An Evaluation of Different I/O Techniques for Checkpoint/Restart

***Faisal Shahzad*, Markus Wittmann, Thomas Zeiser, Georg Hager, Gerhard Wellein**
Erlangen Regional Computing Center (RRZE), University of Erlangen-Nuremberg
Erlangen, Germany

May 24th 2013

Outline:

- **Motivation**
- **Fault Tolerance**
- **Asynchronous checkpointing**
- **Implementation and overhead estimation model**
- **Performance results**
- **Conclusion**

Challenge

- **Nowadays, the increasing computational capacity is mainly due to extreme level of hardware parallelism.**
- **The reliability of hardware components does not increase with the similar rate.**
- **With future machines, the Mean time to failure is expected to be in minutes and hours.**
- **Absence of fault tolerant environment will put precious data at risk.**

Fault Tolerance Approaches

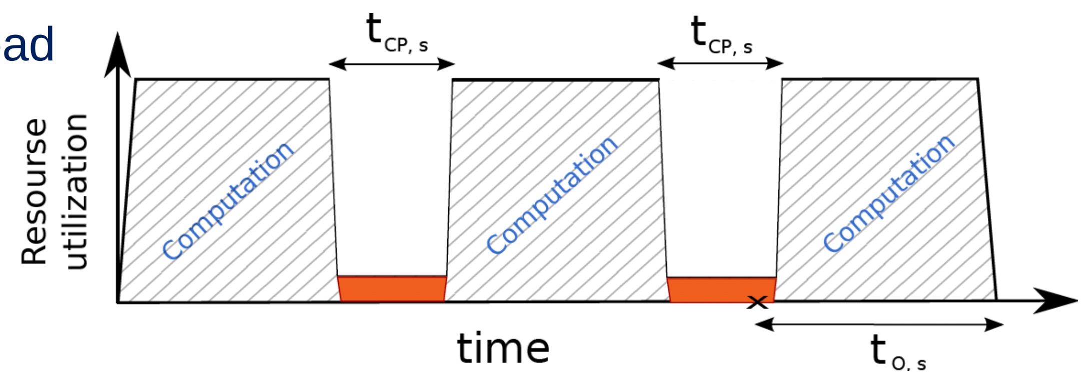
1. **Algorithm Based Fault Tolerance (ABFT)**
2. **Message Logging**
3. **Redundancy**
4. **Fault Prediction**
5. **Checkpoint/Restart (C/R)**
   - State of each process is periodically stored to a stable storage
   - In case of a failure, application can be restarted from these states
   - Three types* :
     1. Application level
     2. User level
     3. System level
   - Checkpoint overhead can be huge
   - Checkpoint frequency is a critical factor
   - Main bottleneck: I/O bandwidth

Each of these fault tolerance approaches carries overhead in terms of time and/or resources.

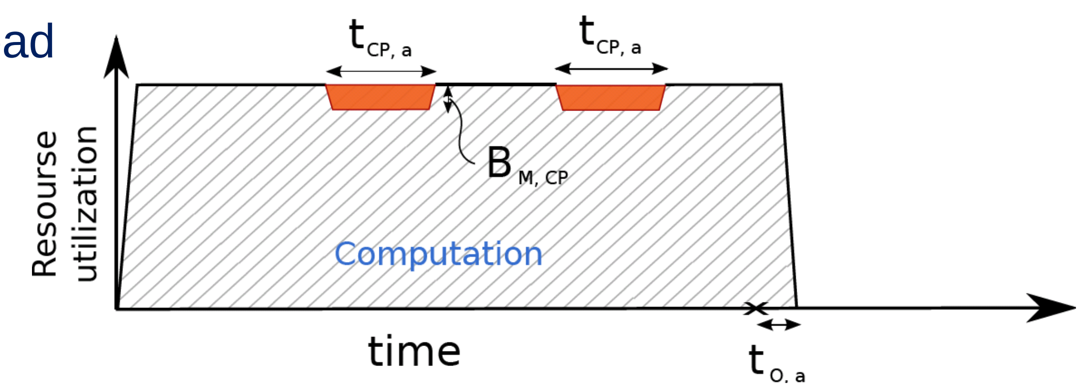
* J. Hursey, "Coordinated Checkpoint/Restart Process Fault Tolerance for MPI Applications on HPC Systems," Ph.D. dissertation, Indiana University, Bloomington, IN, USA, July 2010

Asynchronous checkpointing

- **Synchronous checkpointing:**
  - Computation halts for I/O time.
  - High execution time overhead



- **Asynchronous checkpointing:**
  - Using dedicated threads for performing asynchronous I/O
  - Low execution time overhead
  - An in-memory copy of checkpoint is required.



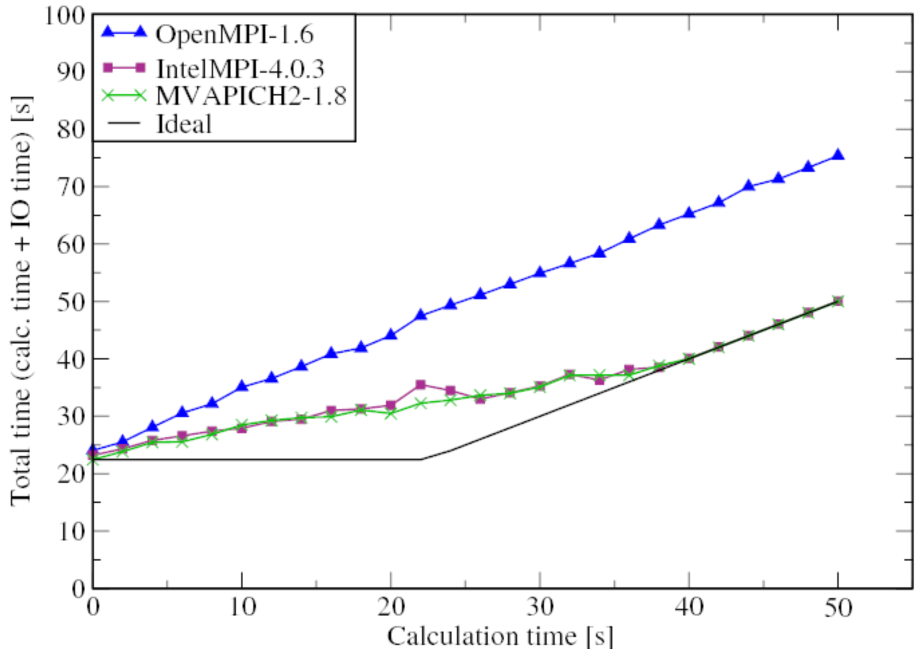
In principle, non-blocking MPI-IO can be used to perform asynchronous checkpointing!

Is non-blocking MPI-IO truly asnychronous?

Total time
(calc. time + IO time)

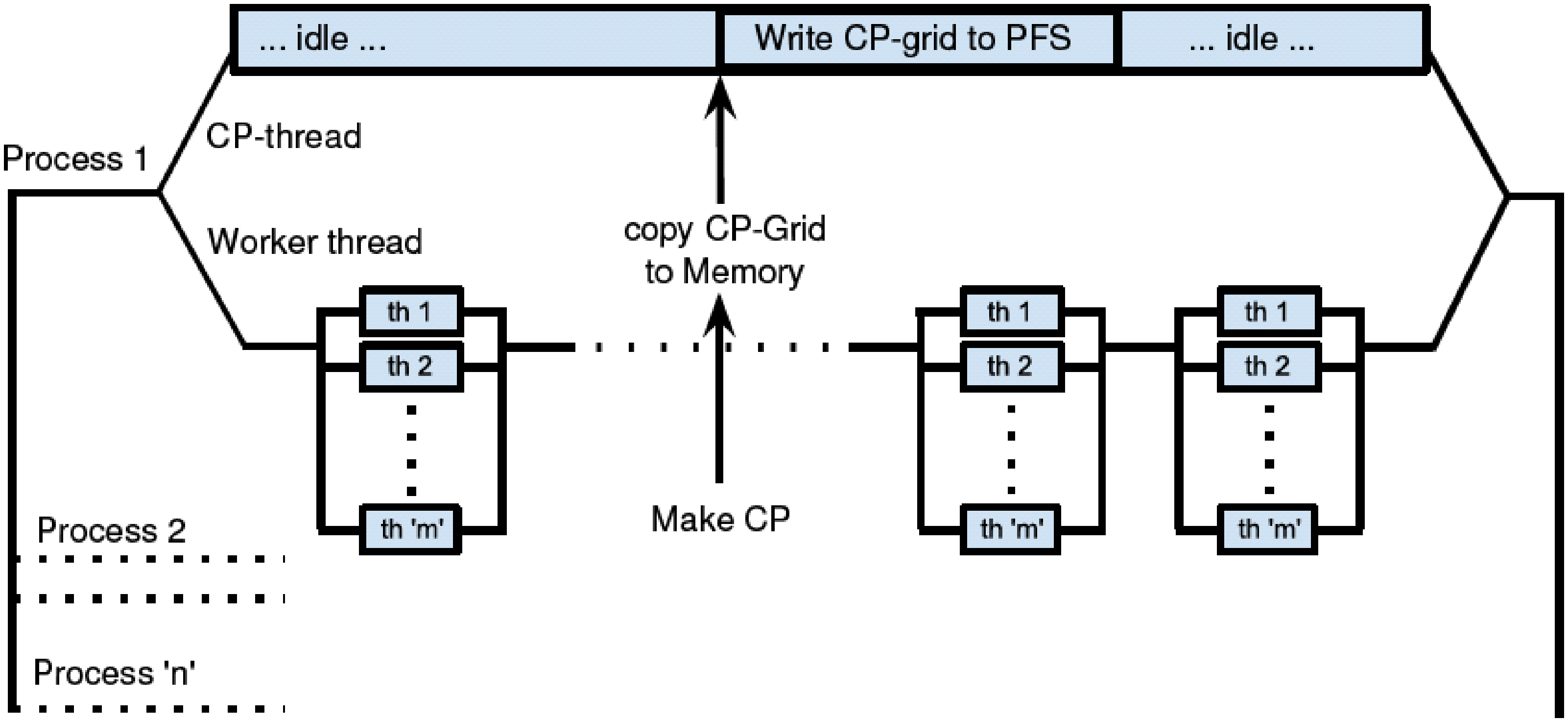
compute bound
calculation of configurable
amount of time (calc. time)

```
get_walltime_(&starttime_total);
MPI_File_iwrite(fh, buf, ndoubles_write,
                MPI_DOUBLE, &request);
perform_calc(calc_time);      //==== DUMMY CALCULATION
MPI_Wait( &request, &status );
get_walltime_(&endtime_total);
```



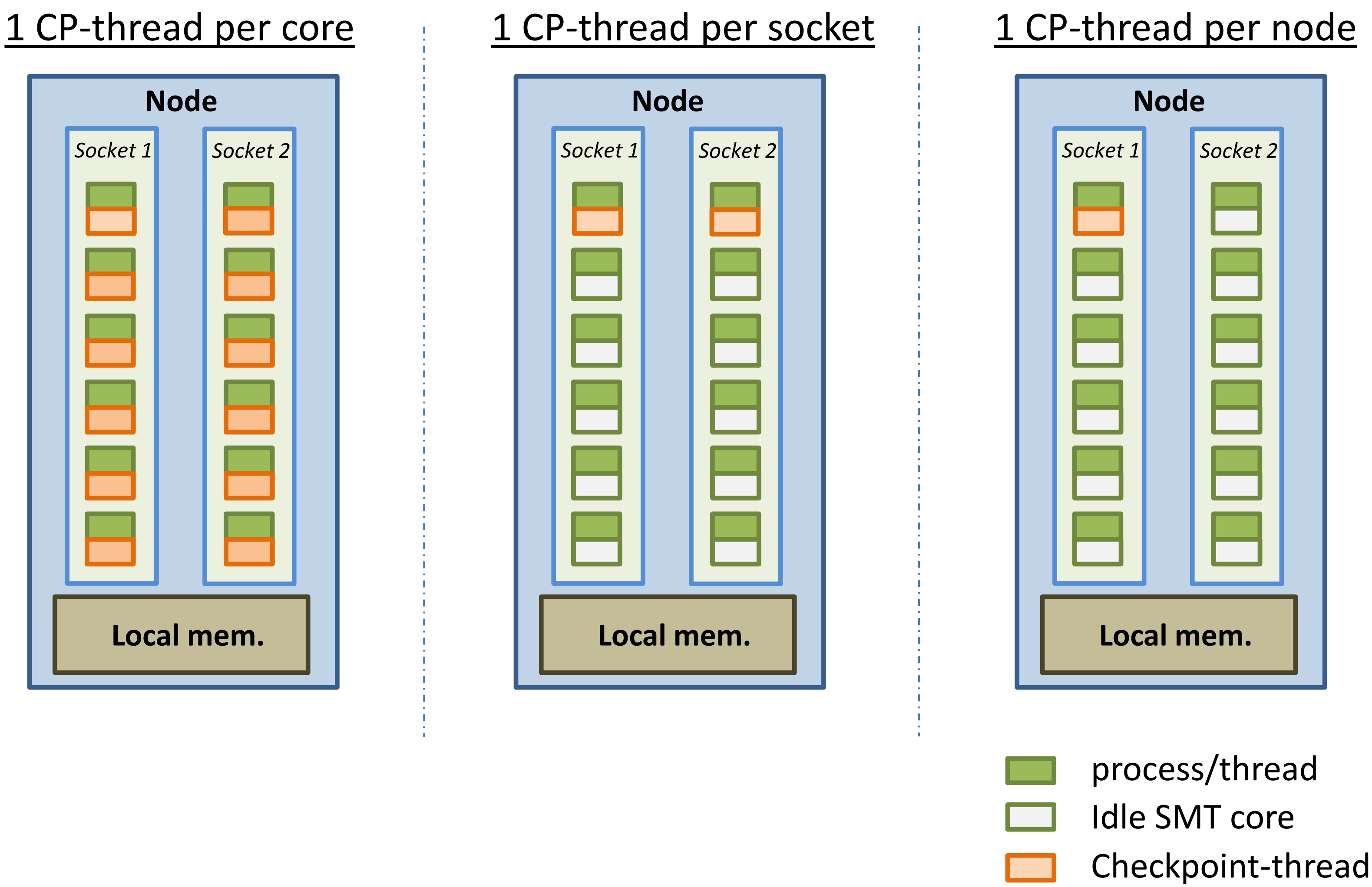
Asynchronous checkpointing by dedicated threads (I)

- **Hybrid (MPI/OpenMP) parallel approach**



Asynchronous checkpointing by dedicated threads (II)

- **Execution options with hybrid approach on SMT enabled CPUs**



- **Application:**
  - A prototype CFD solver based on Lattice Boltzmann Method (LBM).

- **Cluster:**
  - LiMa (Erlangen) : QDR Infiniband cluster, **500** nodes (Dual socket Intel Xeon 5650 “Westmere”), Lustre based PFS Bandwidth ~ **3GB/s**
  - HERMIT (Stuttgart): CRAY XE6, **3552** nodes (Dual socket AMD Opteron 6278 “Interlagos”), Lustre PFS ~ **150 GB/s**

- **Approaches:**
  - Synchronous CP
  - Asynchronous CP
  - Scalable Checkpoint Restart (SCR) Library

Implementation with LBM

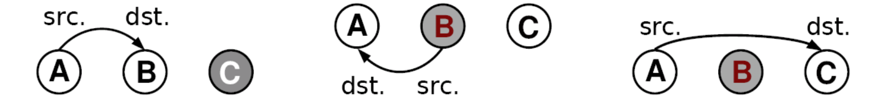
- **Worker-thread:**
  - Performs computation iterations
  - Creates in-memory copy of the checkpoint and signals the CP-thread.

```
//========WORKER THREAD =========//
 while(current_time_step<=timesteps){
     computation_step();
     apply_BoundaryCondition();
     if(current_time_step==checkpoint_iter){
         CP_temp_swap=src_grid;
         src_grid=CP_grid;
         CP_grid=dst_grid;
         signal_write_checkpoint();
     }
     if(current_time_step==(checkpoint_iter+1)){
         src_grid=CP_temp_swap;
     }
     switch_grid_pointers(dst_grid,src_grid);
     ++current_time_step;
 }
```

- **Checkpoint-thread:**
  - Waits for the signal from worker-thread.
  - Writes the checkpoint PFS.

```
//=========CP THREAD=========//
  while(!iteration_finished){
      wait_for_write_checkpoint_signal();
      if(signaled_write_checkpoint()){
          write_checkpoint_to_PFS();
      }
  }
```

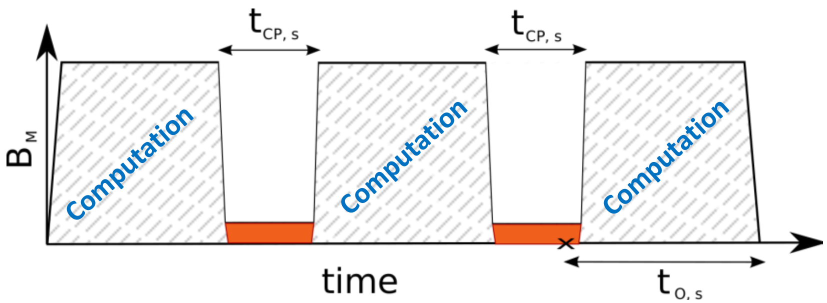
- **For “toggle grids” based stancil algorithm (e.g LBM), effective pointer switching can be used to avoid in-memory copy of the checkpoint.**



(a) *checkpoint* iteration (b) *checkpoint* + 1 iteration (c) *checkpoint* + 2 iteration

Checkpoint overhead estimation model (I)

Synchronous Checkpointing



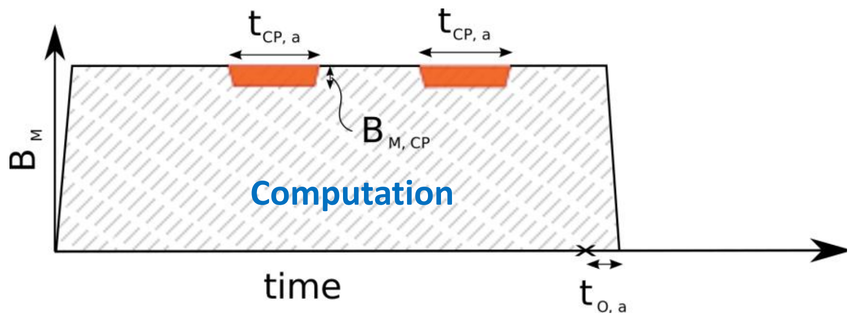
$$
\begin{aligned}
t_{O,s} &= \text{overhead for synchronous checkpoints} \\
t_{CP,s} &= \text{duration of a synchronous checkpoint} \\
S_{CP} &= \text{size of a single checkpoint in bytes} \\
B_{IO} &= \text{I/O bandwidth to the file system in bytes/s} \\
B_M &= \text{memory bandwidth of a node in bytes/s} \\
n &= \text{number of checkpoints}
\end{aligned}
$$

$$t_{O,s} = n \cdot t_{CP,s}$$

$$t_{O,s} = n \cdot \frac{S_{CP}}{B_{IO}}$$

For weak scaling, overhead is directly proportional to the number of nodes

Asynchronous Checkpointing



$$
\begin{aligned}
t_{O,a} &= \text{overhead for asynchronous checkpoints} \\
t_{CP,a} &= \text{duration of an asynchronous checkpoint} \\
S_{CP,node} &= \text{checkpoint size per node in bytes} \\
B_{M,CP} &= \text{memory bandwidth used for checkpoint-I/O in bytes/s}
\end{aligned}
$$

$$B_M \cdot t_{O,a} = n \cdot B_{M,CP} \cdot t_{CP,a}$$

For I/O purposes, the amount of data traffic (reads/writes) between memory and processor can be "m" times larger than the file size itself. Our study reveals this factor to be between 5-7 for OpemMPI (m=5-7).

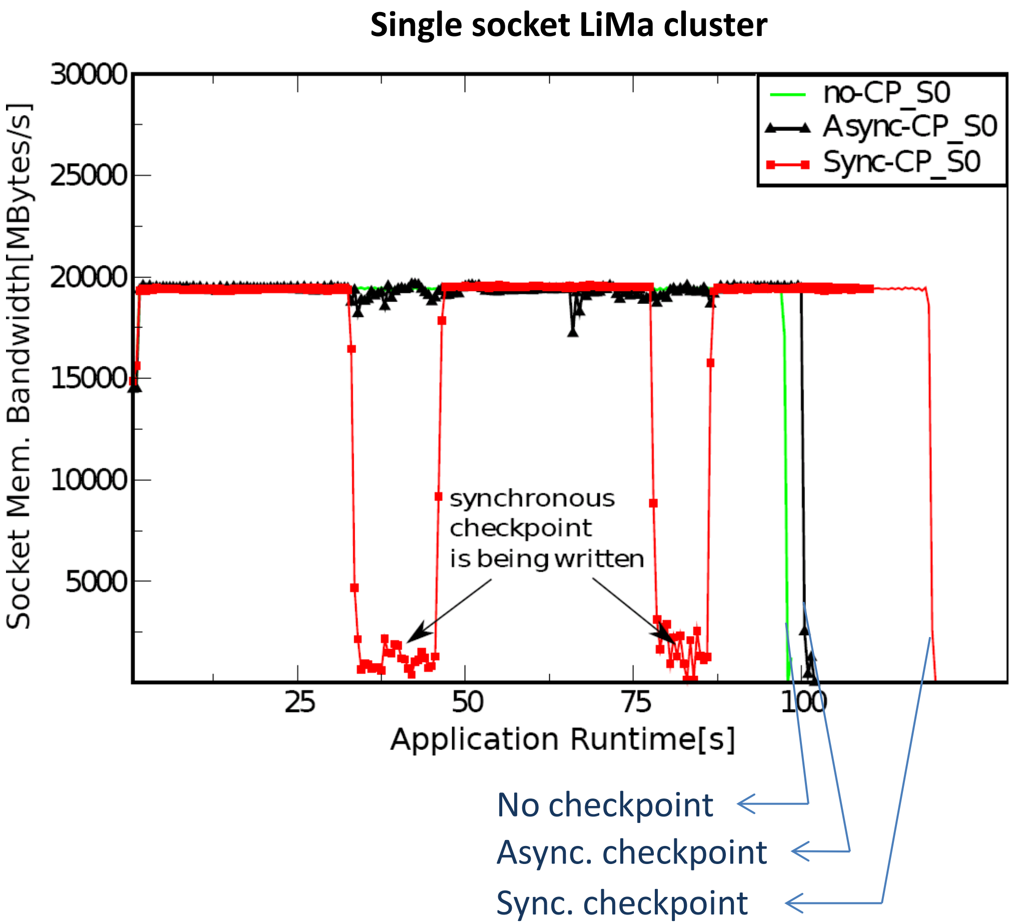
$$B_{M,CP} = \frac{m \cdot S_{CP,node}}{t_{CP,a}}$$

$$t_{O,a} = \frac{m \cdot S_{CP,node}}{B_M} \cdot n$$

Overhead remains constant for weak scaling.

Checkpoint overhead estimation model (II)

- **Validation of asynchronous overhead estimation model is done by using *likwid-perfctr** tool.**
  - Memory bandwidth of each socket is measured every 500ms.

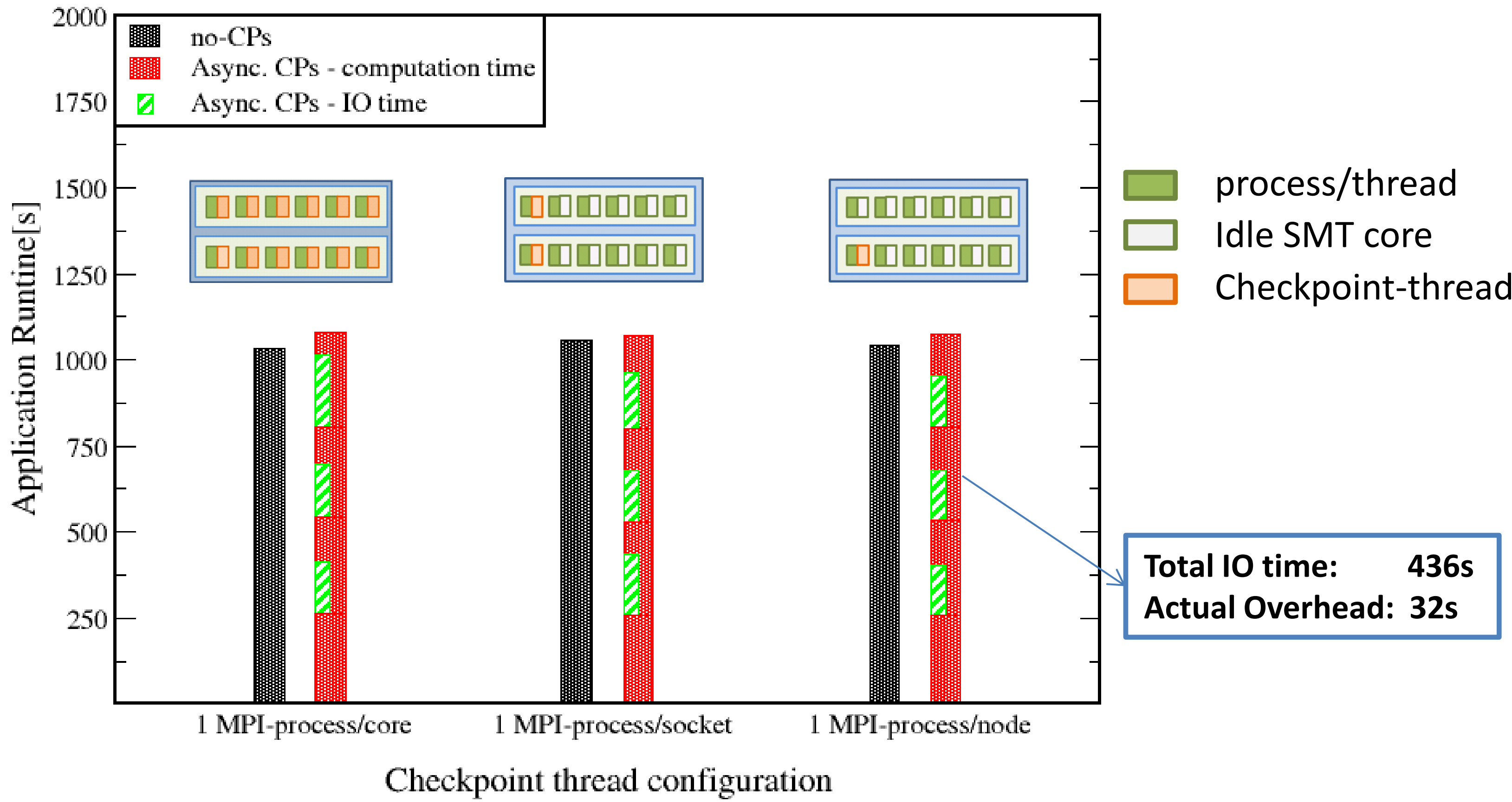


- **Estimated overhead:**
  - 2.2s (n=2, $S_{cp,node}$=6.25GB, B_M=40GB/s, m=7)
- **Actual overhead:**
  - 2.6s

* *https://code.google.com/p/likwid/*

- **Hybrid (MPI-OpenMP) configuration performance comparison**

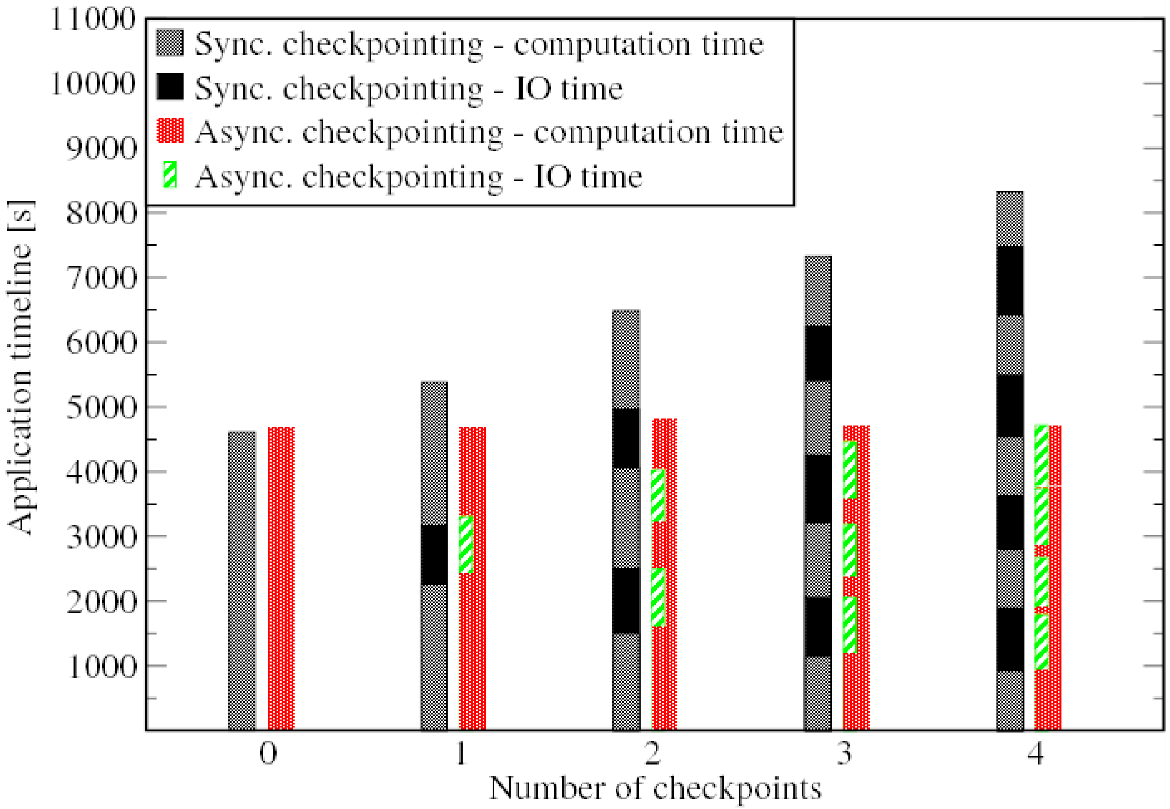
Cluster: LiMa, num. of nodes = 32, PFS = LXFS, Aggregated CP size = 200 GB/CP



Asynchronous vs. Synchronous Checkpointing

- **LiMa**

Num. of nodes = 128, np = 1536, PFS = LXFS, Aggregated CP size = 800GB/CP



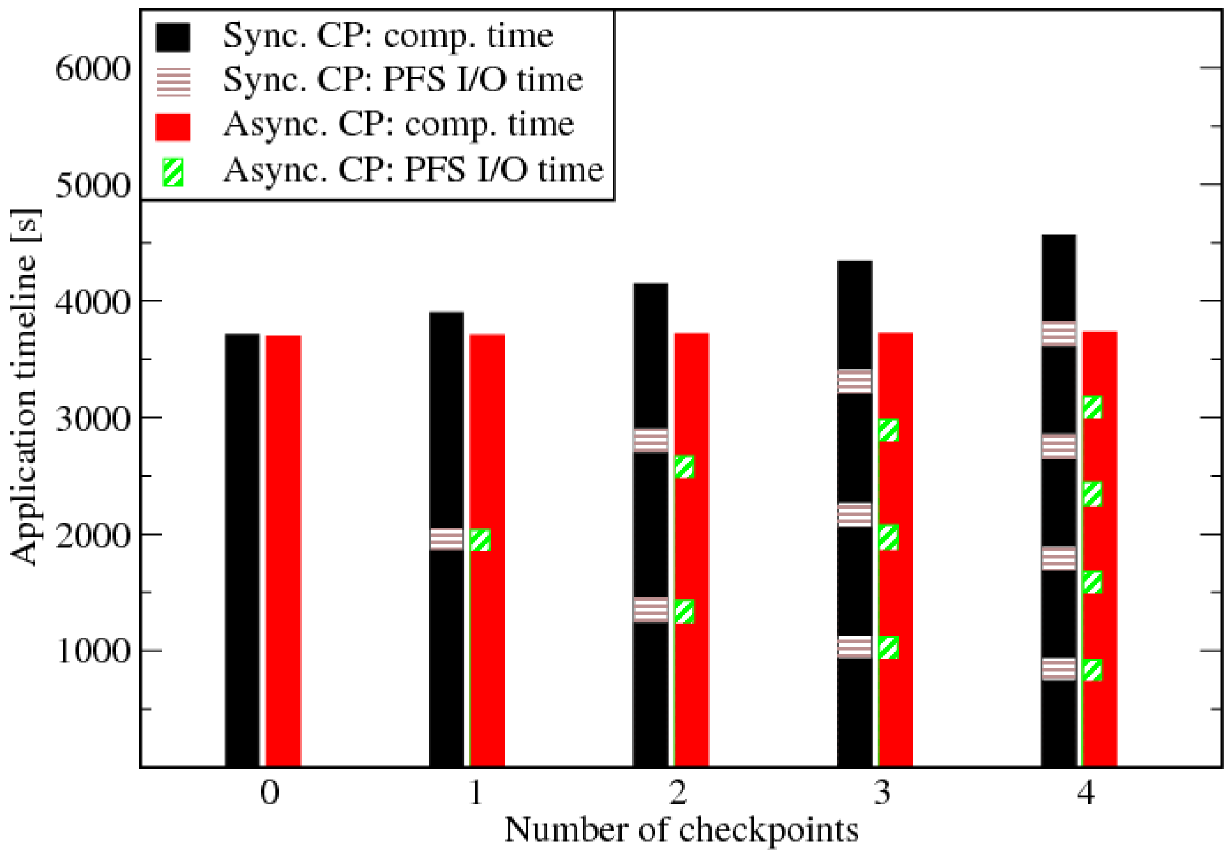
<u>% overhead:</u>

1 Sync. CP = 20 %

1 Async. CP = 0.4 %

- **HERMIT**

Num. of nodes = 256, np = 8192, PFS = Lustre, Aggregated CP size = 2.3TB/CP



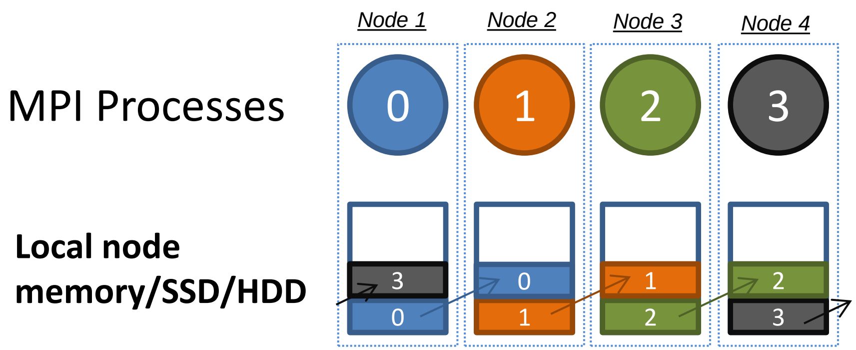
<u>% overhead:</u>

1 Sync. CP = 5.6%

1 Async. CP = 0.2 %

Scalable Checkpoint Restart

- **Scalable Checkpoint/Restart is a library developed by LLNL(Adam Moody)***
- **Key idea: Node-level checkpoints (memory, Hard disk)**
- **Checkpointing Features**
  - **LOCAL**
  - **PARTNER**

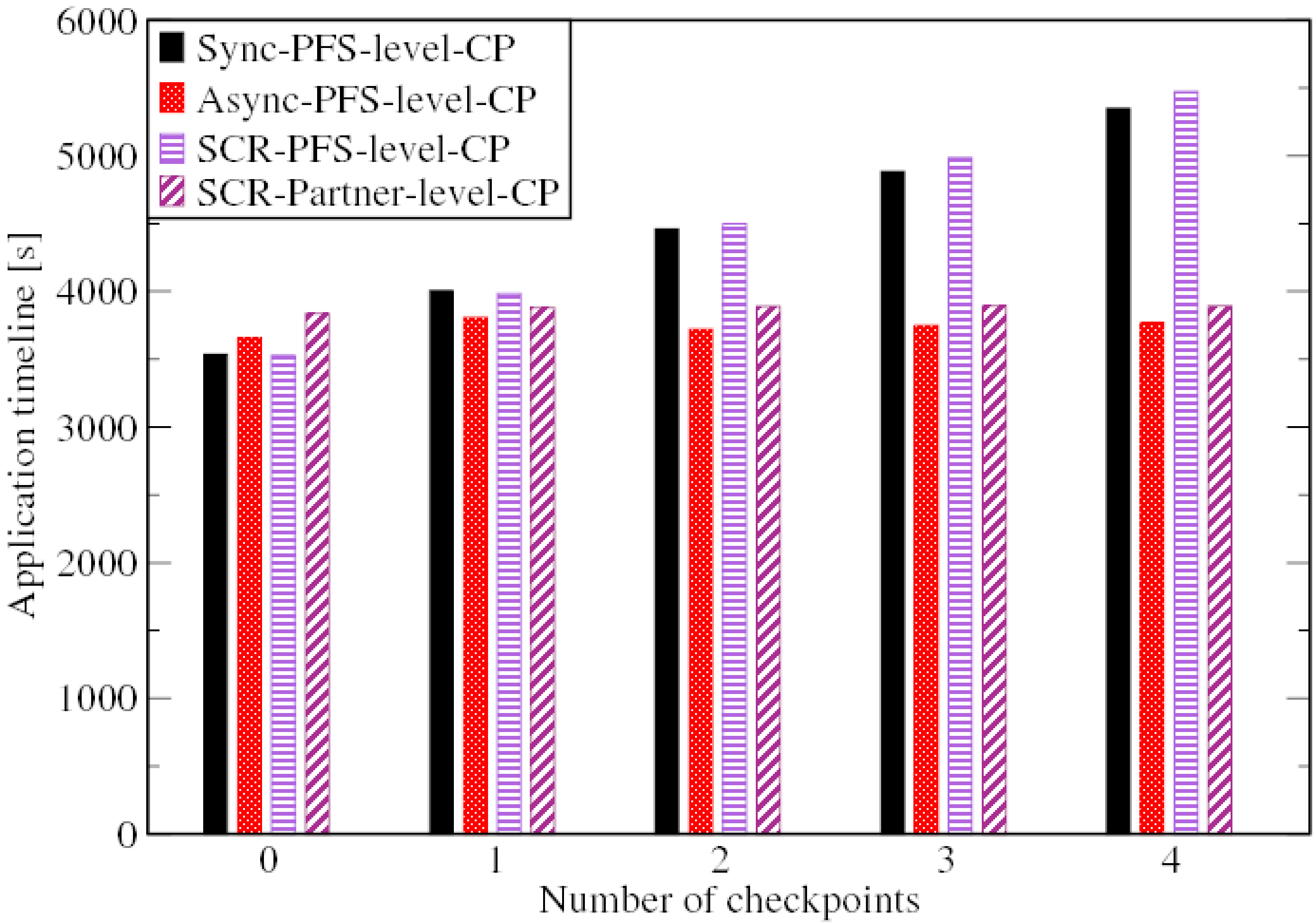


  - **PARTNER XOR**
  - **Parallel File System (PFS) level checkpoints**
    - To deal with catastrophic failures

* *http://sourceforge.net/projects/scalablecr/*

- **LiMa**

Num. of nodes = 128, PFS = LXFS, Aggregated CP size = 510 GB /CP



<u>% overhead:</u>

1 Sync. CP   = 13 %

1 Async. CP = 1.3 %

1 Partner. CP = 1 %

Conclusion:

- **Effective implementation of C/R and effective resource utilization can reduce overhead to minimum level.**

- **The overhead due to I/O bottlenecks can be reduced with asynchronous checkpointing approach.**

- **Although SCR on node-level is highly scalable, PFS-level checkpoints carry less overhead with asynchronous approach.**

Thank you!

Questions?