www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

MUSA Tutorial Session 2: Completing a real simulation

Francesc Martínez
RoMoL at BSC-CNS

Barcelona, 11th May 2018

Outline

- Generating the memory mode trace
- Executing the memory mode simulation
- Integrating the memory mode simulation
- Result analysis

Step 1: Generating the memory mode trace

- To modify the configuration:

```
$> vim job_tracer_memory.bash
```

- We can modify:

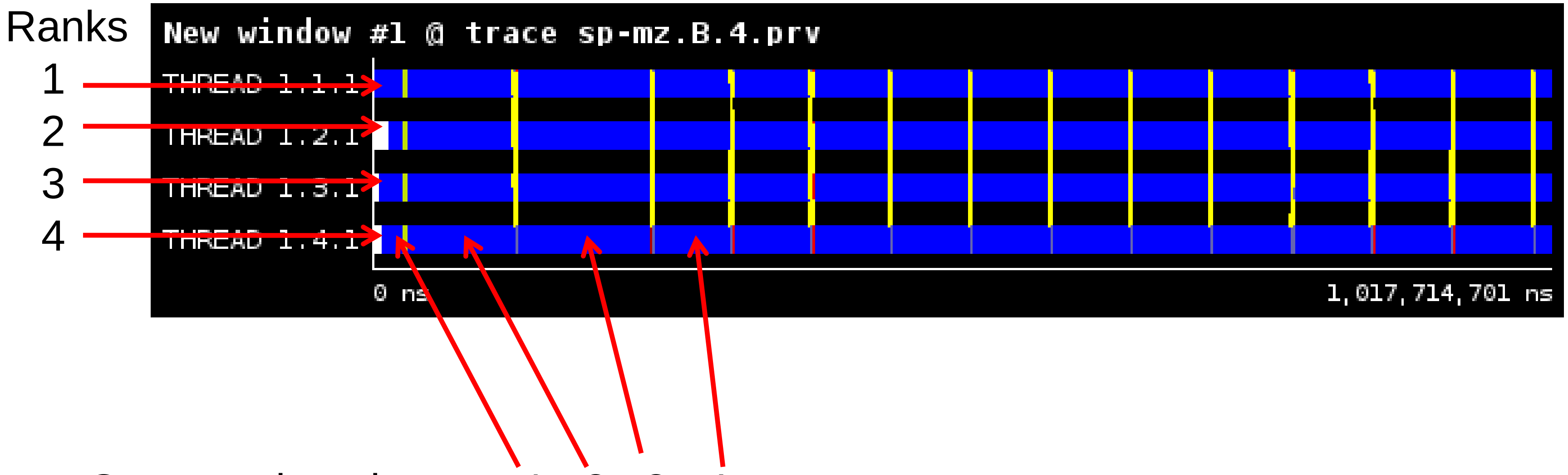
– TSMPI_RANK_INIT	First MPI Rank to be traced
– TSMPI_RANK_NUM	Number of MPI Ranks tot be traced
– TSMPI_MEM_PHASES_INIT	First computation phase to be traced
– TSMPI_MEM_PHASES_NUM	Number of computation phases to be traced

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Step 1: Generating the Memory mode trace

```
$> cd TRACE_sp-mz.B.4_000004_BRST/ TRACE/trace_prv
$> module load paraver
$> wxparaver trace_sp-mz.B.4.prv
```

Ranks

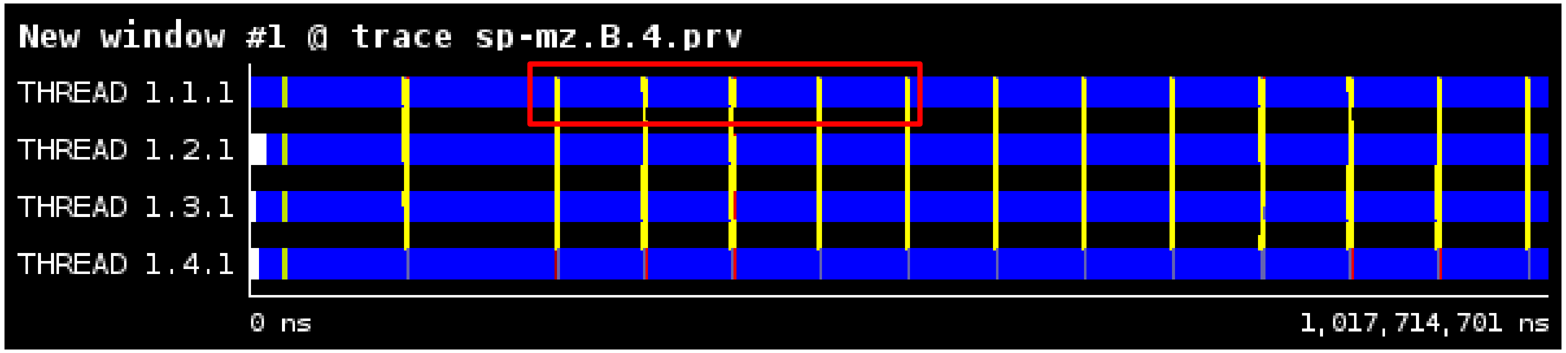


Computation phases: 1 2 3 4

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Step 1: Generating the Memory mode trace

- TSMPI_RANK_INIT = 1
- TSMPI_RANK_NUM = 1
- TSMPI_MEM_PHASES_INIT = 4
- TSMPI_MEM_PHASES_NUM = 4



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

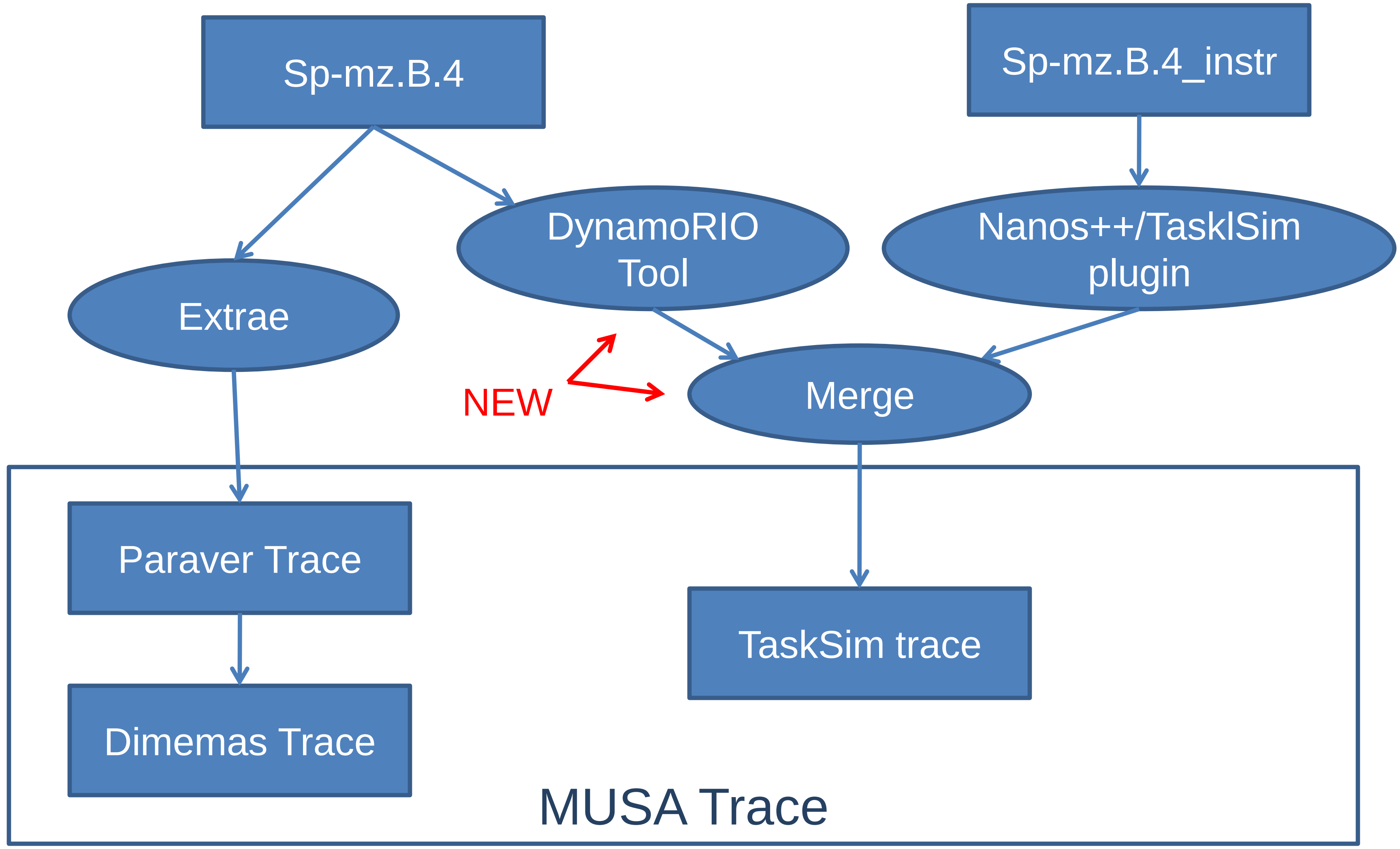
Step 1: Generating the Memory mode trace

- To generate the Memory mode trace:

```
$> sbatch job_tracer_memory.bash
```

Step 1: Generating the Memory mode trace

- Both binaries are used!



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Step 1: Generating the Memory mode trace

- A Memory mode trace contains:
  - MPI Events and durations
    - MPI_WaitAll, MPI_Send/Recv, MPI_Barrier, …
  - OpenMP/OmpSs Events and durations
    - Task Creation, Dependencies, TaskWaits, …
  - For every OpenMP/OmpSs task (NEW):
    - List of executed instructions
    - List of accessed memory addresses

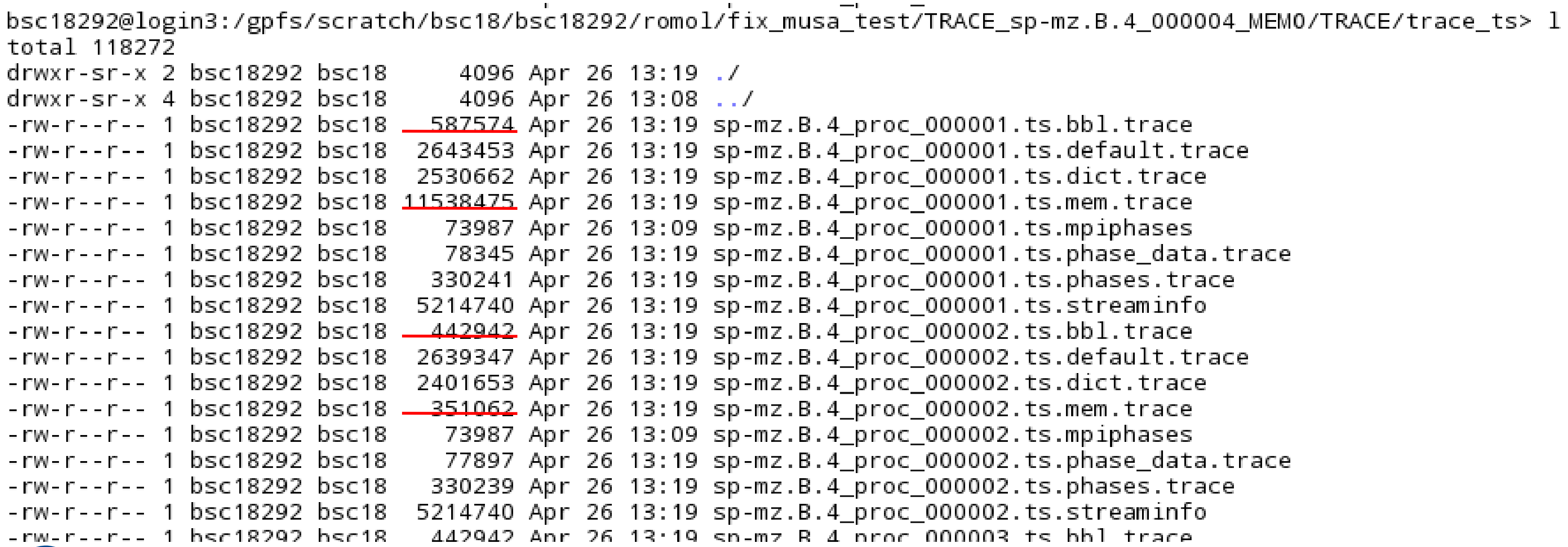
Step 1: Generating the Memory mode trace

What the script has generated:

- `logs_musa_generation-${jobid}.[out|err]`
- `TRACE_sp-mz.B.4_000004_MEMO/`
  - `LOGS/`
  - `TRACE/`
    - `trace_prv/`
    - **`trace_ts/`**
  - `SIMULATION/`
    - `A1_PRESIM/`
    - `A2_INTEGRATION_PRESIM/`

Step 1: Generating the Memory mode trace

- TaskSim trace:
  - One per rank:
  - .mem contains the list of addresses
  - .bbl contains the list of basic blocs executed



```
bsc18292@login3:/gpfs/scratch/bsc18/bsc18292/romol/fix_musa_test/TRACE_sp-mz.B.4_000004_MEMO/TRACE/trace_ts> l
total 118272
drwxr-sr-x 2 bsc18292 bsc18     4096 Apr 26 13:19 ./
drwxr-sr-x 4 bsc18292 bsc18     4096 Apr 26 13:08 ../
-rw-r--r-- 1 bsc18292 bsc18   587574 Apr 26 13:19 sp-mz.B.4_proc_000001.ts.bbl.trace
-rw-r--r-- 1 bsc18292 bsc18  2643453 Apr 26 13:19 sp-mz.B.4_proc_000001.ts.default.trace
-rw-r--r-- 1 bsc18292 bsc18  2530662 Apr 26 13:19 sp-mz.B.4_proc_000001.ts.dict.trace
-rw-r--r-- 1 bsc18292 bsc18 11538475 Apr 26 13:19 sp-mz.B.4_proc_000001.ts.mem.trace
-rw-r--r-- 1 bsc18292 bsc18    73987 Apr 26 13:09 sp-mz.B.4_proc_000001.ts.mpiphases
-rw-r--r-- 1 bsc18292 bsc18    78345 Apr 26 13:19 sp-mz.B.4_proc_000001.ts.phase_data.trace
-rw-r--r-- 1 bsc18292 bsc18   330241 Apr 26 13:19 sp-mz.B.4_proc_000001.ts.phases.trace
-rw-r--r-- 1 bsc18292 bsc18  5214740 Apr 26 13:19 sp-mz.B.4_proc_000001.ts.streaminfo
-rw-r--r-- 1 bsc18292 bsc18   442942 Apr 26 13:19 sp-mz.B.4_proc_000002.ts.bbl.trace
-rw-r--r-- 1 bsc18292 bsc18  2639347 Apr 26 13:19 sp-mz.B.4_proc_000002.ts.default.trace
-rw-r--r-- 1 bsc18292 bsc18  2401653 Apr 26 13:19 sp-mz.B.4_proc_000002.ts.dict.trace
-rw-r--r-- 1 bsc18292 bsc18   351062 Apr 26 13:19 sp-mz.B.4_proc_000002.ts.mem.trace
-rw-r--r-- 1 bsc18292 bsc18    73987 Apr 26 13:09 sp-mz.B.4_proc_000002.ts.mpiphases
-rw-r--r-- 1 bsc18292 bsc18    77897 Apr 26 13:19 sp-mz.B.4_proc_000002.ts.phase_data.trace
-rw-r--r-- 1 bsc18292 bsc18   330239 Apr 26 13:19 sp-mz.B.4_proc_000002.ts.phases.trace
-rw-r--r-- 1 bsc18292 bsc18  5214740 Apr 26 13:19 sp-mz.B.4_proc_000002.ts.streaminfo
-rw-r--r-- 1 bsc18292 bsc18   442942 Apr 26 13:19 sp-mz.B.4_proc_000003.ts.bbl.trace
```

Step 2: Executing a Memory mode simulation

- Go to the simulation folder:

```
$> cd TRACE_sp-mz.B.4_000004_MEMO/SIMULATION/A1_PRESIM
```

- Generate the greasy command files:

```
$> ./generate_musa_presim.bash
```

- When finished submit all Burst mode simulations:

```
$> sbatch launch_all_musa_presims.bash
```

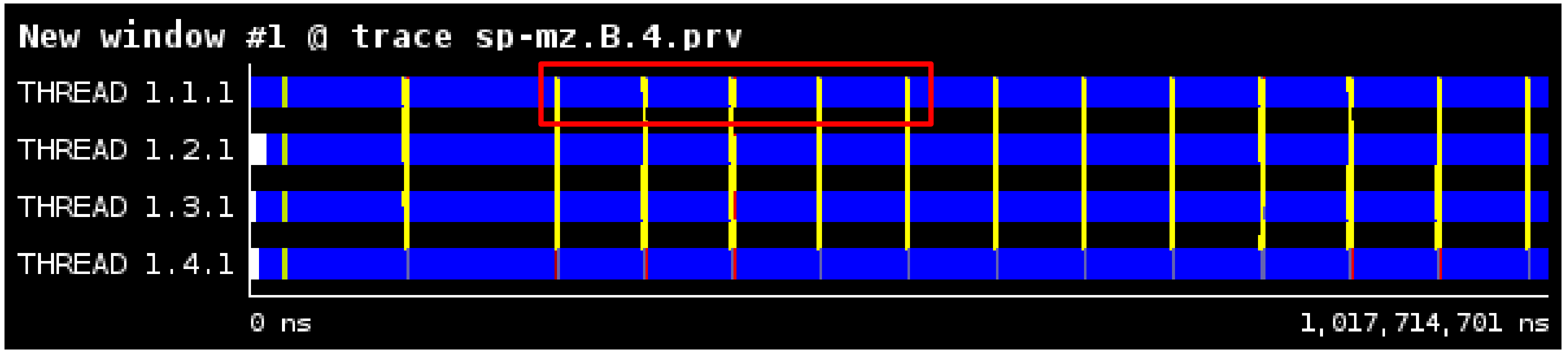
- If something fails:

```
$> ./cleanup.bash
```

Step 2: Executing a Memory mode simulation

What is actually happening:

- We run detailed simulations for the phases between mpi events for which we have detailed information.
- For all the phases, TaskSim runs on Burst mode
- In parallel



Step 2: Executing a Memory mode simulation

- Pre-simulation results:
  - musa_out_sp-mz.B.4_000001/

    mn4_musa_000001_BRST/mn4_musa_000001_BRST.dat

    mn4_musa_000001_MEMO/mn4_musa_000001_MEMO.dat
  - RANK:PHASE_ID:777:0:MEMORY_MODE:DURATION

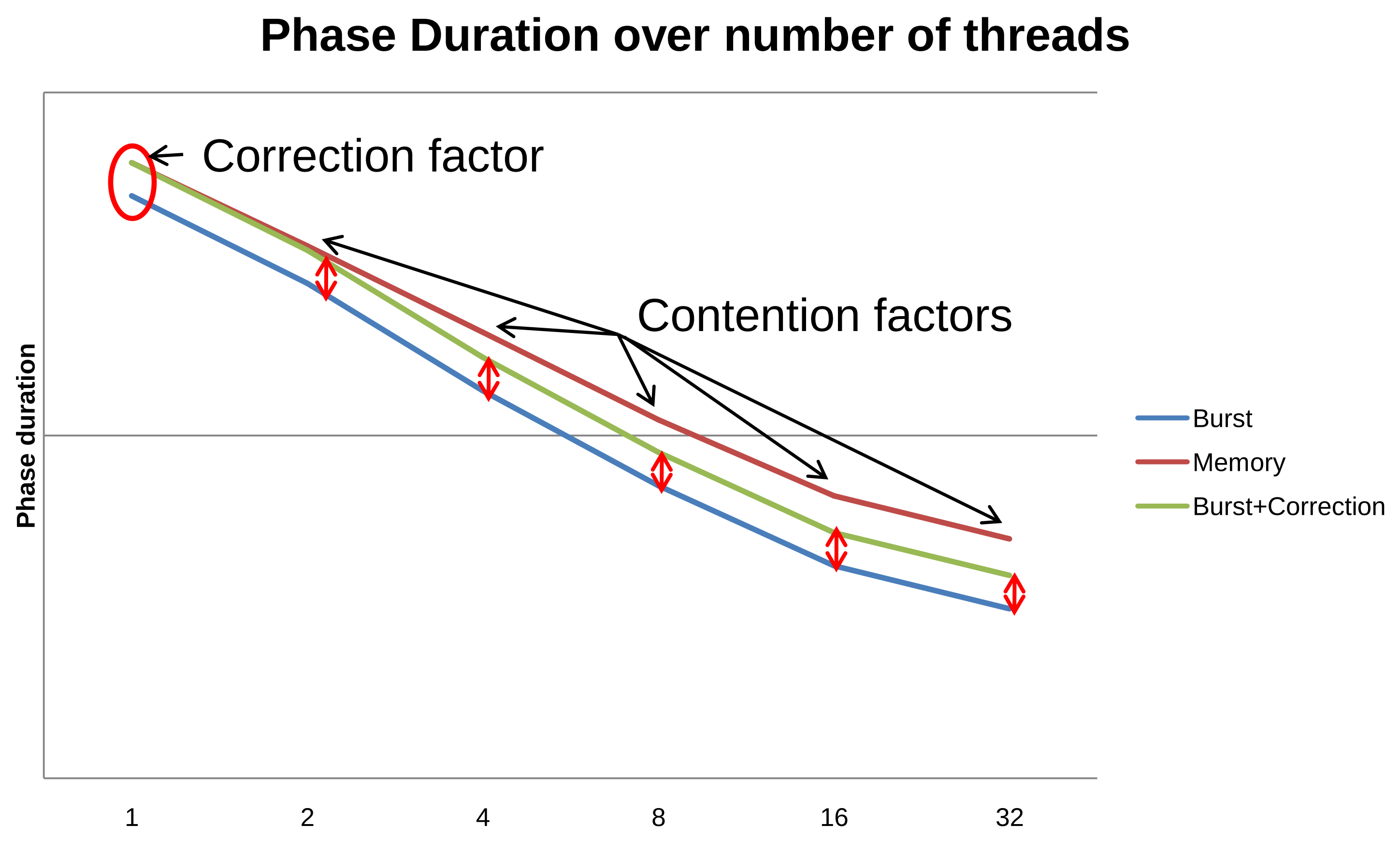
Step 3: Integrating the memory mode simulation

- We use two correction factors:
  - Memory/Burst bias ratio (extracted with 1 thread).
  - Memory contention ratio (each simulation its own).
- To execute the correction:

```
$> ./extrapolate_burst_duration_mn4_musa.bash
```

- This will generate a set of TOTAL files with the Memory durations and the corrected burst durations

Step 3: Integrating the memory mode simulation



Step 3: Integrating the memory mode simulation

- Integrating the results with Dimemas

```
$> cd TRACE_sp-mz.B.4_000004_MEMO/SIMULATION/
   A2_INTEGRATION_PRESIM
$> ./integrate_dimemas_simulations.bash
```

- This will:
  - Run Dimemas with the original extrae trace
  - Replace original phase duration with TaskSim Memory+Burst results.
  - Generate Paraver traces for each configuration

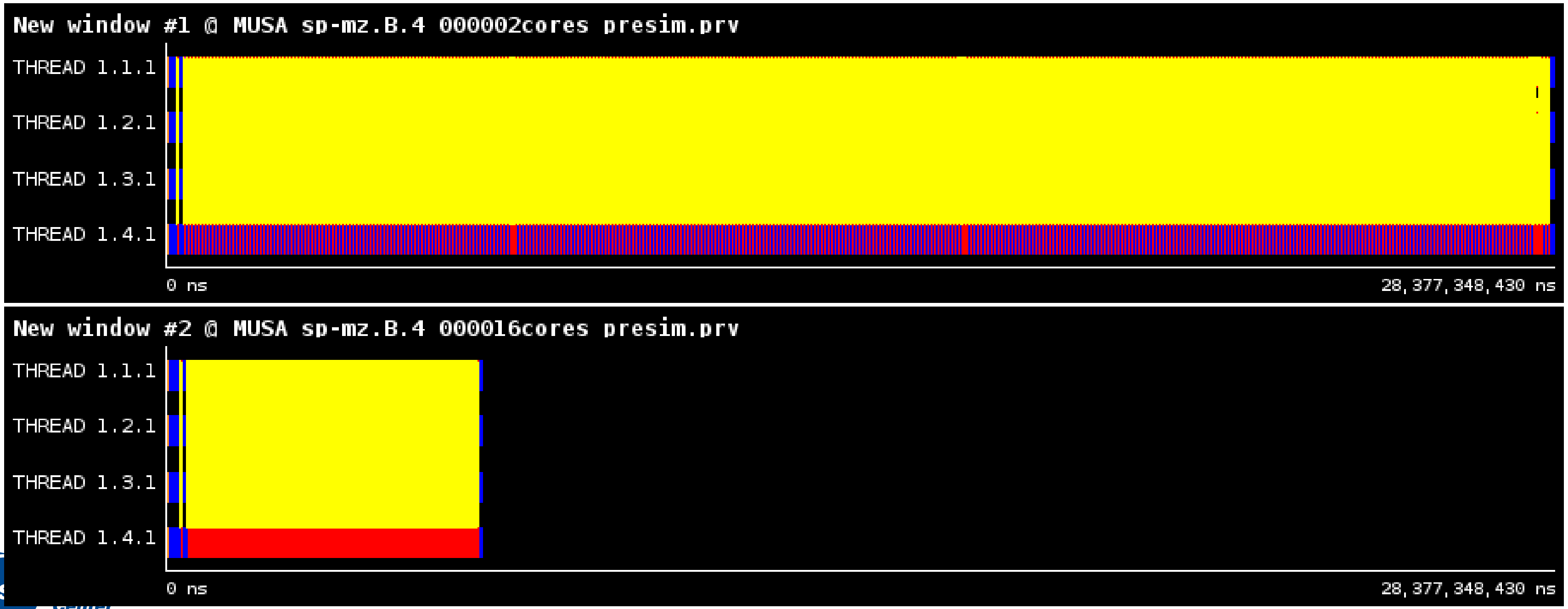
Step 4: Result analysis

- Open the Paraver trace:

```
$> module load paraver

$> cd TRACE_sp-mz.B.4_000004_MEMO/SIMULATION/
   A2_INTEGRATION_PRESIM/trace_SIMULATED

$> wxparaver MUSA_sp-mz.B.4_000008cores_presim.prv
```



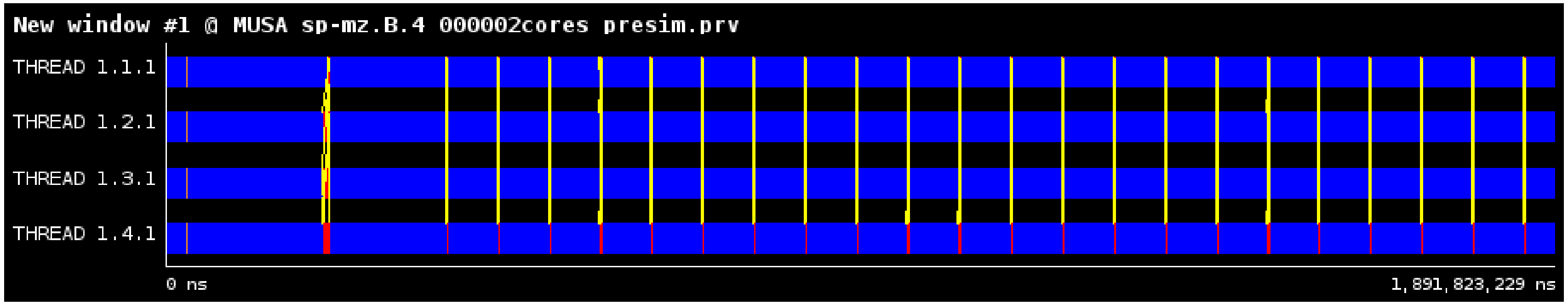
Step 4: Result analysis

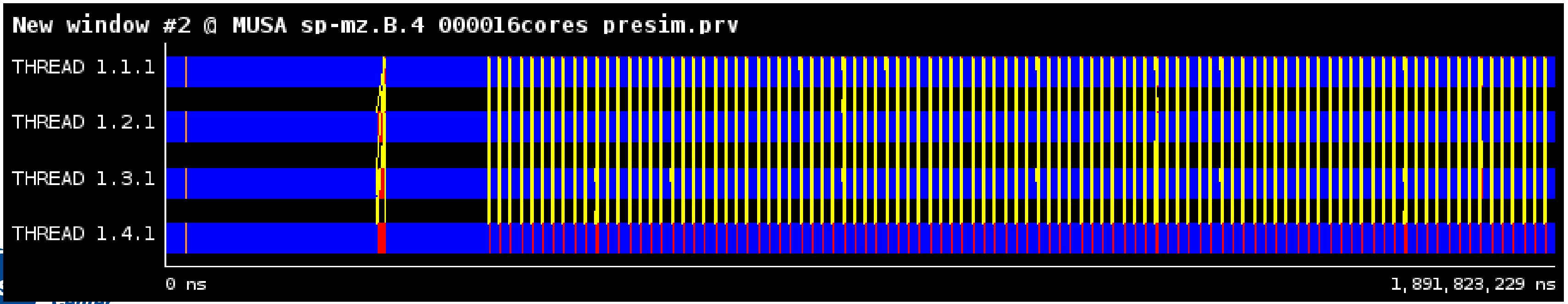
Open the Paraver trace:

```
$> module load paraver

$> cd TRACE_sp-mz.B.4_000004_MEMO/SIMULATION/
  A2_INTEGRATION_PRESIM/trace_SIMULATED

$> wxparaver MUSA_sp-mz.B.4_000008cores_mn4_musa_presim.prv
```

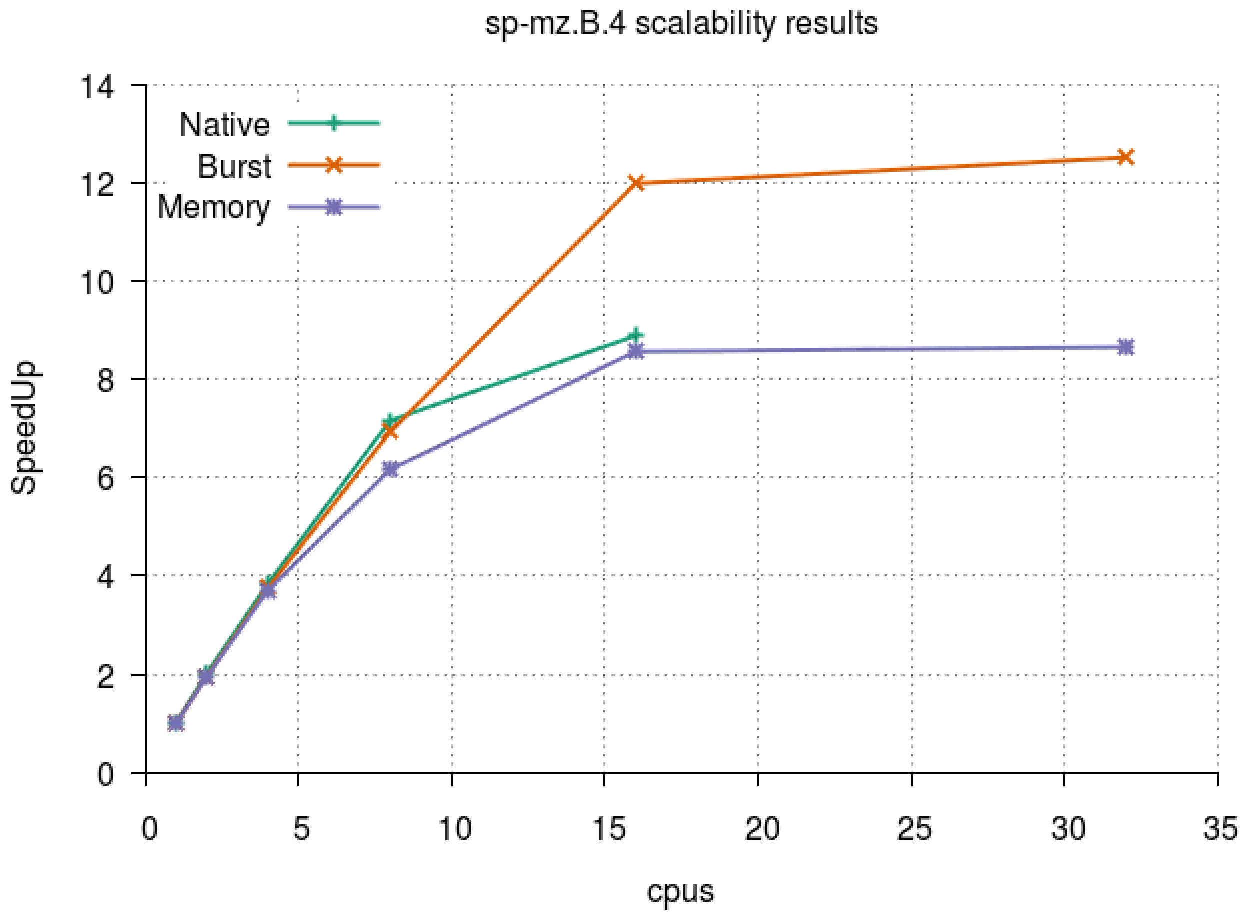




Step 4: Result analysis

Generate the speedup graph:

```
$> ./generate_speedup_graph.bash
```



Step 4: Result analysis

Conclusions

- We can simulate theoretical architectures very fast
- We can tune the degree of accuracy
- But remember, this is only a FIRST APROACH
- Every phase is simulated independently: cold caches
- We know and correct certain errors:
  - Correction factors are applied to all phases (there are non-parallel phases!!!)
  - Rank-Phase simulation must be studied in detail
  - In some situations we are optimistic (vectorization, runtime phase duration)

Questions?