www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Evaluating the Impact of OpenMP 4.0 Extensions on Relevant Parallel Workloads

Raul Vidal, **Marc Casas**, Miquel Moreto, Dimitrios Chasapis, Roger Ferrer, Xavier Martorell, Eduard Ayguade, Jesus Labarta, and Mateo Valero

Scope

- OpenMP is the most widely used programming model for shared memory environments
- OmpSs is a forerunner of OpenMP
  - Handles the main OpenMP constructs
  - Includes some extra features
- Several OmpSs features not included in the OpenMP standard are discussed and evaluated in this paper:
  - **the multi-dependences feature**, which allows to specify different data-dependence scenarios in a single #pragma annotation
  - **the concurrent clause**, which relaxes task synchronization and allows increased overlap of task creation with remaining computations
  - **runtime support for NUMA-aware scheduling** of tasks, which schedules them on the cores closest to the data the task accesses

Targeted Applications

Fluidanimate

- Incompressible fluid simulation on a 2D grid
- Each iteration is composed of 8 different kernels
  - computing fluid densities and forces at given points
  - handling fluid collisions or updating particle locations
- The concurrent and the multi-dependence features are applied

Streamcluster

- Online clustering problem
- Points are assigned to centers
  - Each point is assigned to the closest center
  - New centers are opened if the benefits of opening them are significant
- The NUMA-aware scheduling feature is applied

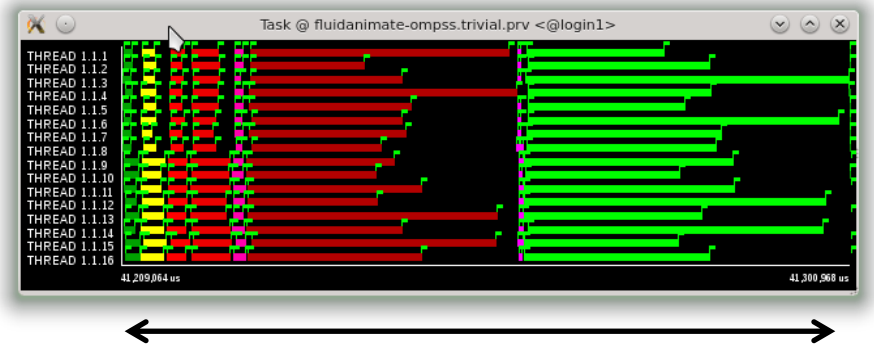
PARSEC 2.1 release (2010)

- Release 2.1 includes codes in
  - Pthreads
  - OpenMP
  - Intel TBB
- These benchmarks are widely used to evaluate many-core architectures

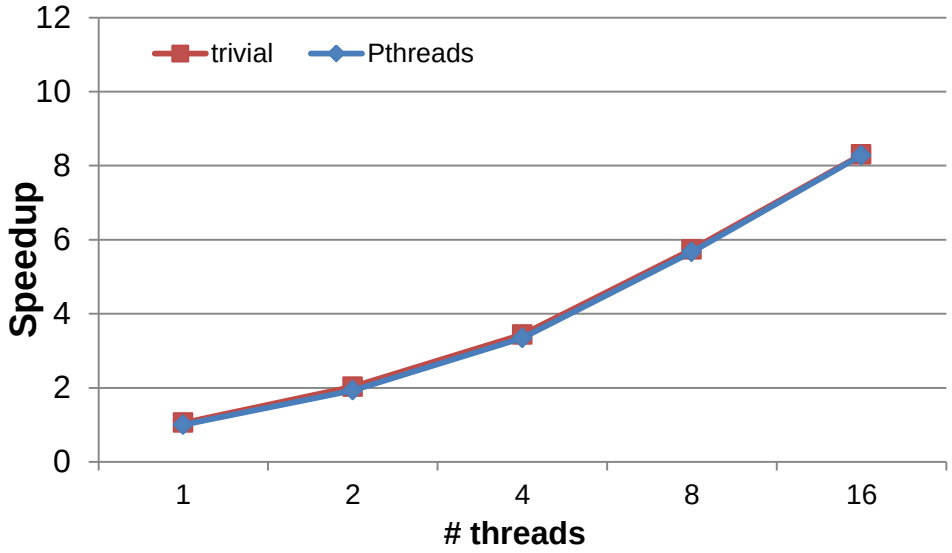
PARSEC 2.1	Parallelization Model		
	Pthreads	OpenMP	Intel TBB
blackscholes	Yes	Yes	Yes
bodytrack	Yes	Yes	Yes
canneal	Yes	No	No
dedup	Yes	No	No
facesim	Yes	No	No
ferret	Yes	No	No
fluidanimate	Yes	No	Yes
freqmine	No	Yes	No
raytrace	Yes	No	No
streamcluster	Yes	No	Yes
swaptions	Yes	No	Yes
vips	Yes	No	No
x264	Yes	No	No

Fluidanimate

- Each iteration of Fluidanimate consists in 8 different kernels that operate over the elements of a 2D grid.
  - **Pthreads Parallelization:**
    - 8 barriers per iteration between the 8 kernels
    - Threads need the previous kernels' computations on its grid segment and its neighbors to be finished once the execution of the new kernel start.
  - **Trivial OmpSs:** Same strategy



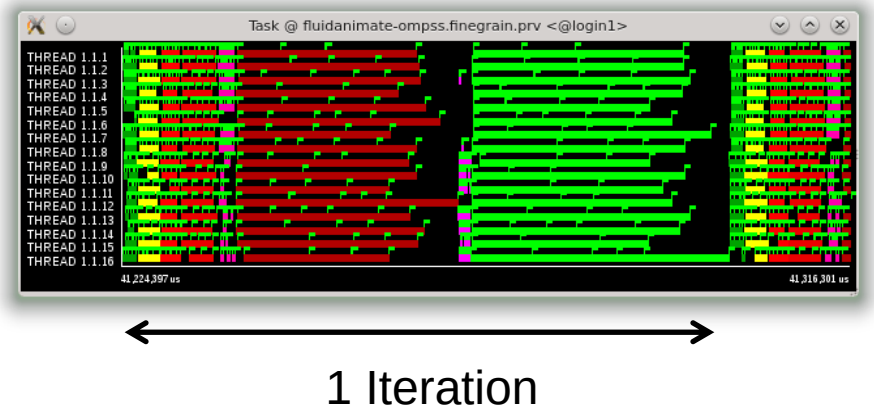
1 Iteration

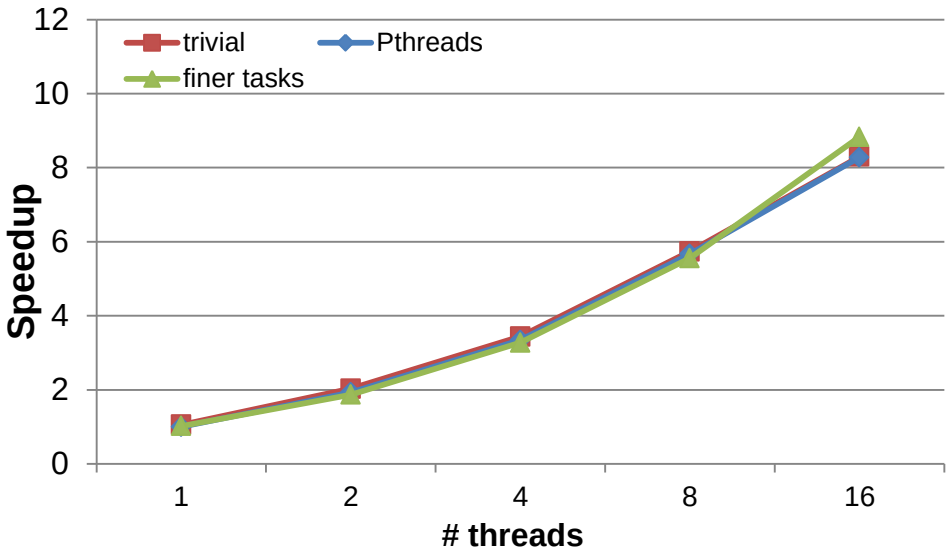


Experiments run on a NUMA node composed of two 8-core Intel Xeon E5-2670

Fluidanimate

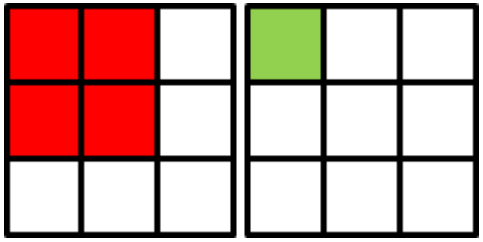
- Each iteration of Fluidanimate consists in 8 different kernels
  - **Pthreads Parallelization:** 8 barriers per iteration
  - **Trivial OmpSs:** Same strategy
  - **Finer Tasks:** Tasks are 4 times finer



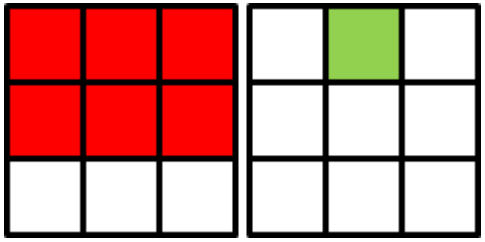


Fluidanimate

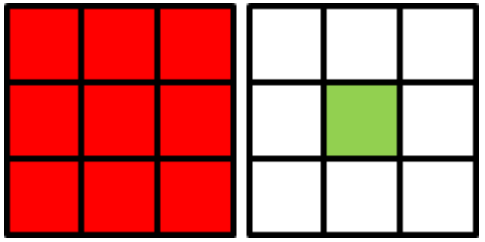
- Each iteration of Fluidanimate consists in 8 different kernels
  - **Pthreads Parallelization:** 8 barriers per iteration
  - **Trivial OmpSs:** Same strategy
  - **Finer Tasks:** Tasks are 4 times finer
  - **OmpSs Multidependences:** Intra-iteration barriers replaced by dependences
    - There are three dependence scenarios, depending on the grid element the task operates at:



Element in the corner:
4 dependences



Element in the halo:
6 dependences



Element in the center:
9 dependences

Fluidanimate

- Each iteration of Fluidanimate consists in 8 different kernels
  - **Pthreads Parallelization:** 8 barriers per iteration
  - **Trivial OmpSs:** Same strategy
  - **Finer Tasks:** Tasks are 4 times finer
  - **OmpSs Multidependences:** Intra-iteration barriers replaced by dependences
    - 8 kernels x 4 dependence scenarios → 32 pragma annotations!

```
if (segment in corner)
#pragma omp task in( neighborhood[0], ..., neighborhood[3] )
//Task Code
else if (segment in boundary)
#pragma omp task in( neighborhood[0], ..., neighborhood[5] )
//Task Code
else if (internal segment)
#pragma omp task in( neighborhood[0], ..., neighborhood[8] )
//Task Code
```

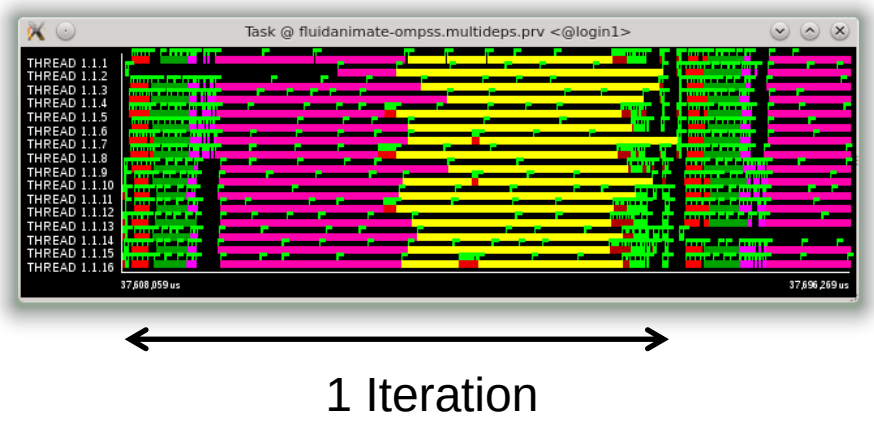
Fluidanimate

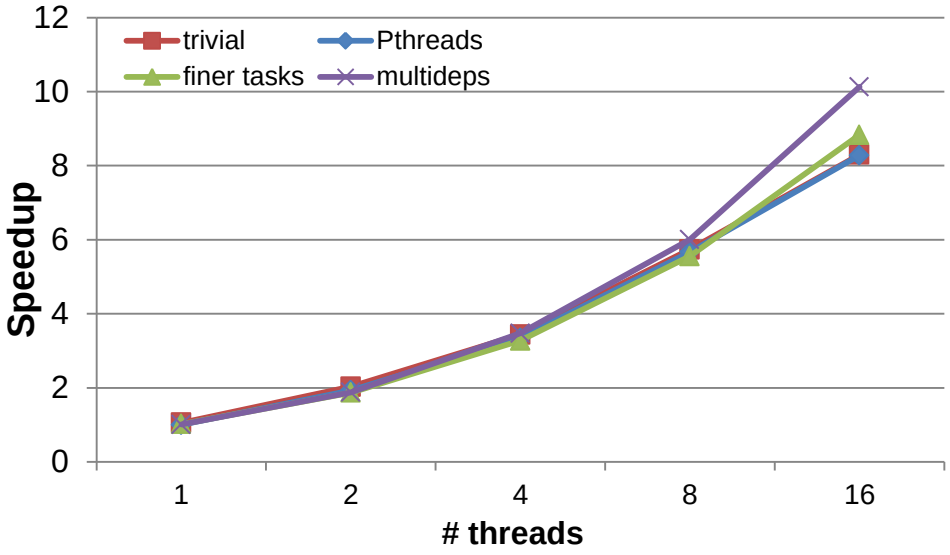
- Each iteration of Fluidanimate consists in 8 different kernels
  - **Pthreads Parallelization:** 8 barriers per iteration
  - **Trivial OmpSs:** Same strategy
  - **Finer Tasks:** Tasks are 4 times finer
  - **OmpSs Multidependences:** Intra-iteration barriers replaced by dependences
    - Multidepence capability allows handling several dependence scenarios with a single pragma annotation

```
#pragma omp task in( { neighborhood[j] , j=0:neighborhood.size() } )
//Task Code
```

Fluidanimate

- Each iteration of Fluidanimate consists in 8 different kernels
  - **Pthreads Parallelization:** 8 barriers per iteration
  - **Trivial OmpSs:** Same strategy
  - **Finer Tasks:** Tasks are 4 times finer
  - **OmpSs Multidependences:** Intra-iteration barriers replaced by dependences





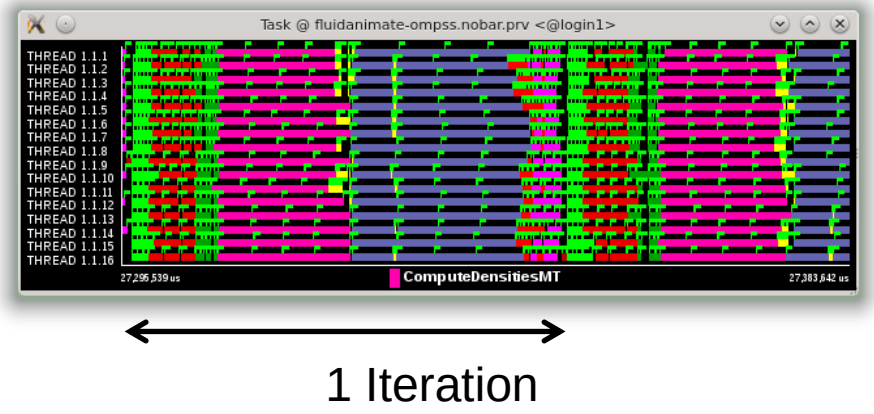
Fluidanimate

- Each iteration of Fluidanimate consists in 8 different kernels
  - **Pthreads Parallelization:** 8 barriers per iteration
  - **Trivial OmpSs:** Same strategy
  - **Finer Tasks:** Tasks are 4 times finer
  - **OmpSs Multidependences:** Intra-iteration barriers replaced by dependences
  - **No Barriers:** Inter-iterations barriers replaced by concurrent clause
    - The concurrent clause is equivalent to an inout dependency on *variable*, but allows the tasks to operate concurrently on it
    - It relaxes synchronization points and allow some degree of overlap between task creation and computation

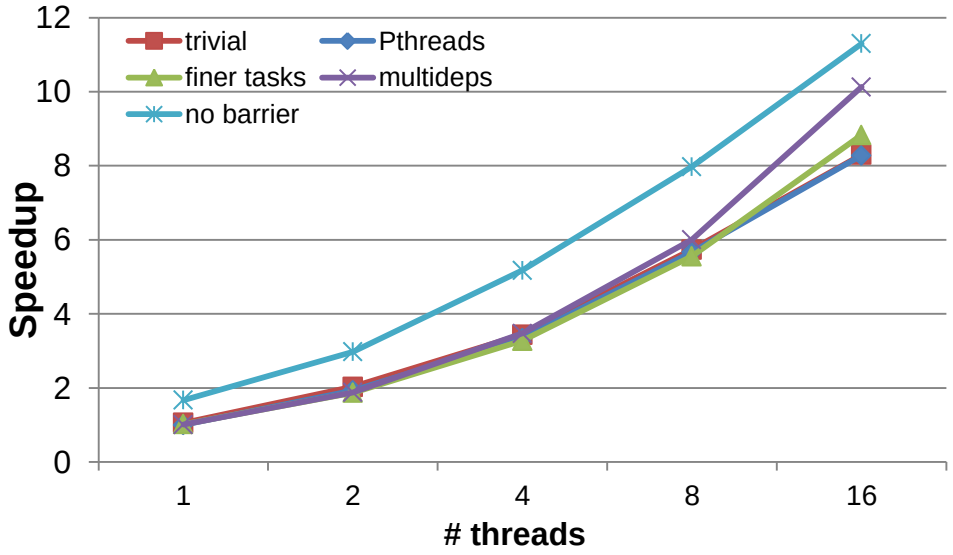
```
for each partition
#pragma omp task concurrent(variable)
    taskfunction1();

for each partition
#pragma omp task in(variable)
    taskfunction2();
```

Fluidanimate

- Each iteration of Fluidanimate consists in 8 different kernels
  - **Pthreads Parallelization:** 8 barriers per iteration
  - **Trivial OmpSs:** Same strategy
  - **Finer Tasks:** Tasks are 4 times finer
  - **OmpSs Multidependences:** Intra-iteration barriers replaced by dependences
  - **No Barriers:** Inter-iterations barriers replaced by concurrent clause



1 Iteration



Targeted Applications

Fluidanimate

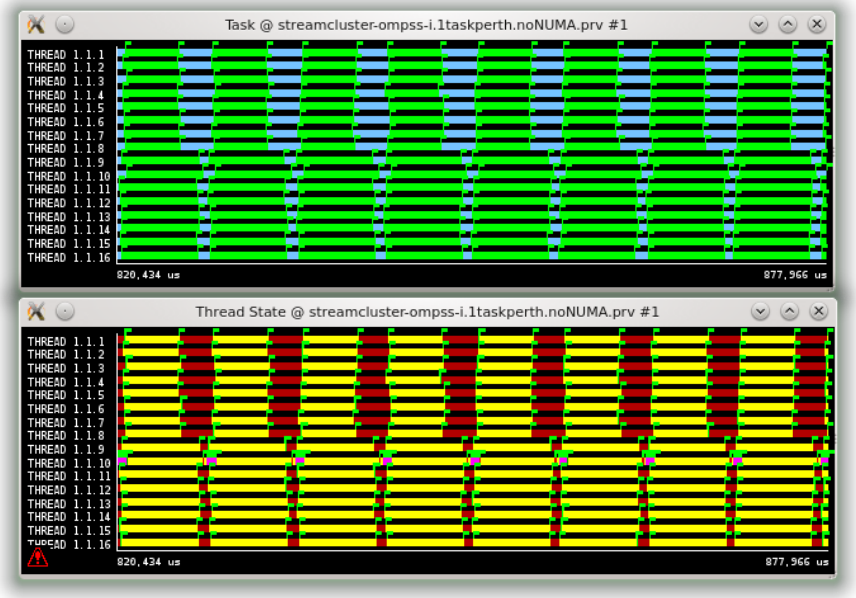
- Incompressible fluid simulation
- Each iteration is composed of 8 different kernels
  - computing fluid densities and forces at given points
  - handling fluid collisions or updating particle locations
- The concurrent and the multi-dependence features are applied to it

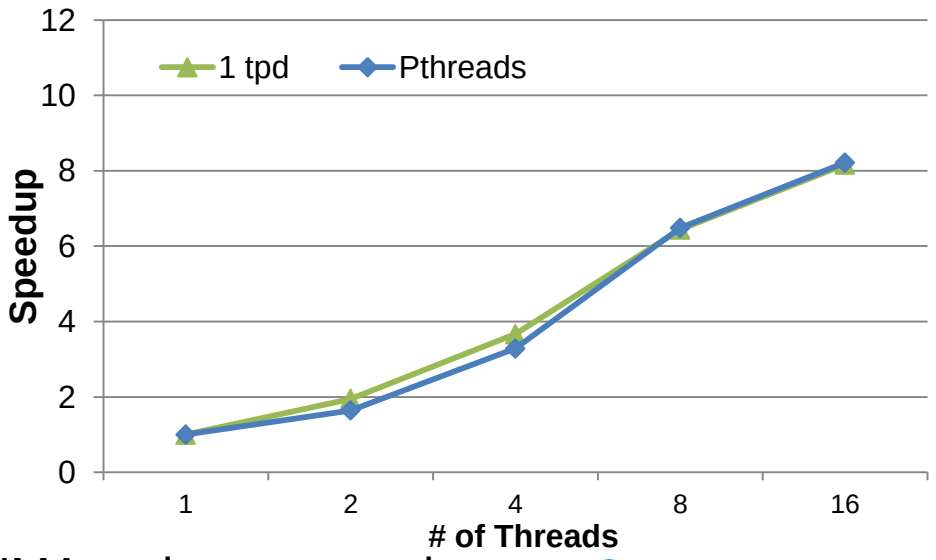
Streamcluster

- Online clustering problem
- Points are assigned to centers
  - Each point is assigned to the closest center
  - New centers are opened if the benefits of opening them are significant
- NUMA-aware scheduling feature applied to it

Streamcluster

- The array containing all points is split into chunks
  - **Pthread Parallelization**: Each chunk is then split into several domains, one per thread. Barrier synch different chunks
  - **Trivial OmpSs**: Same strategy, one task per domain (1tpd)
- NUMA effects, since data structures are allocated before creating the threads

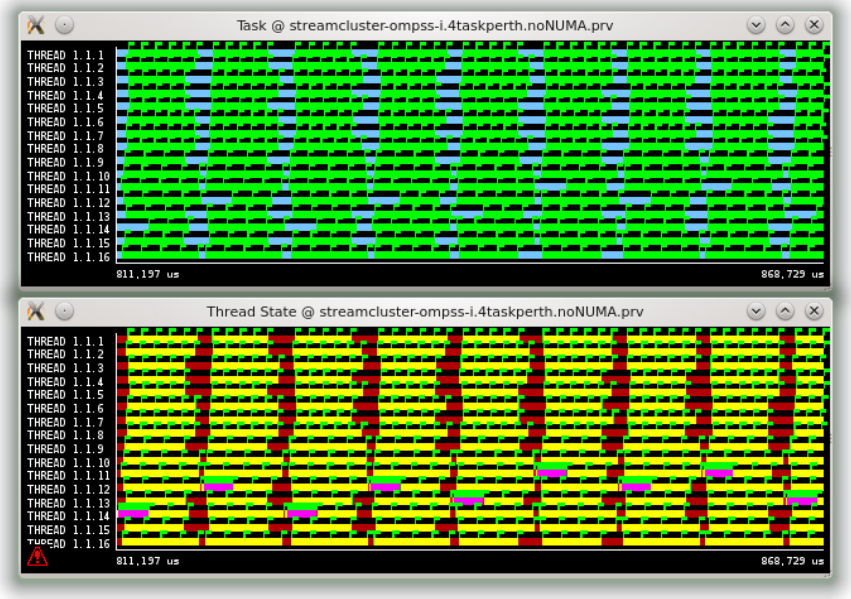


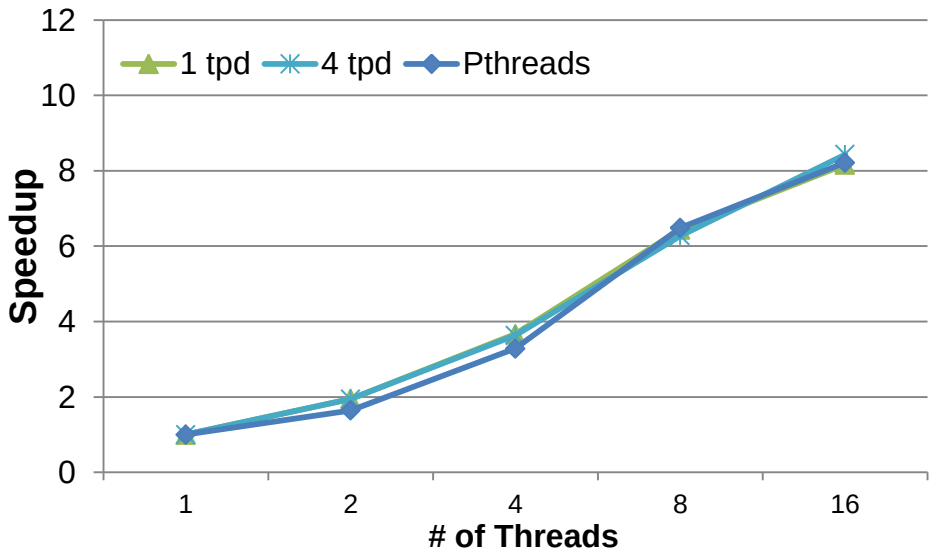


Experiments run on a NUMA node composed of two 8-core Intel Xeon E5-2670

Streamcluster

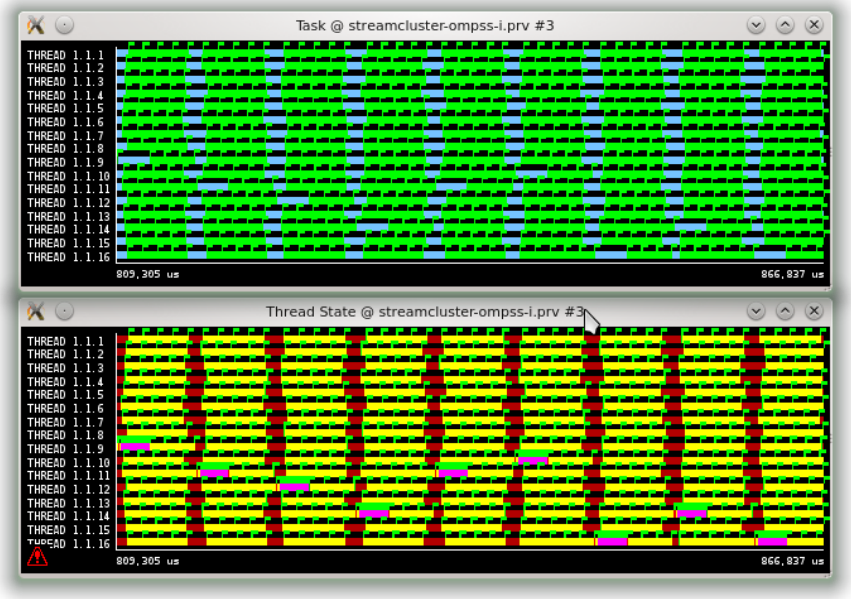
- The array containing all points is split into chunks
  - **Pthread Parallelization**: Each chunk is then split into several domains
  - **Trivial OmpSs**: Same strategy, 1 task per domain (1tpd)
  - **Fine Grain OmpSs:** Reduce task granularity, 4 task per domain (4tpd)
- Idle time is reduced, task creation overhead is increased

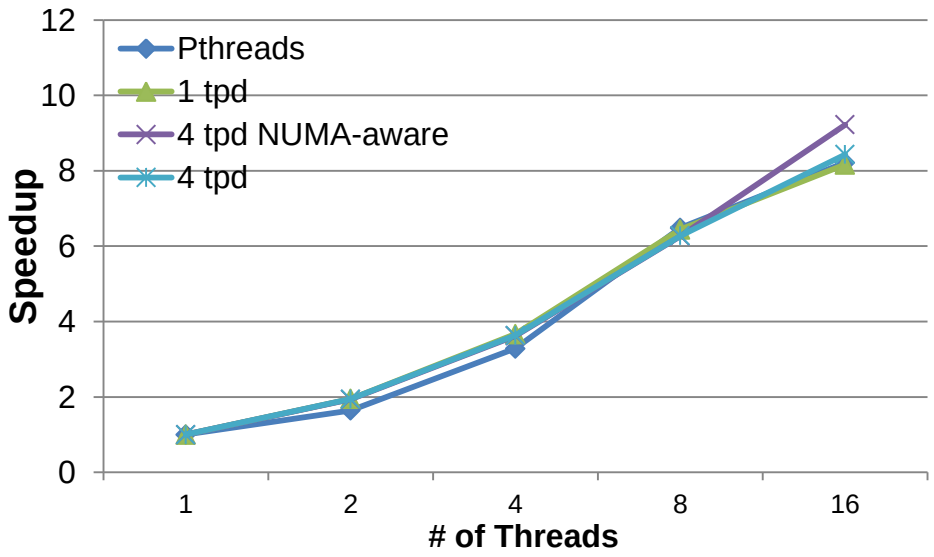




Streamcluster

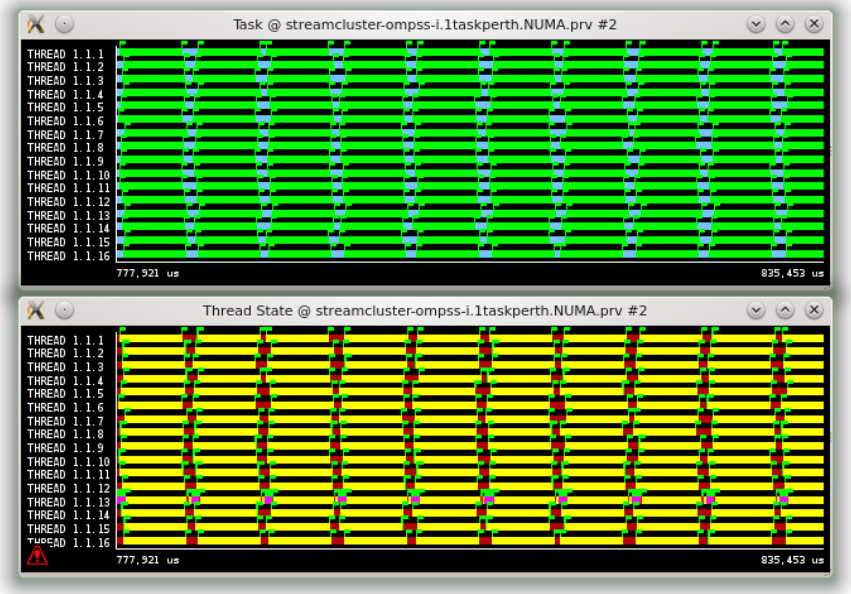
- The array containing all points is split into chunks
  - **Pthread Parallelization**: Each chunk is then split into several domains
  - **Trivial OmpSs**: Same strategy, 1 task per domain (1tpd)
  - **Fine Grain OmpSs:** Reduce task granularity, 4 task per domain (4tpd)
  - **NUMA-Aware OmpSs:** Distribute memory allocation evenly among the two sockets and run tasks on the sockets where data is allocated

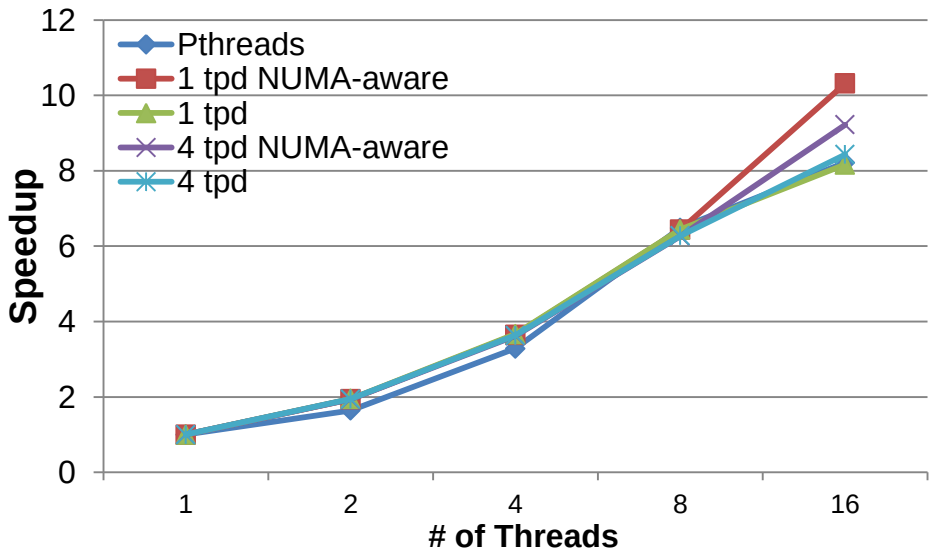




Streamcluster

- The array containing all points is split into chunks
  - **Pthread Parallelization**: Each chunk is then split into several domains
  - **Trivial OmpSs**: Same strategy, 1 task per domain (1tpd)
  - **Fine Grain OmpSs:** Reduce task granularity, 4 task per domain (4tpd)
  - **NUMA-Aware OmpSs:** 4 tasks per domain
  - **NUMA-Aware OmpSs:** 1 task per domain





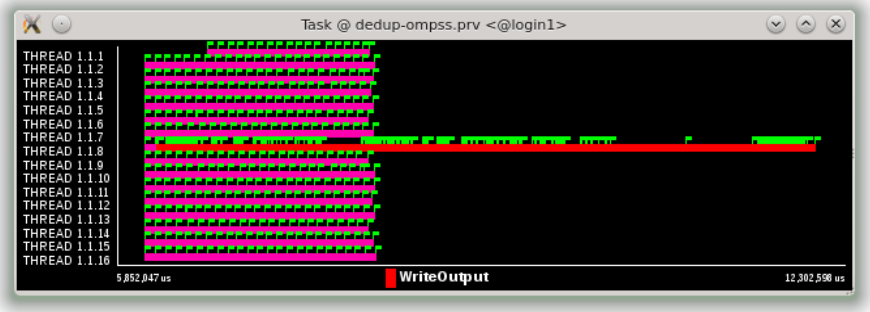
PARSECSs (2015)

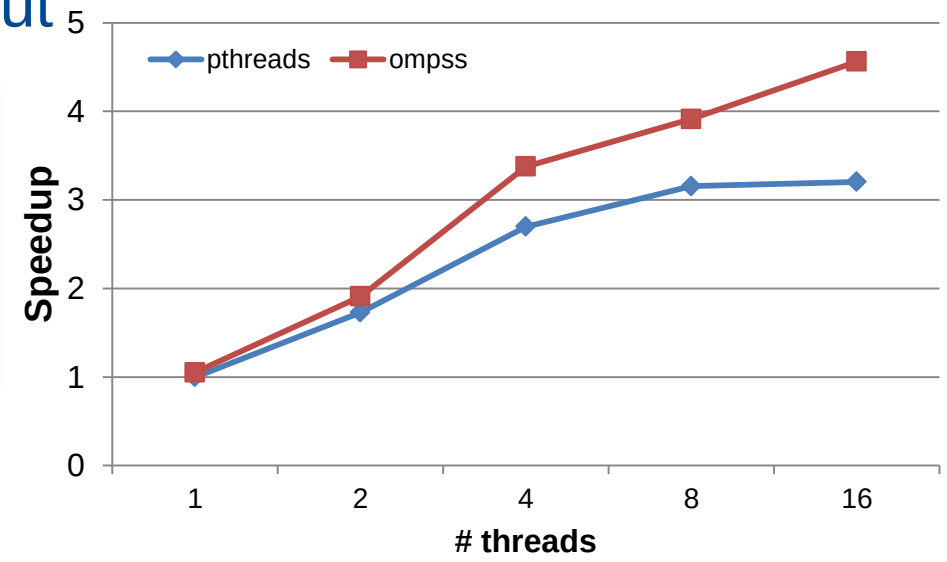
- PARSECSs includes
  - Pthreads
  - OpenMP 2.0
  - Intel TBB
  - OmpSs
- New versions aim to fully use multi-core architectures
- Scalability significantly improved

	Parallelization Model			
	Pthreads	**OpenMP**	**Intel TBB**	**OmpSs**
blackscholes	Yes	Yes	Yes	Yes
bodytrack	Yes	Yes	Yes	Yes
canneal	Yes	No	No	Yes
dedup	Yes	No	No	Yes
facesim	Yes	No	No	Yes
ferret	Yes	No	No	Yes
fluidanimate	Yes	No	Yes	Yes
freqmine	No	Yes	No	Yes
raytrace	Yes	No	No	No
streamcluster	Yes	No	Yes	Yes
swaptions	Yes	No	Yes	Yes
vips	Yes	No	No	No
x264	Yes	No	No	Yes

Dedup

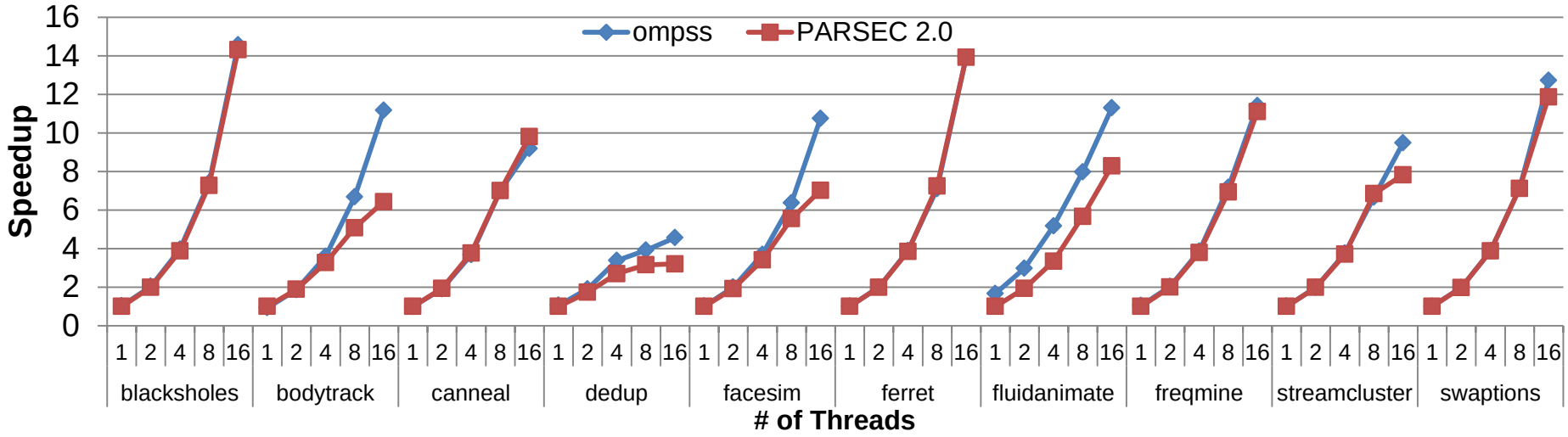
- Dedup compresses data streams and writes them down to a file. As such, there are two main routines:
  - **FragmentRefine:** Compression stages, one per each data stream
  - **WriteOutput:** Compressed data is written into a file. Streams must appear in a certain order, so the tasks muse run in that same order
- OmpSs allows to run FragmentRefine tasks in parallel and overlap them with WriteOutput



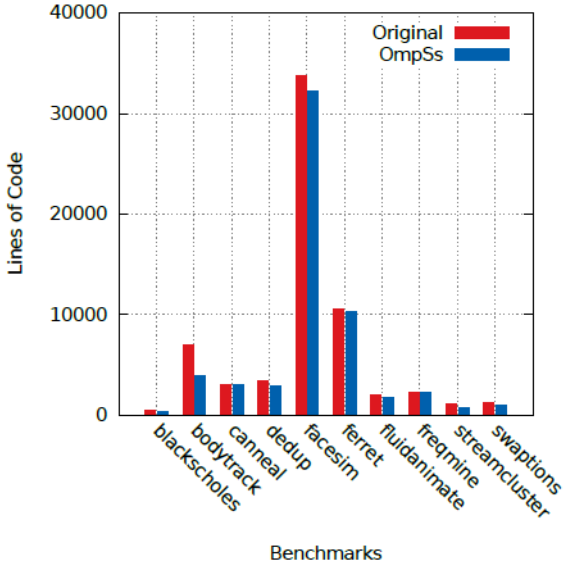


PARSECSs Evaluation

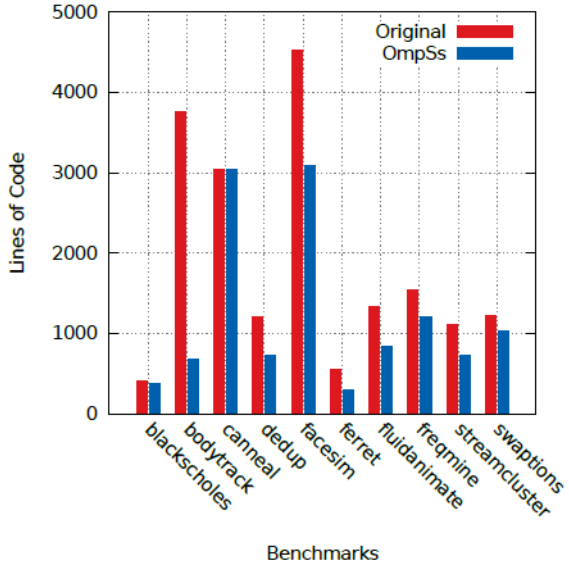
Performance



Lines of code



(a) Comparison between all source files.



(b) Comparison between only source files containing parallel code.