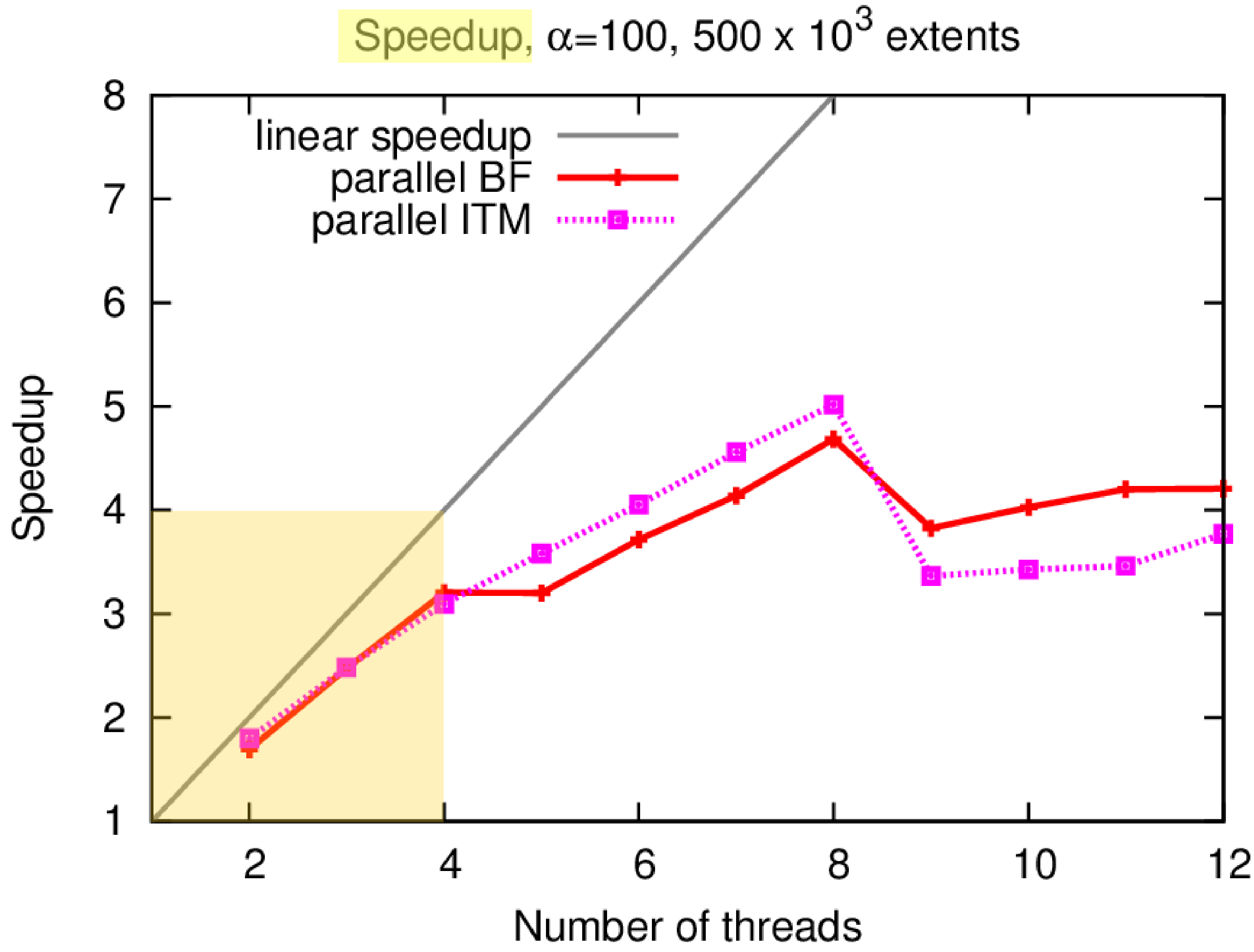
Experimental evaluation: parallel execution



Experimental evaluation: parallel execution

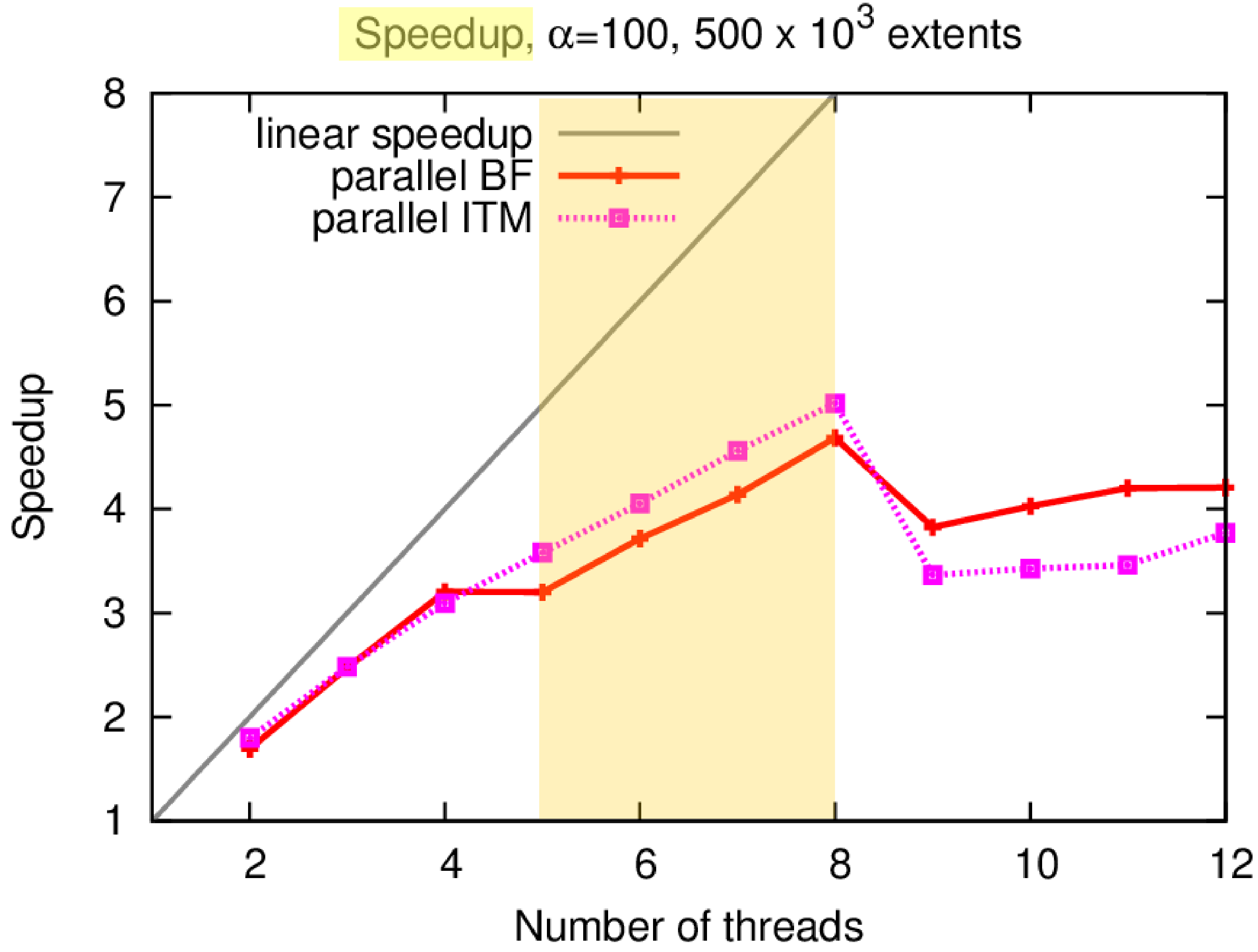


Experimental evaluation: parallel execution



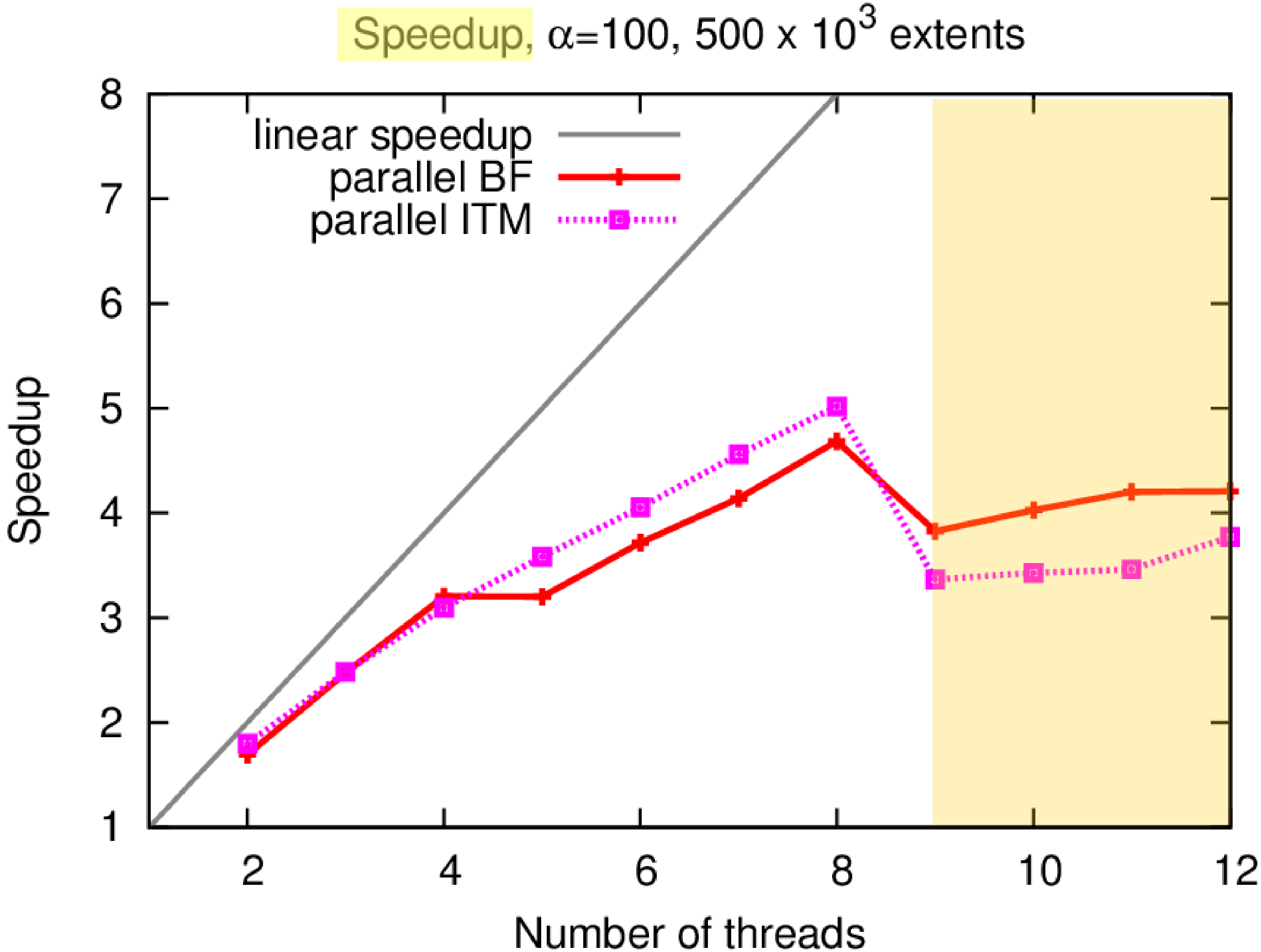
BF is embarrassingly parallel, in the range [1, 4] (physical cores) both algorithms obtain almost the same speedup

Experimental evaluation: parallel execution



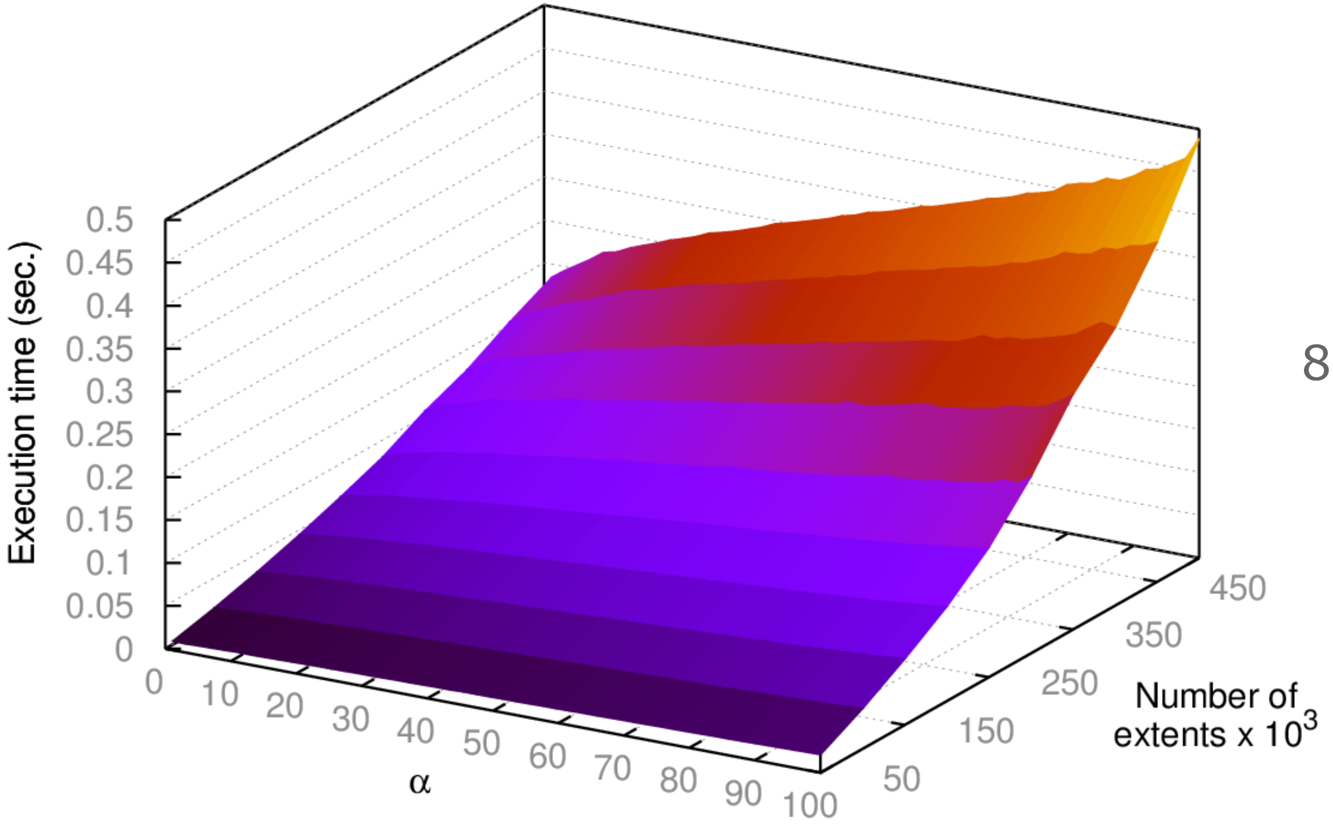
In the range [5, 8] (**H**yper-**T**hreading logical cores) **ITM** is better than **BF**. **HT** provides a performance boost

Experimental evaluation: parallel execution



With more threads than logical cores there is a significant performance drop

Experimental evaluation: parallel execution



8 threads

As expected, the **ITM** depends on both the **number of extents** and the **overlap degree** α

Conclusions

- We've proposed a new parallel **D**ata **D**istribution **M**anagement (**DDM**) mechanism called **I**nterval **T**rees **M**atching (**ITM**)

- Both sequential and parallel implementations of **ITM** show good results when compared to the state of the art

- Since updates on the interval tree are efficient, the **ITM** can be easily extended to deal with the **dynamic matching problem**

- All our source code is available with a Free Software license: we believe that all the published results **have to be reproducible**

Further information

Moreno Marzolla, Gabriele D'Angelo, Marco Mandrioli

A Parallel Data Distribution Management Algorithm

Proceedings of the 16-th ACM/IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2013). Delft, Netherlands, 2013

An **draft version** of this paper is available on the

open e-print archive

All the **source code** used in this paper is available at

http://pads.cs.unibo.it

Gabriele D'Angelo

- E-mail: <g.dangelo@unibo.it>
- http://www.cs.unibo.it/gdangelo/

A Parallel Data Distribution Management Algorithm

Gabriele D’Angelo

<g.dangelo@unibo.it>

http://www.cs.unibo.it/gdangelo/

joint work with:

Moreno Marzolla *and* ***Marco Mandrioli***

Delft, Netherlands

Distributed **S**imulation and **R**eal **T**ime Applications (**DS-RT**), 2013