DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

Parallel Programming and Execution Models

CEA, DAM, DIF, F-91297 Arpajon, France

P. CARRIBAULT and M. PERACHE

Outline

- MPC framework
- Runtime Optimizations
- Programming model
- Tools

Context

- Starting point: legacy codes
  - Most used standards: MPI and/or OpenMP
  - Current architectures: petaflopic machines such as TERA100
  - Languages: C, C++ and Fortran
  - Large amount of existing codes and libraries
- Main target: ease the transition to Exascale for user codes and libraries
  - Provide efficient runtime to evaluate mix of programming models
    - Unique programming model for all codes and libraries may be a non-optimal approach
  - Provide smooth/incremental way to change large codes and associated libraries
    - Avoid full rewriting before any performances results
    - Keep existing libraries at their full current performances coupled with application trying other programming model
    - Example: MPI application calling OpenMP-optimized schemes/libraries
- Multi-Processor Computing (MPC)

MPC FRAMEWORK

MPC Overview

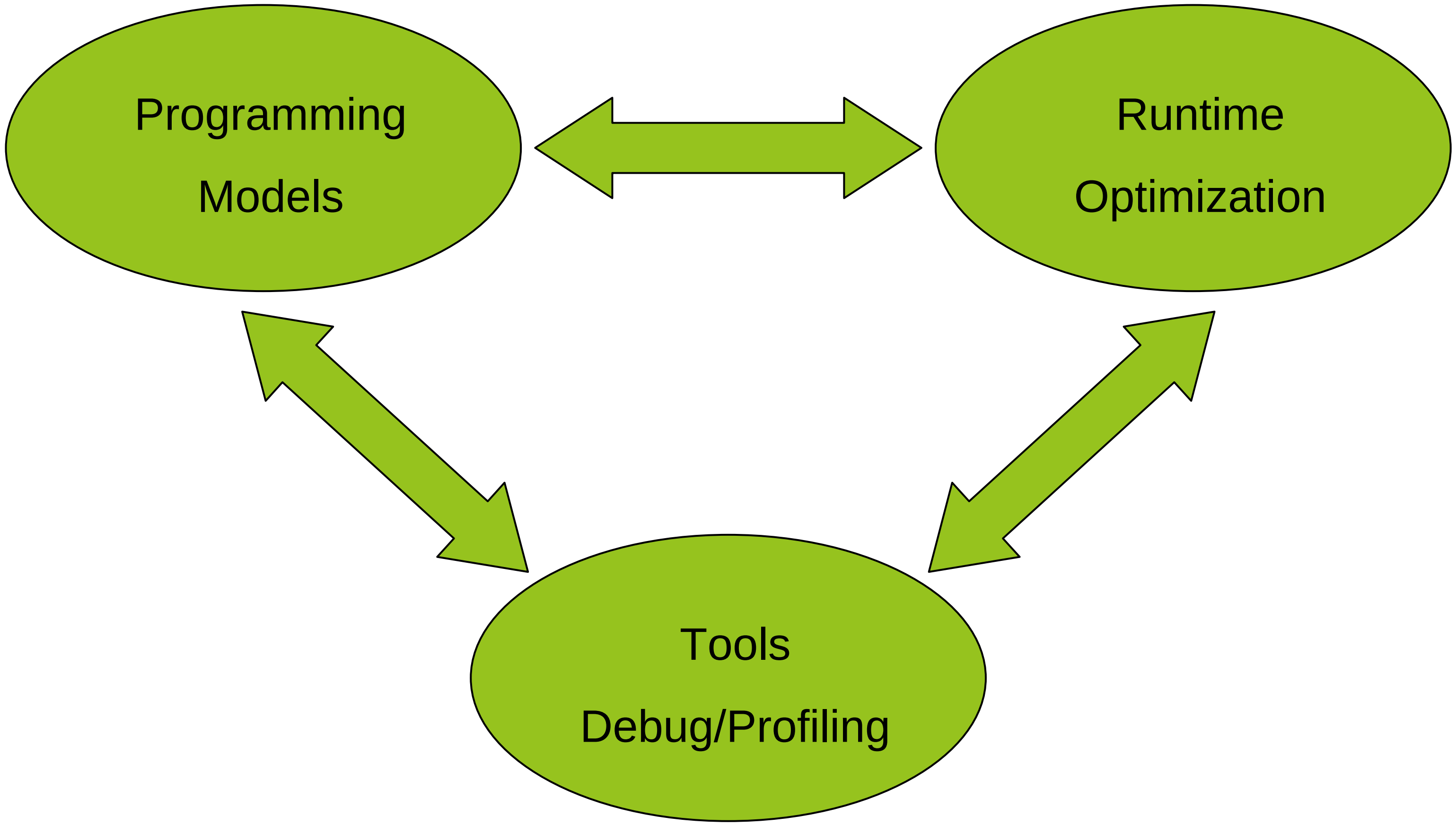
- Multi-Processor Computing (MPC) framework
  - Runtime system and software stack for HPC
  - Project started in 2003 at CEA/DAM (PhD work)
  - Team as of October 2012 (CEA/DAM and ECR Lab)
    - 3 research scientists, 2 postdoc fellows, 8 PhD students, 1 apprentice, 1 engineer
  - Freely available at http://mpc.sourceforge.net (version 2.4.0)
    - Contact: marc.perache@cea.fr or patrick.carribault@cea.fr

- Summary
  - Unified parallel runtime for clusters of NUMA machines

- Unification of several parallel programming models
  - MPI, POSIX Thread, OpenMP, …

- Integration with other HPC components
  - Parallel memory allocator, patched GCC, patched GDB, HWLOC, …

MPC Framework



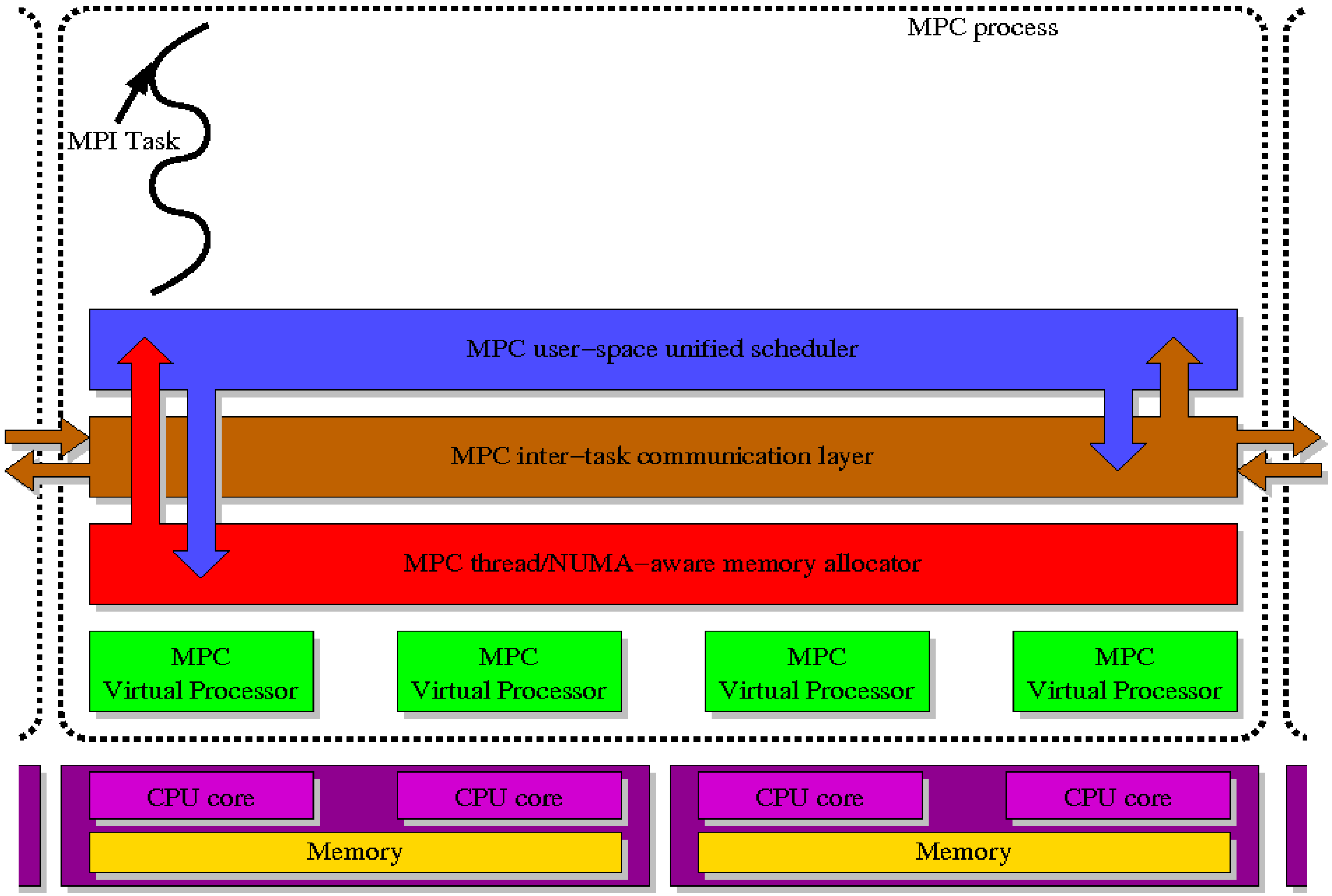
Runtime Optimization

Runtime Optimization

- Provide standard programming models
  - MPI
  - OpenMP
  - PThread (integration with other runtimes)
- Optimized runtime for current architectures
  - Petascale architectures: T100, Curie
- Deal with manycore issues
  - Manycore scheduler optimization
  - Memory-consumption reduction
  - Memory allocation in multithread context
- Provide mechanisms to integrate multiple programming models
  - Applications, libraries, numerical schemes using different programming model to reach high scalability

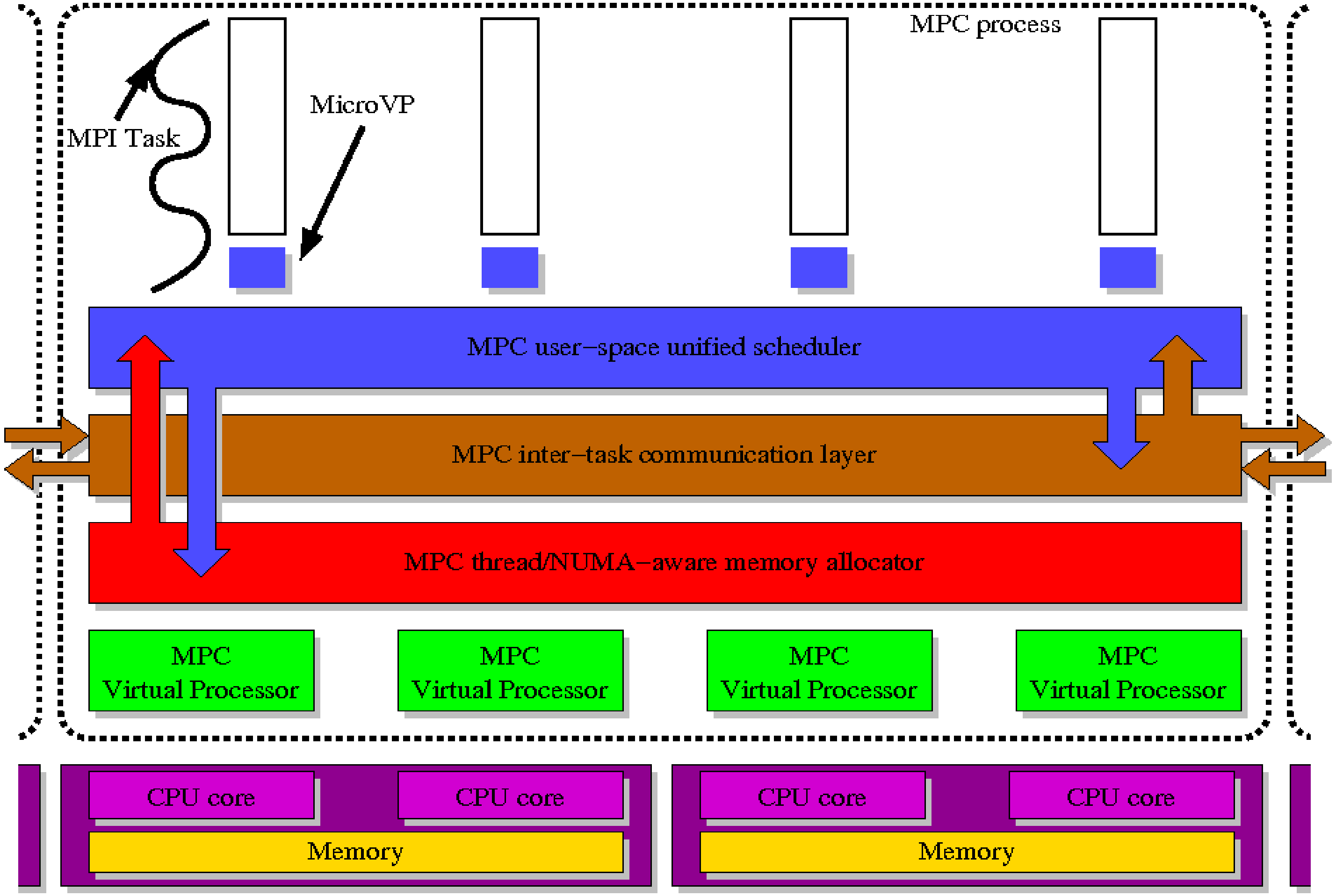
MPC Execution Model: Example #1

- Application with 1 MPI task



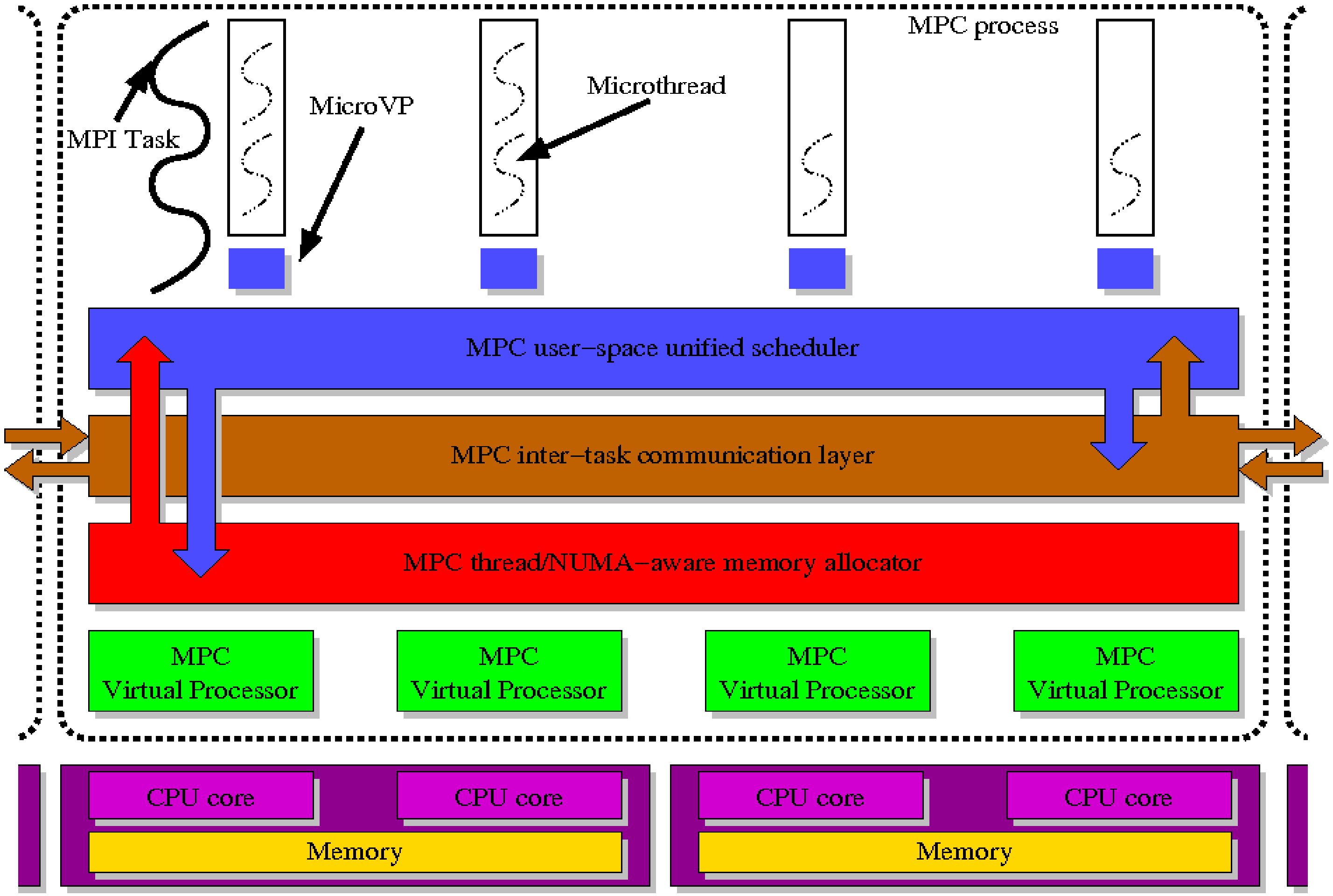
MPC Execution Model : Example #1

- Initialization of OpenMP regions (on the whole node)



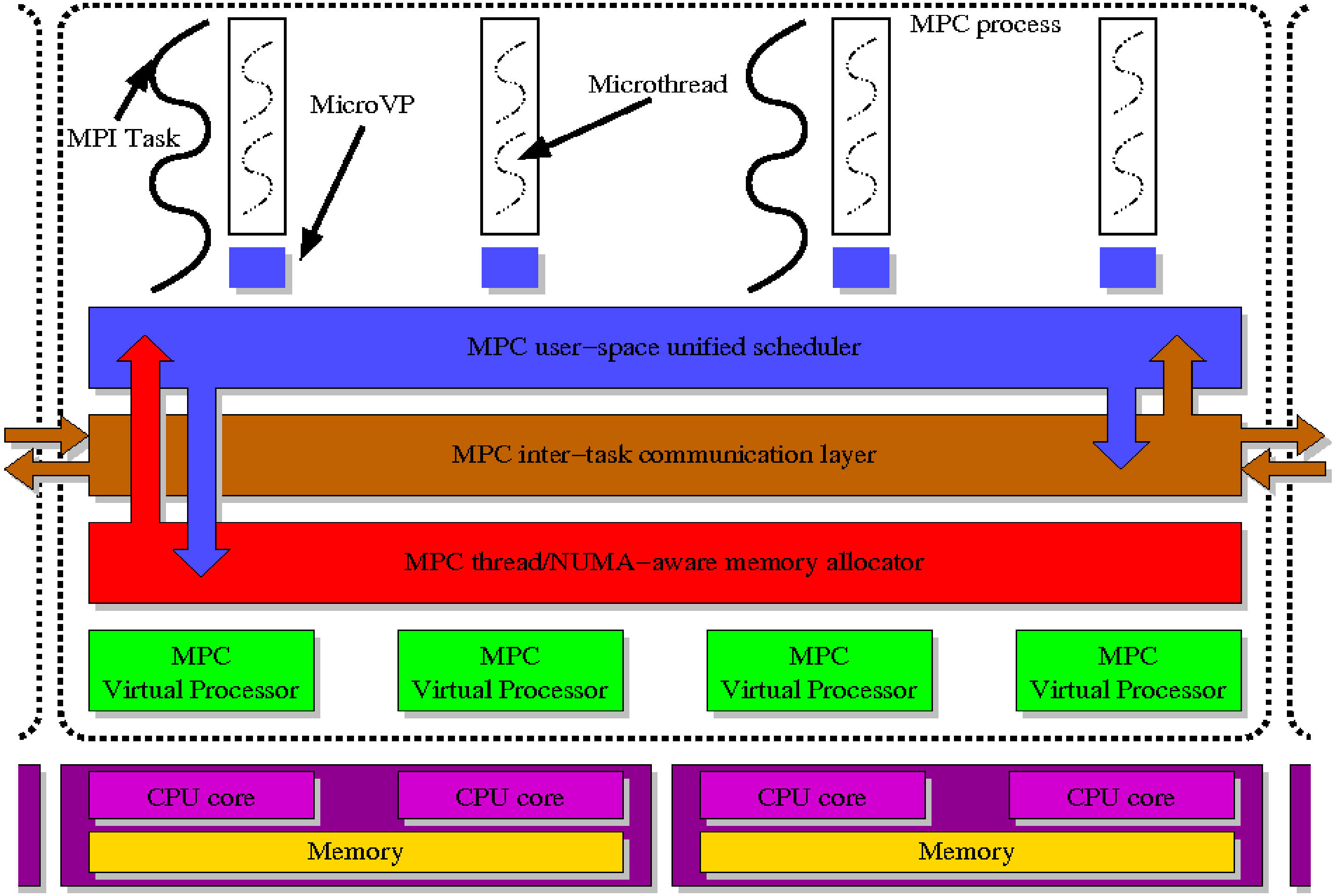
MPC Execution Model : Example #1

- Entering OpenMP parallel region w/ 6 threads



MPC Execution Model: Example #2

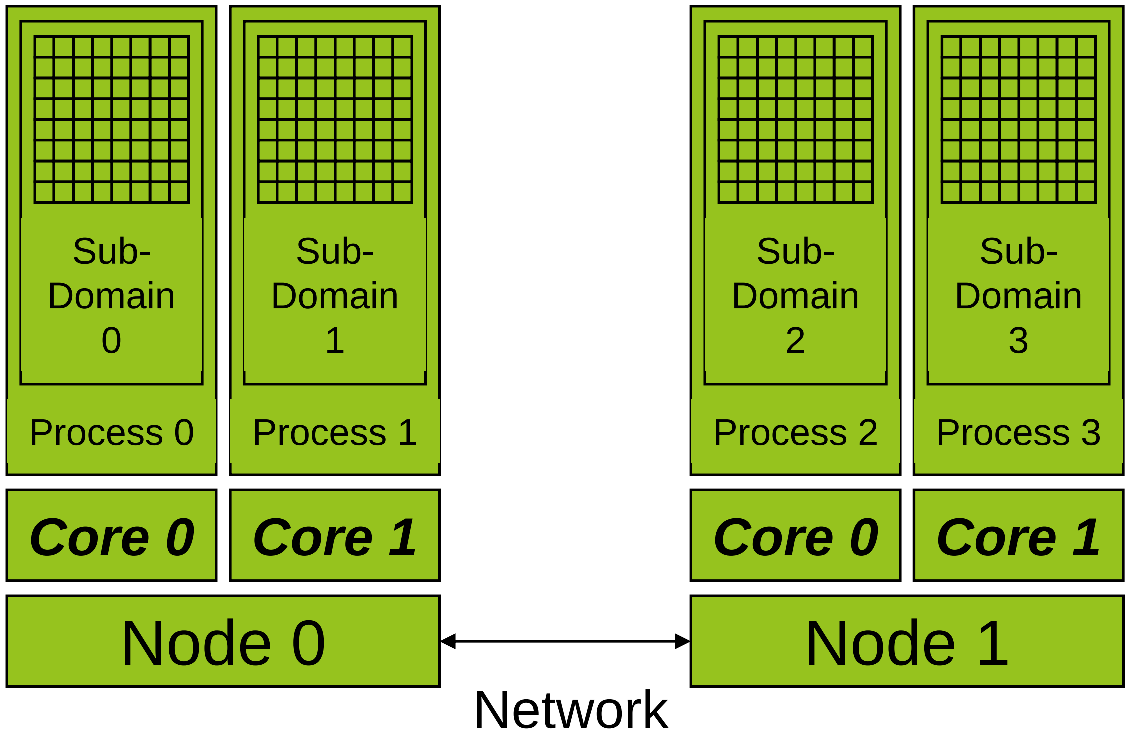
- 2 MPI tasks + OpenMP parallel region w/ 4 threads (on 2 cores)



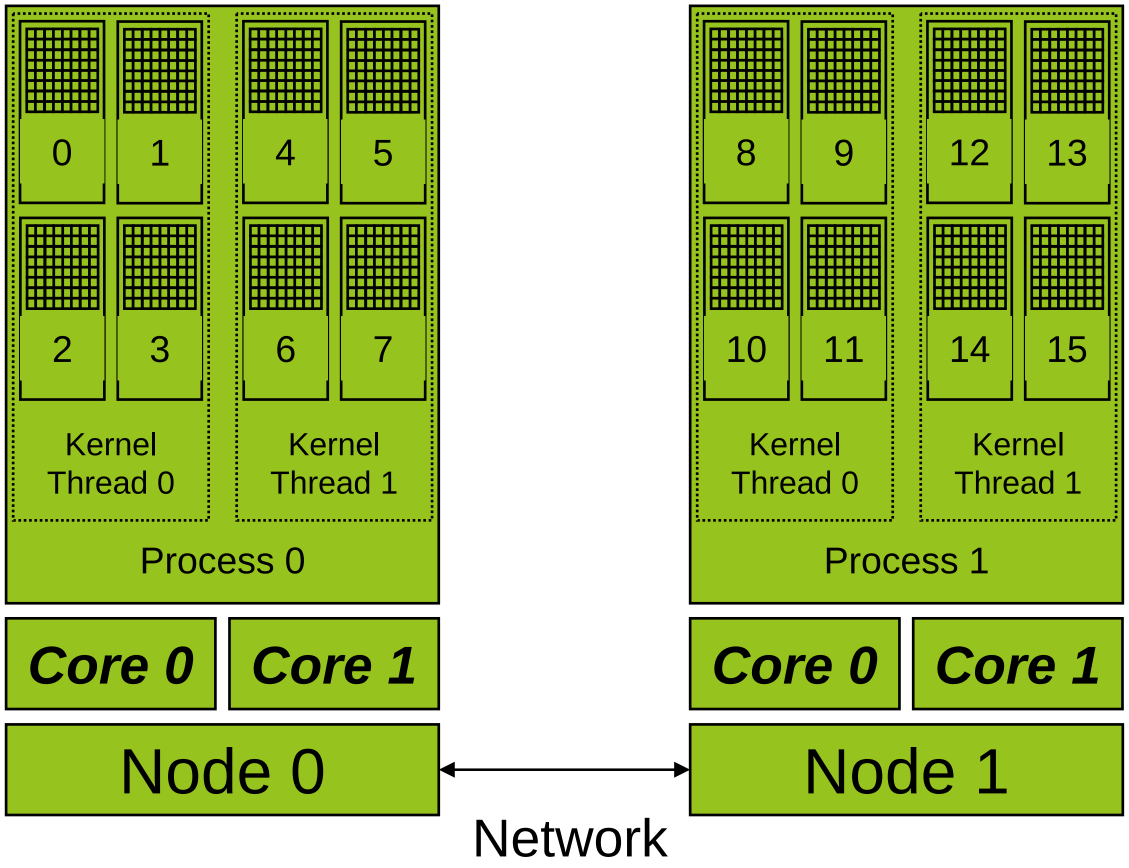
MPI

- **Goals**
  - Smooth integration with multithreaded model
  - Low memory footprint
  - Deal with unbalanced workload
- **MPI 1.3**
  - Fully MPI 1.3 compliant
- **Thread-based MPI**
  - Process virtualization
  - Each MPI process is a thread
- **Thread-level feature**
  - From MPI2 standard
  - Handle up to MPI_THREAD_MULTIPLE level (max level)
  - Easier unification with PThread representation
- **Inter-process communications**
  - Shared memory within node
  - TCP, InfiniBand
- **Tested up to 80,000 cores with various HPC codes**

MPI Approach

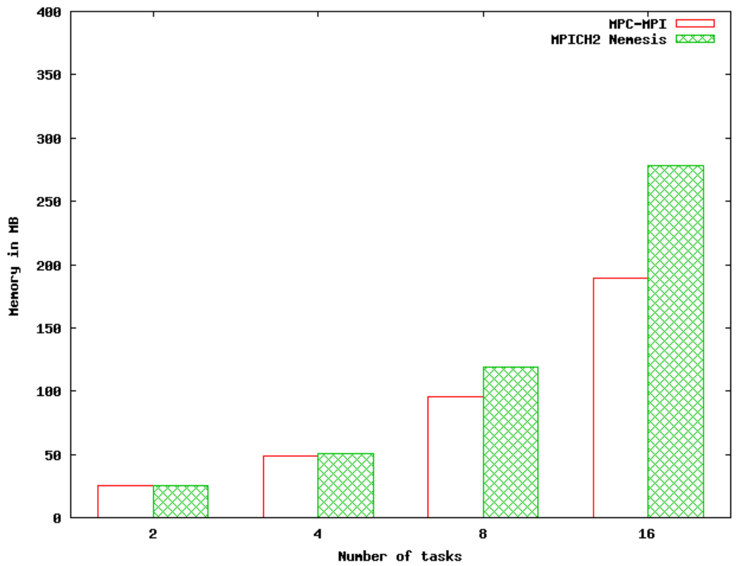


MPC Approach



MPI (Cont.)

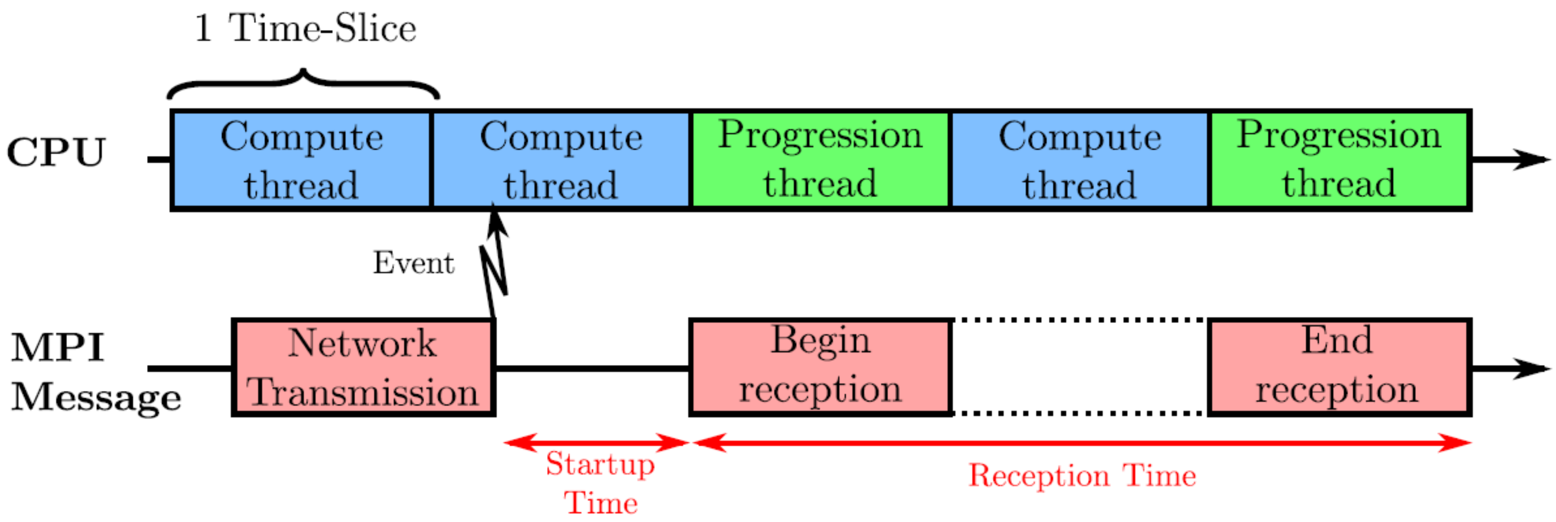
- Optimizations
  - Good integration with multithreaded model [EuroPar 08]
    - No spin locks: programming model fairness without *any* busy waiting
    - *Scheduler-integrated* polling method
    - *Collective communications* directly managed by the *scheduler*
  - Low memory footprint
    - Merge network buffer between MPI tasks [EuroPVM/MPI 09]
    - Dynamically adapt memory footprint (on going)
  - Deal with unbalanced workload
    - Collaborative polling (CP) [EuroMPI 12]



Message progression in MPI

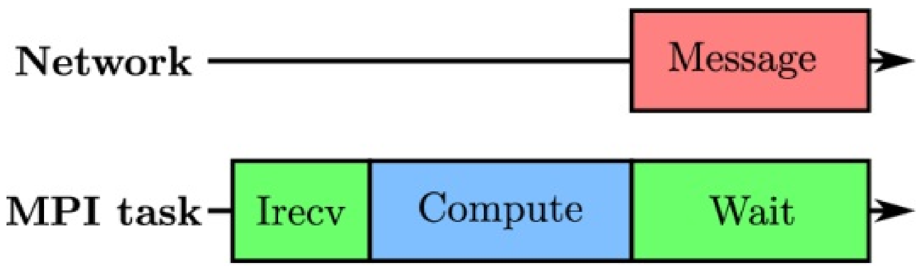
Progression-Threads: overheads

- Reactivity of the scheduler: how much time is required to switch to the progression thread?
- Length of a Time-Slice: is one TS enough to retrieve the message?
- One solution would to use Real-Time threads [HOEFLER08].

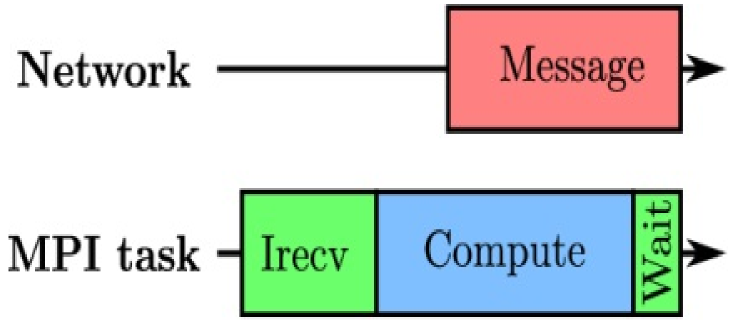


Collaborative Polling: Overview (1/3)

- MPI provides non-blocking calls for point-to-point communications
  - Ability to hide communication latencies with computation



(a) Without overlapping

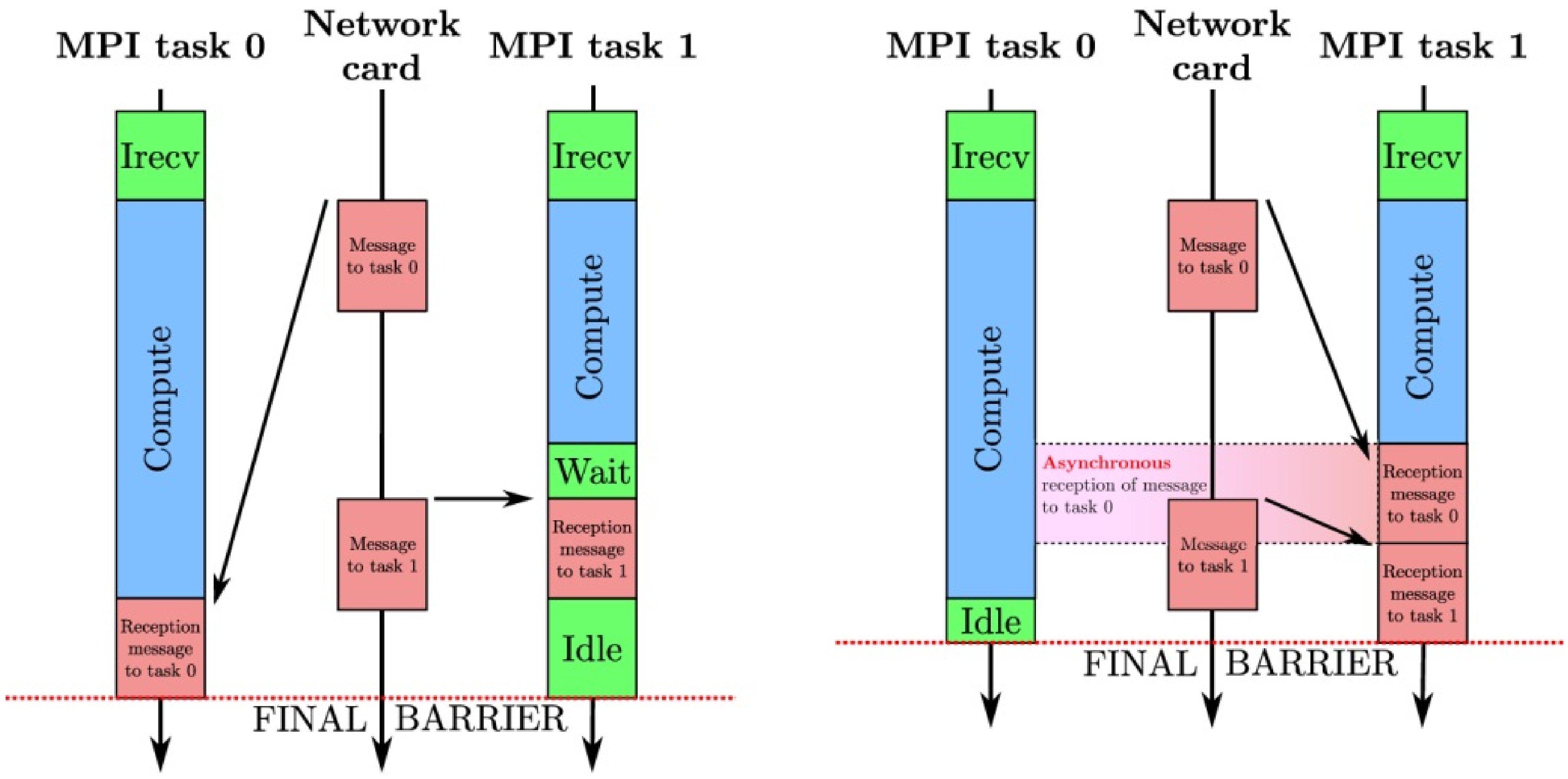


(b) With overlapping

Collaborative Polling: Overview (2/3)

- Common MPI implementations do not provide an efficient support of asynchronous MPI calls.
  - Messages only progressed when an MPI function is called.
  - Issue with long computation loops with no call to MPI (e.g., BLAS, I/O, …)
  - Possibility to enable a progression thread (Open MPI, MVAPICH2) for true asynchronous support.
- But an additional thread may harm code performance in some cases (e.g., low communication/computation ratio)
- Development of *Collaborative Polling* in MPC to benefit from asynchronous communications

Collaborative Polling: Overview (3/3)



(a) Without Collaborative Polling

(b) With Collaborative Polling

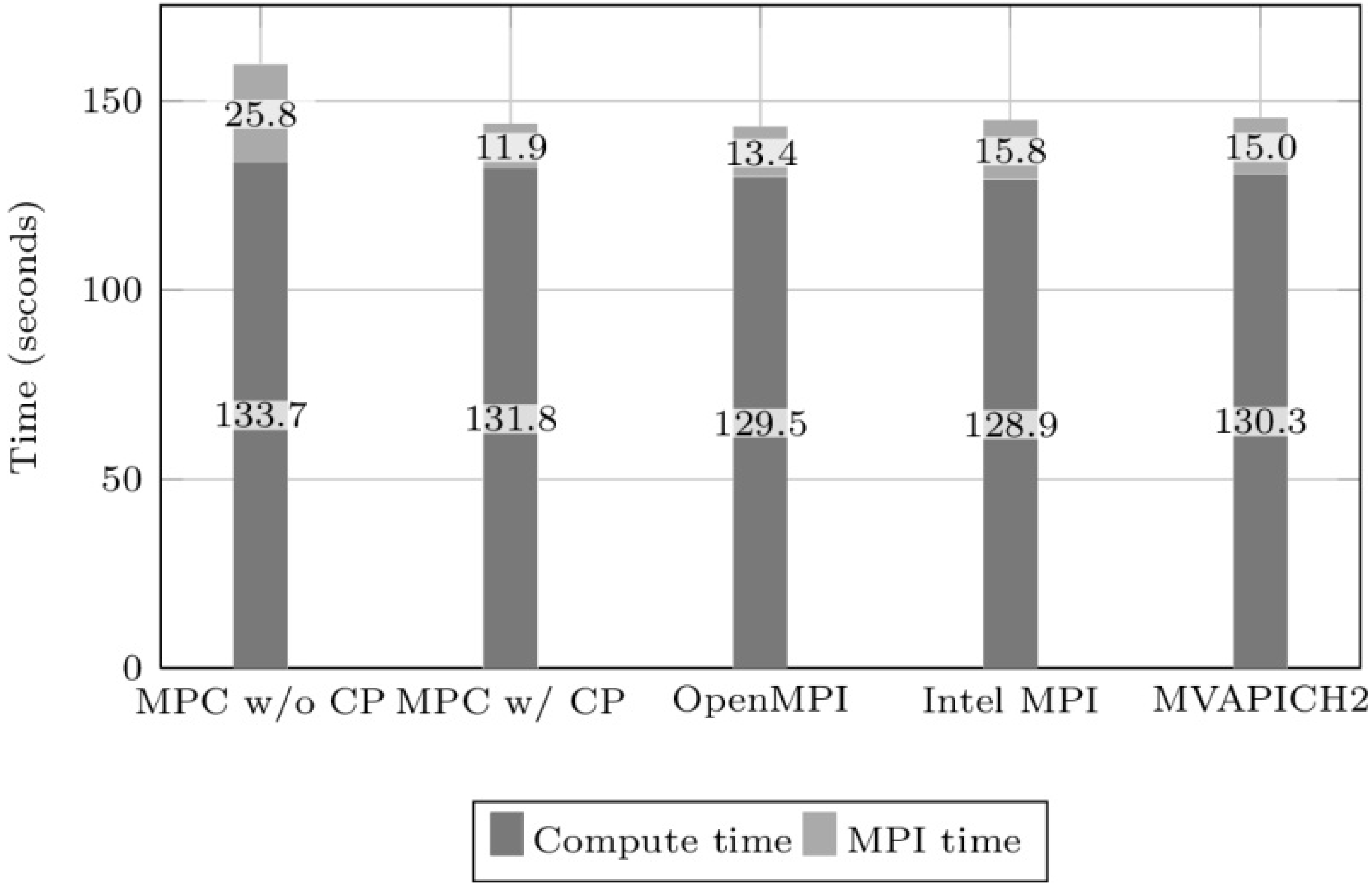
Collaborative Polling: Experimental Results

- Experiments on Curie cluster (PRACE)
  - 4-socket Nehalem EX @ 2.27Ghz (32 cores)
  - Mellanox Infiniband QDR

- Comparison of time spent in MPI libraries
  - MPC w/o Collaborative Polling (CP)
  - MPC w/ CP
  - Open MPI
  - Intel MPI
  - MVAPICH2

- Applications
  - EulerMHD: MPI C++
  - Gadget-2: MPI C

Collaborative Polling: EulerMHD on 1024 cores (1/2)

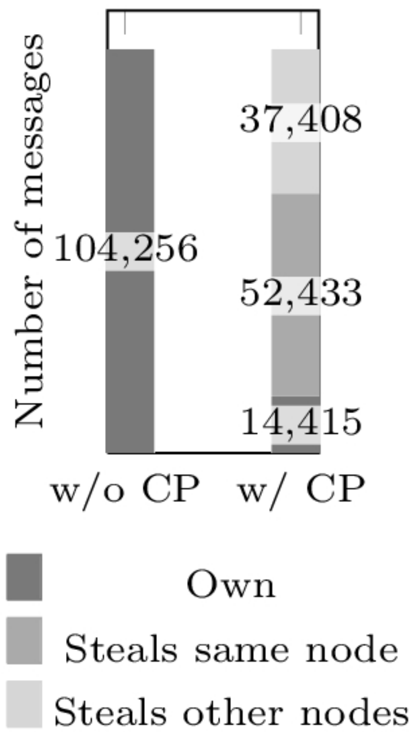


Collaborative Polling: EulerMHD on 1024 cores (2/2)

Function	w/o CP	w/ CP	Speedup
Execution time	159.41	143.74	1.11
Compute time	133.66	131.8	1.01
MPI time	25.76	11.94	2.16
MPI_Allreduce	3.12	2.75	1.13
MPI_Wait	21.86	8.45	2.59
MPI_Isend	0.57	0.49	1.16
MPI_Irecv	0.21	0.24	0.87

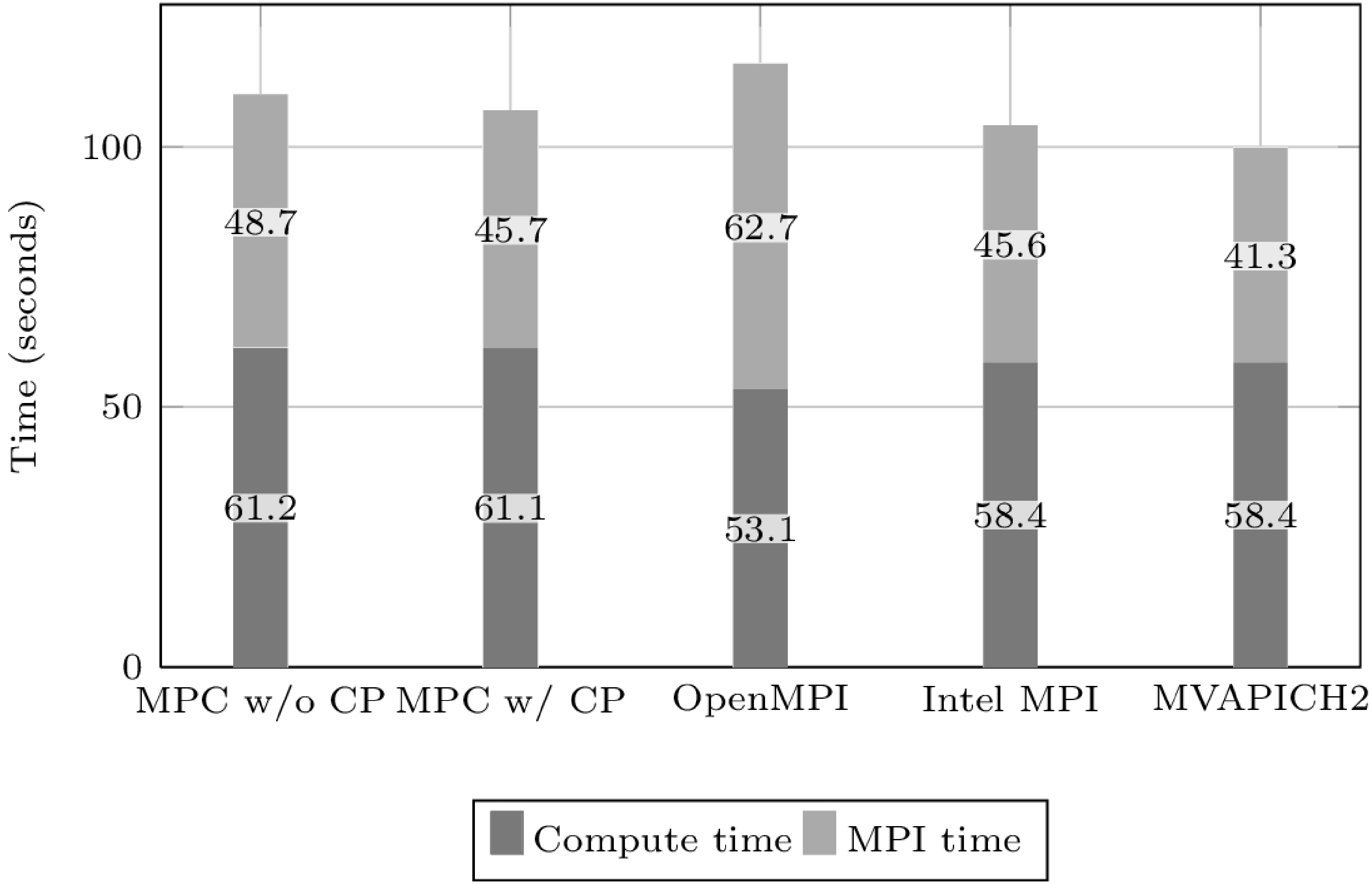
(a) Speedup with Collaborative Polling

- MPI time decreased by a factor of 2!
- 11 % improvement in execution time



(b) Collaborative Polling statistics

Collaborative Polling: Gadget-2 on 256 cores (1/2)

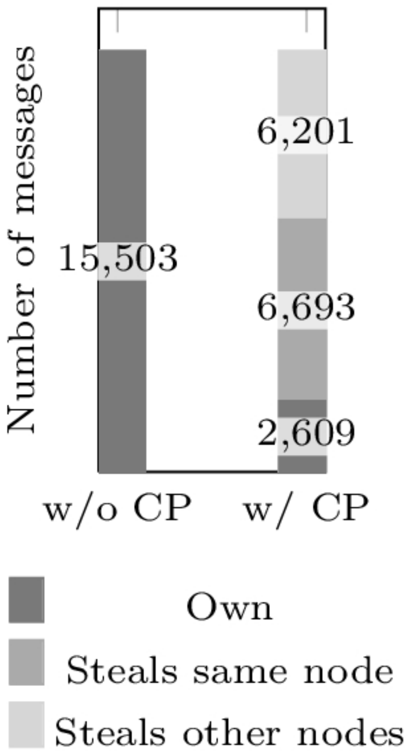


Collaborative Polling: Gadget-2 on 256 cores (2/2)

Function	w/o CP	w/ CP	Speedup
Execution time	109.87	106.8	1.03
Compute time	61.18	61.09	1
MPI time	48.69	45.7	1.07
MPI_Reduce	1.03	0.83	1.25
MPI_Allreduce	3.81	4.24	0.9
MPI_Recv	2.62	1.31	2
MPI_Barrier	6.55	6.56	1
MPI_Bcast	0.32	0.25	1.26
MPI_Allgather	9.07	9.22	0.98
MPI_Sendrecv	6.25	5.06	1.24
MPI_Gather	$4.62 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$	1.21
MPI_Ssend	0.18	0.18	0.99
MPI_Allgatherv	18.85	18.05	1.04

(a) Speedup with Collaborative Polling

7% improvement in MPI



(b) Collaborative Polling statistics

OpenMP

OpenMP 2.5

- OpenMP 2.5-compliant runtime integrated to MPC
- Directive-lowering process done by patched GCC (C,C++,Fortran)
  - Generate calls to MPC ABI instead of GOMP (GCC OpenMP implementation)

Lightweight implementation

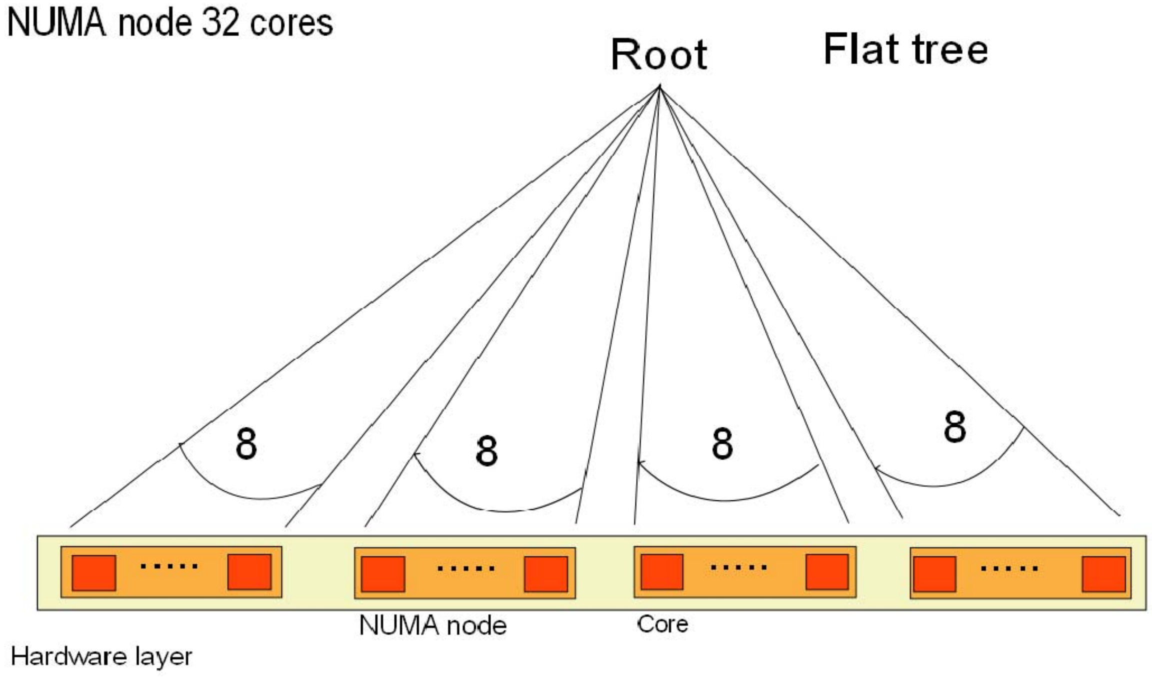
- *Stack-less* and *context-less* threads (*microthreads*)
- Dedicated scheduler (*microVP*)
  - On-the-fly stack creation
- Support of *oversubscribed* mode
  - *Many more* OpenMP threads than CPU cores

Hybrid optimizations

- *Unified* representation of *MPI tasks* and *OpenMP threads* [IWOMP 10]
- Scheduler-integrated Multi-level polling methods
- Message-buffer privatization
- Parallel message reception
- Large NUMA node optimization [IWOMP 12]

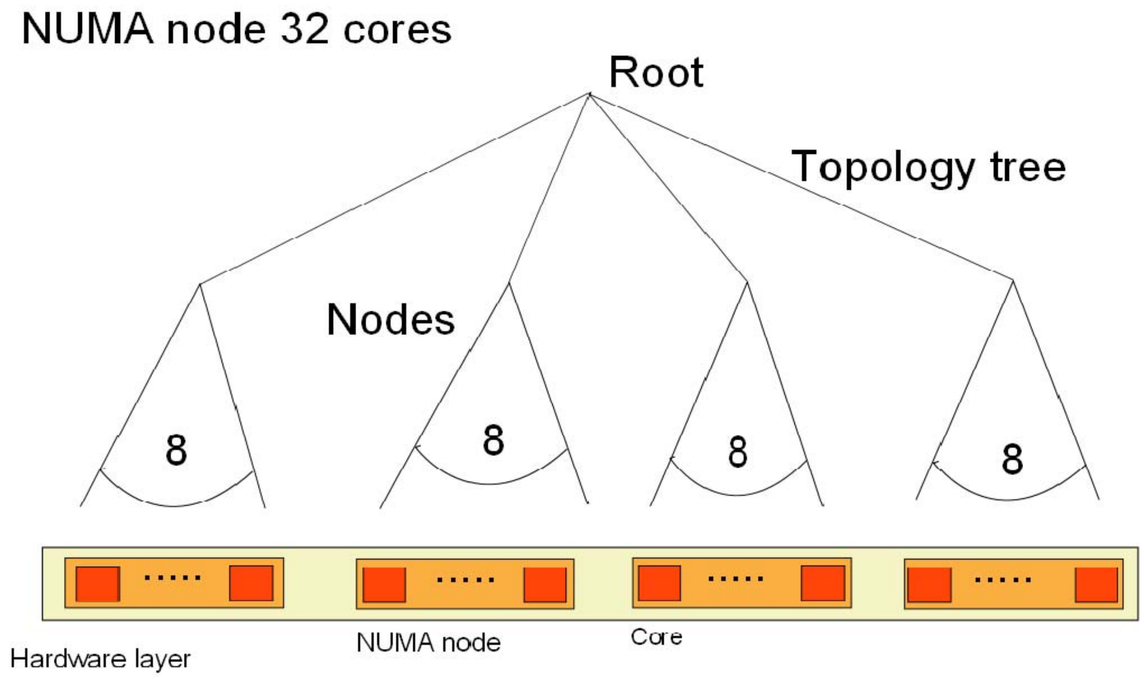
OpenMP Scalability: Tree on Mesca Node (1/2)

- Flat tree is the most simple structure to use
  - Fast to wake few threads
  - Large overhead to traverse many threads



OpenMP Scalability: Tree on Mesca Node (2/2)

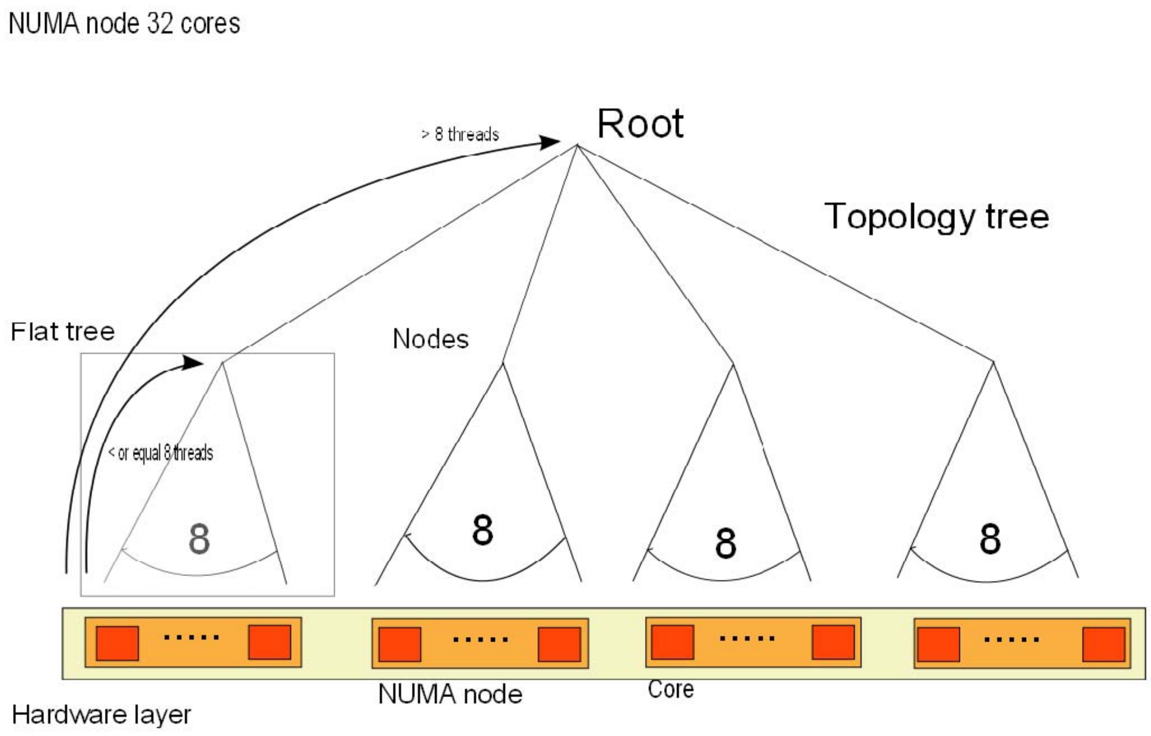
- Tree following the architecture topology
  - 4 NUMA nodes with 8 cores → "4-8" tree
  - More parallelism to wake large number of threads
  - Overhead for few threads (tree height)



OpenMP Scalability: Mixed Tree for Mesca Node

- Contribution

  - Exploit sub-trees inside the topology tree for efficient synchronization
  - Depending on the number of threads, use different sub-trees



OpenMP Scalability: Experimental Results

Experimental environment

- TERA 100 node (32 cores)
- Node w/ BCS (128 cores)

Benchmark

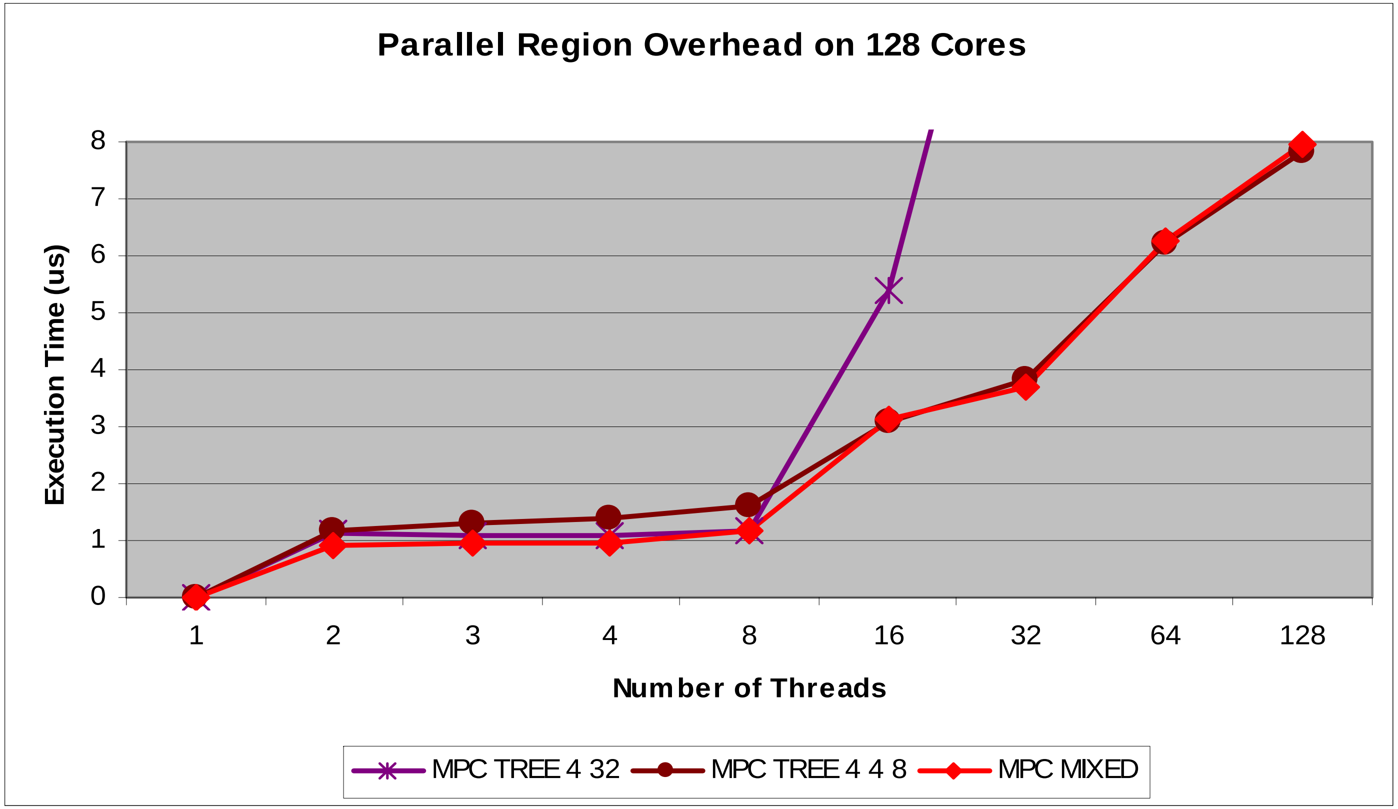
- EPCC microbenchmarks
- Measure overhead of OpenMP construct
- Focus on 2 constructs
  - `#pragma omp parallel`
  - `#pragma omp barrier`

Evaluation

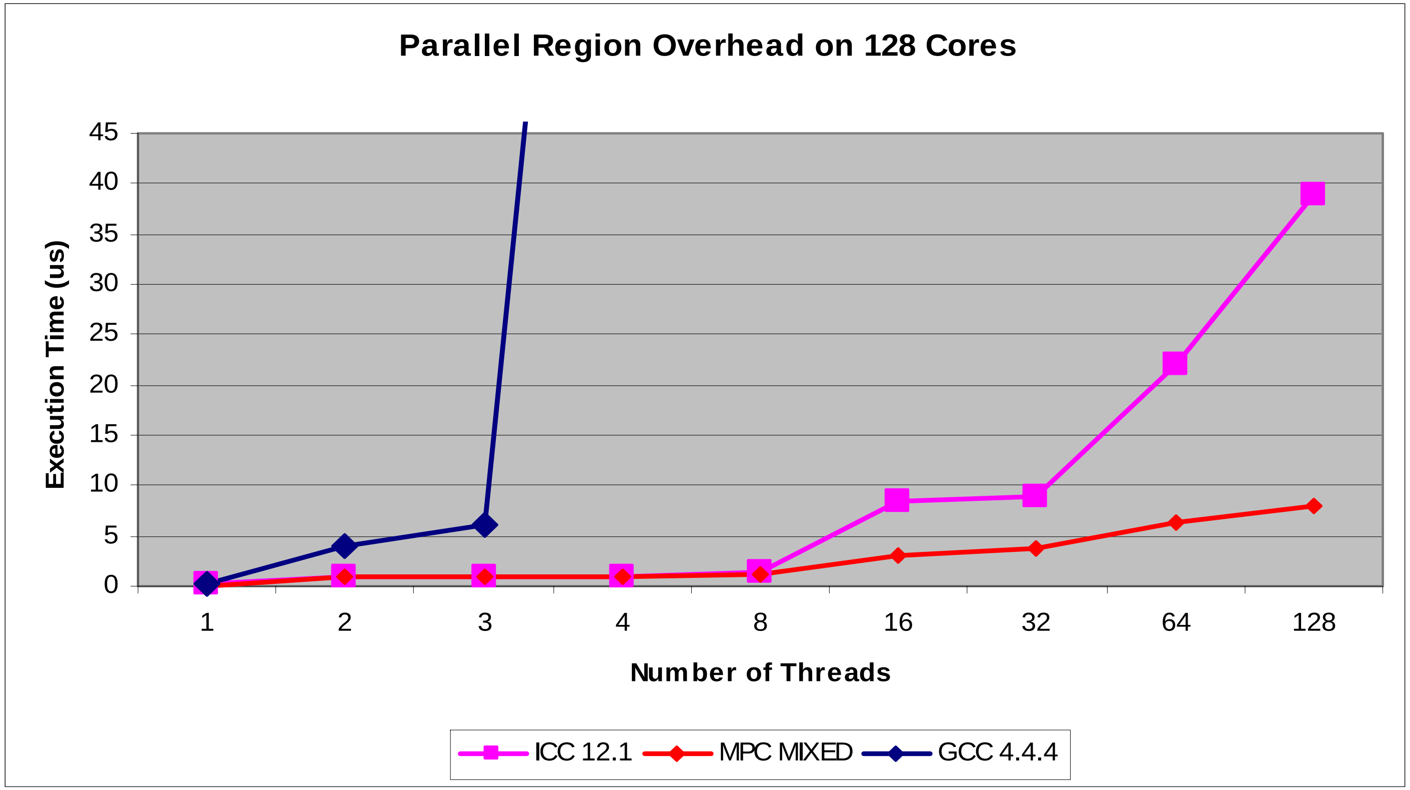
- MPC with multiple trees
- Intel ICC compiler (v. 12.1)
- GCC compiler (v. 4.4.4)

OpenMP Scalability: Parallel Construct (1/2)

- Mix tree with “4-32” and “4-4-8”
- Results in better performance of both trees



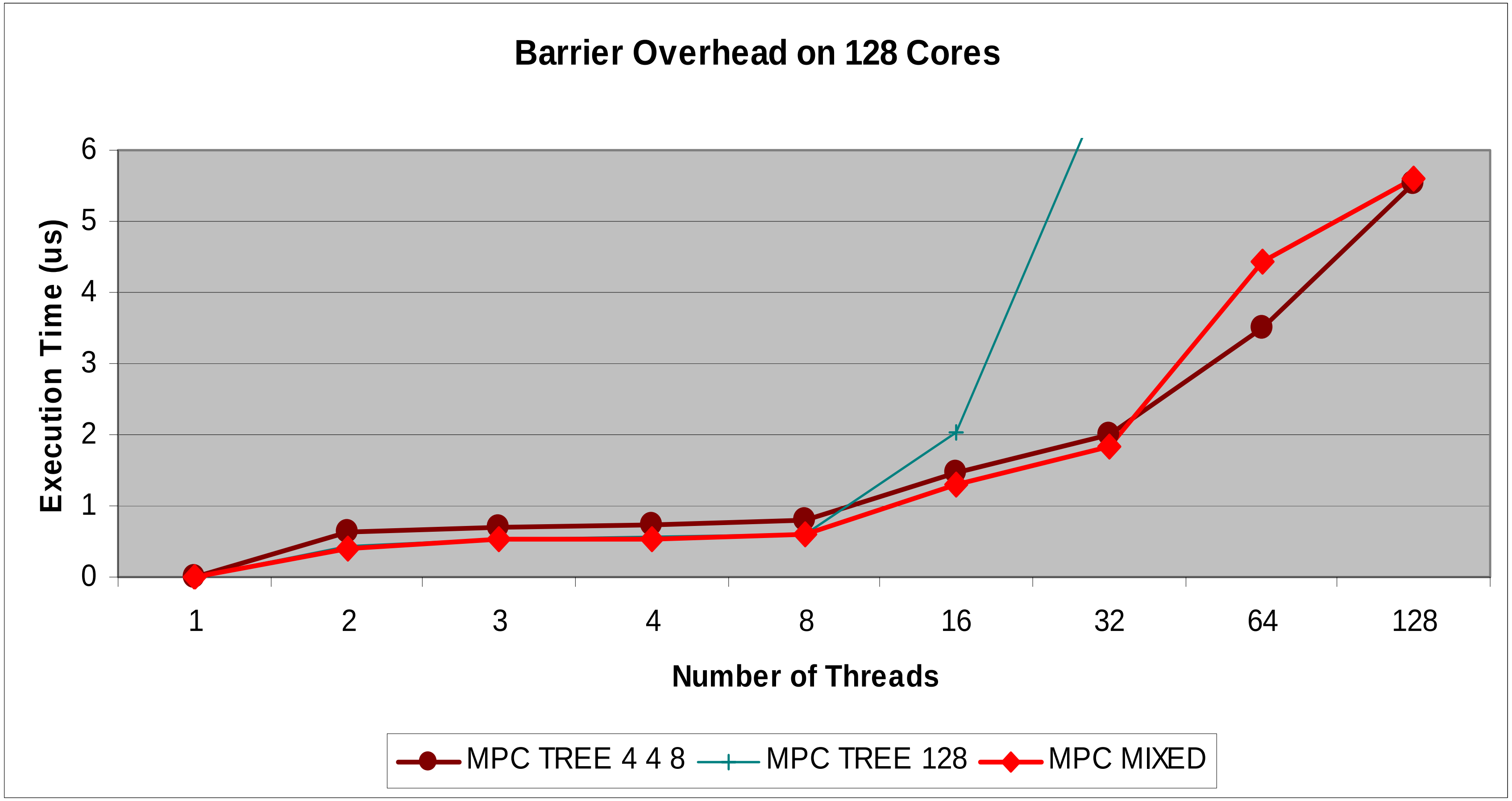
OpenMP Scalability: Parallel Construct (2/2)



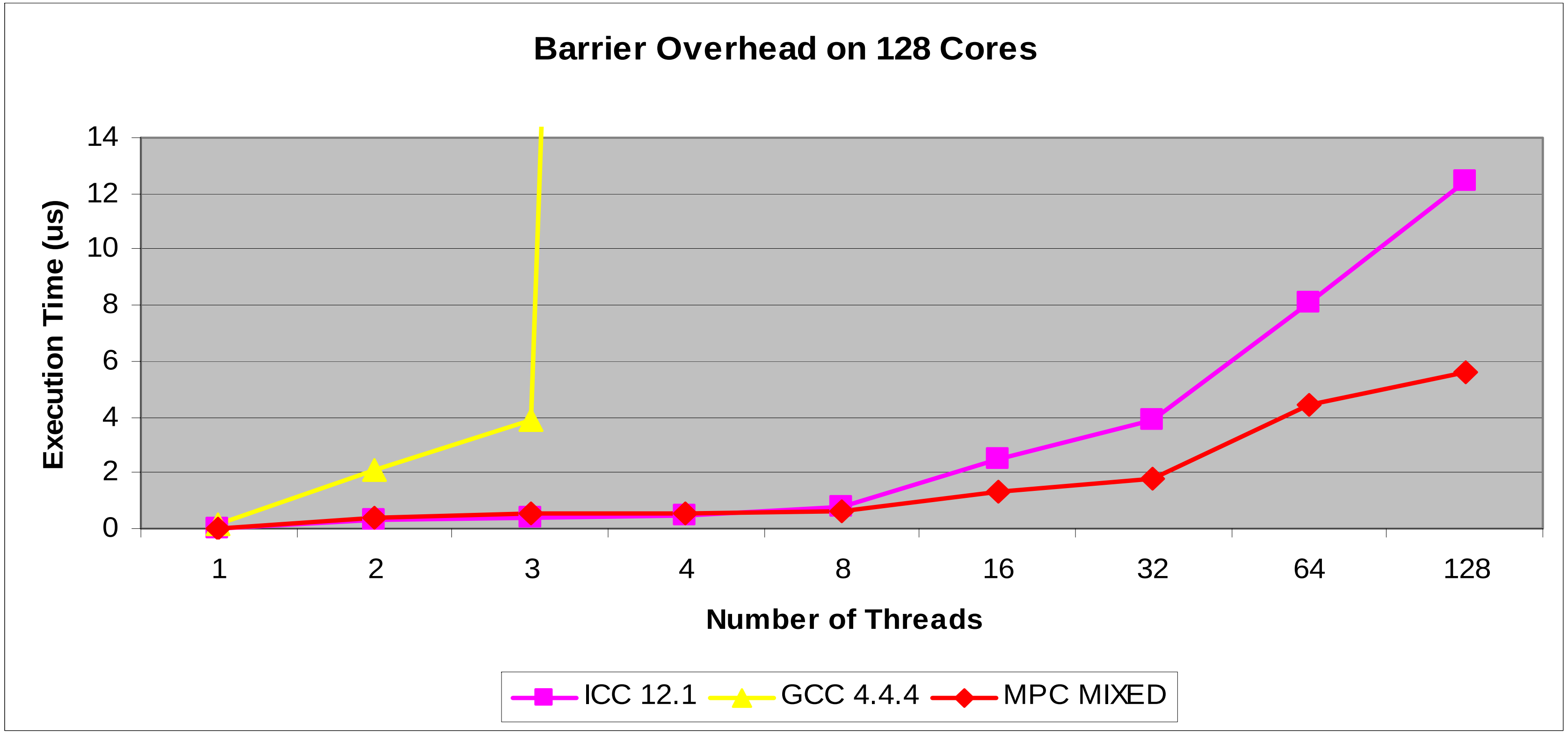
- Comparison of Intel ICC, GCC and MPC with Mixed Tree
  - Large overhead for GCC
  - Speed up of 4x for MPC compared to state-of-the-art ICC

OpenMP Scalability: Barrier Construct (1/2)

- Mix tree with “4-32” and “4-4-8”
- Results in better performance of both trees



OpenMP Scalability: Barrier Construct (2/2)



- Comparison of Intel ICC, GCC and MPC with Mixed Tree
  - Large overhead for GCC
  - Speed up of 2x for MPC compared to state-of-the-art ICC

PThreads

- Thread library completely in user space
  - Non-preemptive library
  - User-level threads on top of kernel threads (usually 1 per CPU core)
  - Automatic binding (kernel threads) + explicit migration (user threads)
  - MxN *O(1)* scheduler
    - Ability to map M threads on N cores (with M>>N)
    - Low complexity

- POSIX compatibility
  - POSIX-thread compliant
  - Expose whole PThread API

- Integration with other thread models:
  - *Intel's Thread Building Blocks* (TBB)
  - ➔ Small patches to remove busy waiting
  - Unified Parallel C (UPC)
  - Cilk

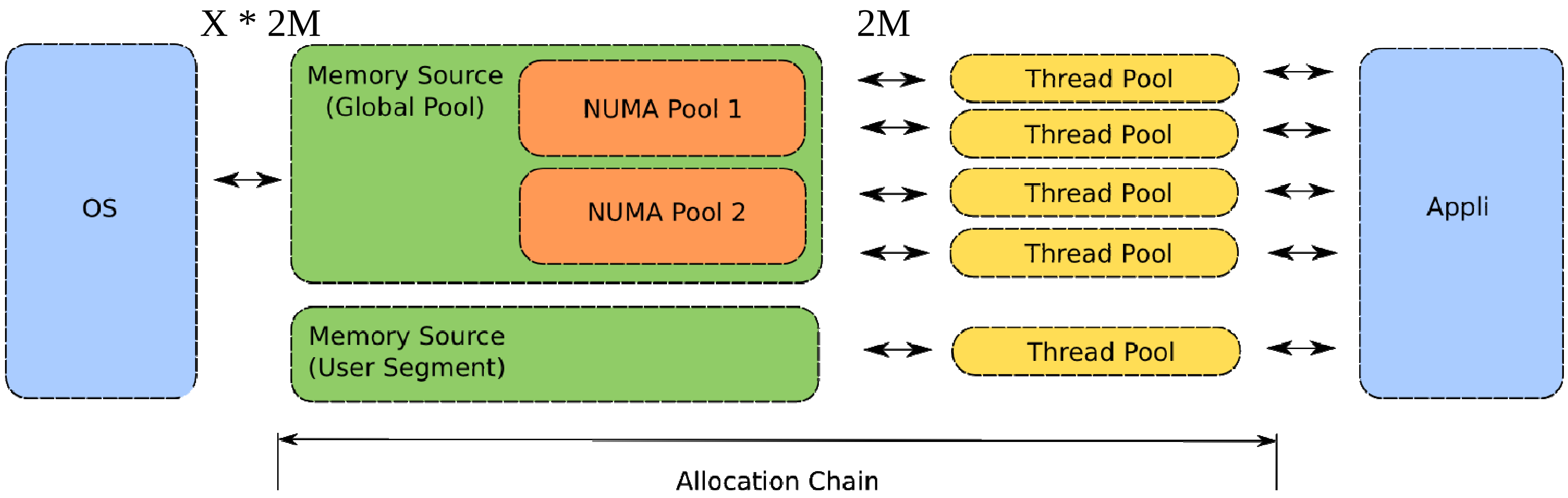
Memory allocation

- Optimize memory allocation in heavily multithreaded context
- Optimize memory alignment and reduce cache conflicts
  - Offset for large arrays
  - Contiguous physical memory allocation
- Optimize memory allocation on node with a large number of cores
  - Trade-of memory consumption/performances

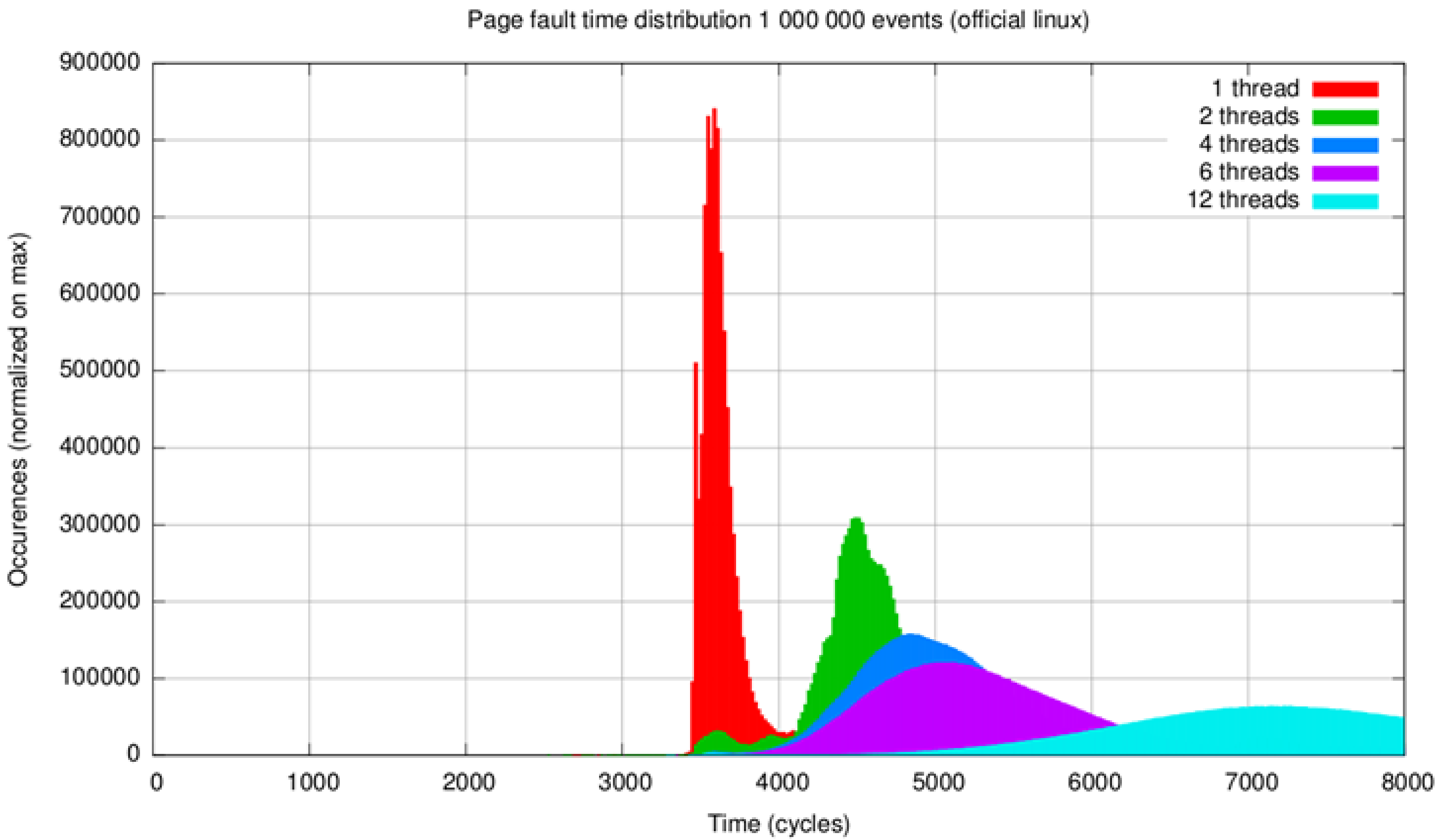
Alloc.	Tot. (s)	Sys.(s)	Mem. (GB)
Jemalloc	140.9	12.4	2.2
MPC v2.4.1	165.9	12.3	2.7
MPC v2.2.0	153.6	4.5	4.4
Glibc	147.4	4.2	3.7
Tcmalloc	137.6	2.1	3.7
Hoard	492.7	182.1	2.8

Memory Allocator: General Design

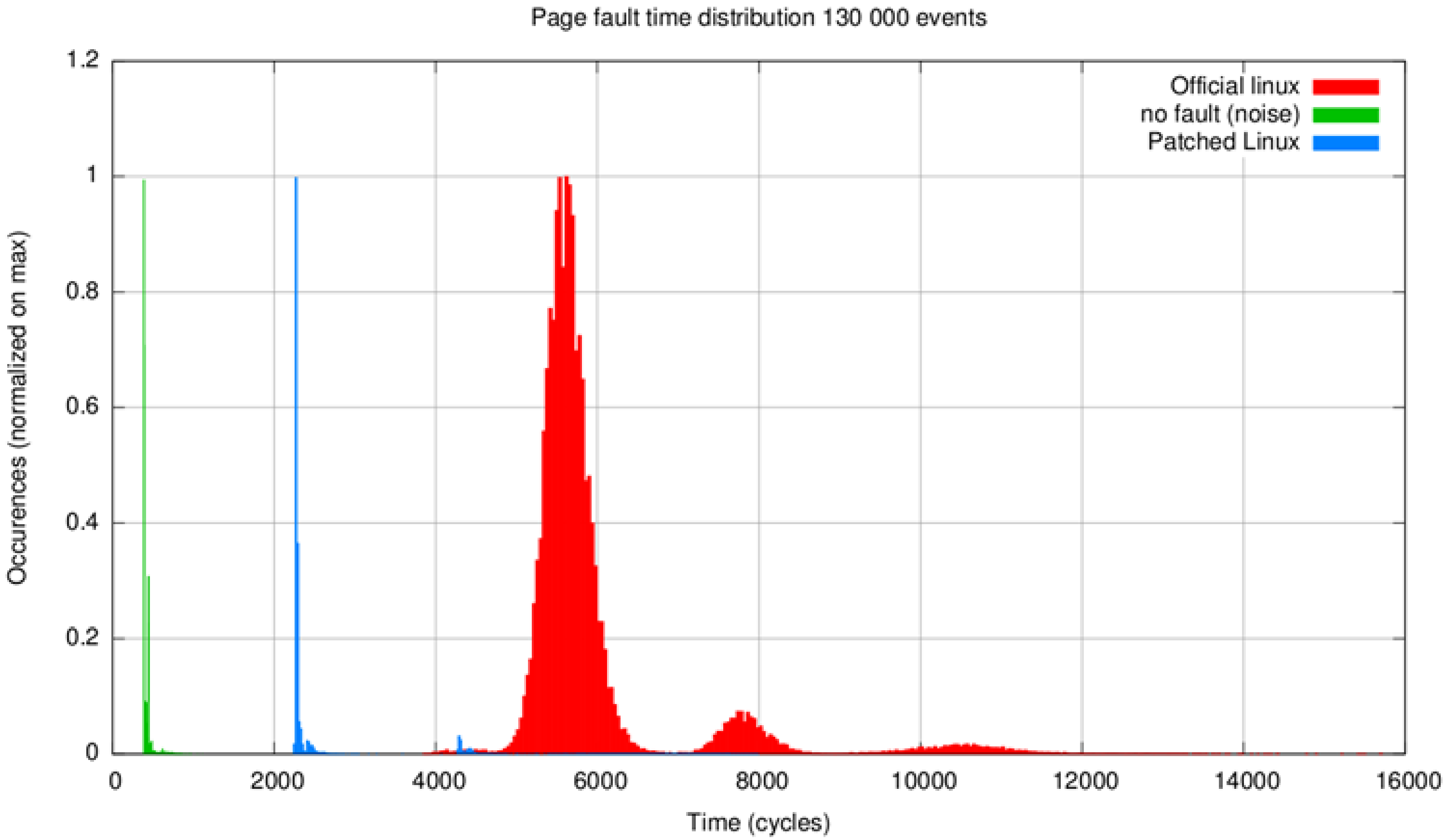
- Define an allocation chain :
  - A « Thread Pool » to manage non used chunks.
  - A memory source
- An allocation chain per thread.
- Exchange by macros-blocs of 2M.



Memory for Manycore Architectures



- One example of memory optimization: lazy “zero-page”



MPC vs MPI with HERA: TERA-100 results

- MPC Multithreading: 1 process per node, 1 thread per core (32 threads)
- 35 million AMR cells, 3 AMR levels (3x3), multi-material 2D hydro

Core count	MPC Multithread + InfiniBand (total time + grind time)		OpenMPI (total time + grind time)		**OpenMPI overhead**
1024	**254 s**	8.88 µs	**371 s**	12.63 µs	**+46%**
2048	**184 s**	11.16 µs	**426 s**	22.12 µs	**+131%**

- **1024 cores**: small number of cells per core (~35k), OpenMPI is *much slower* than MPC (46%)
- **2048 cores**: even smaller number of cells per core (~17k), a gain is still observed with MPC thanks to non-blocking *multithreading, a* slow down appears with OpenMPI.
- Very satisfactory *multithreading* results for *future many-core hardware* with *very small memory* per core (ex: Intel Xeon Phi, …)

DEMO 1

Programming Models

Programming Models

Provide a way to move to “Exascale” programming models

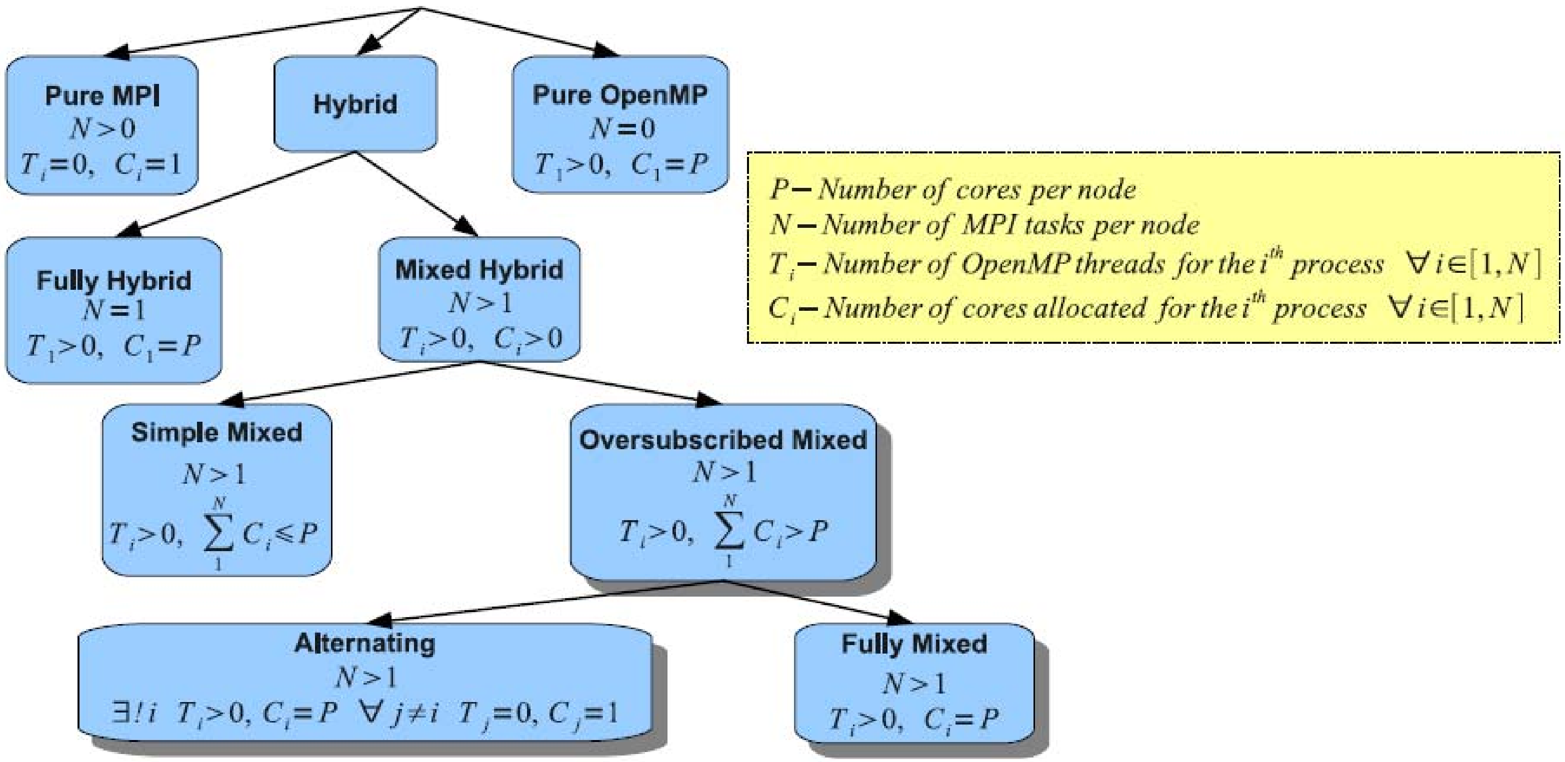
- Starting point: MPI everywhere
- Provide a way to add threads within MPI applications without breaking everything
  - MPI + OpenMP taxonomy and optimizations
  - Extended Thread Local Storage (Extended TLS)
- Provide methods to reduce data replication in MPI
  - Hierarchical Local Storage (HLS)
- Provide methods to exploit dedicated hardware (aka. accelerators) in current applications
  - Incremental method

Emerging models evaluation

- How to integrate multiple runtimes (PGAS + X, MPI + X)

MPI + OpenMP

- Deal with current applications and prepare future
  - Start for an MPI code
  - Smoothly move to MPI + X
  - Require good integration to keep performance on actual hardware
  - Prepare next generation of numerical schemes on current programming models
  - Taxonomy of possible ways to mix MPI and OpenMP:



Extended TLS [IWOMP 11]

- Cooperation between compiler and runtime system
  - Compiler part in GCC
  - Runtime part in MPC (Message-Passing Computing)
  - Linker optimization
- Compiler part (GCC)
  - New middle-end pass to place variables to the right extended-TLS level
  - Modification of backend part for code generation (link to the runtime system)
- Runtime part (MPC)
  - Integrated to user-level thread mechanism
  - Copy-on-write optimization
  - Modified context switch to update pointer to extended TLS variables
- Linker optimization (GLIBC)
  - Support all TLS modes
  - Allow Extended TLS usage without overhead

Hierarchical Local Storage (HLS) [IPDPS 12]

Context

- Allow the possibility to share data among MPI tasks located on the same node
- Target common variables (mainly read, barely written)
- E.g., physics constants

Goal

- Directive-based design and implementation for C, C++ and Fortran
- Compiler part in GCC, runtime part in MPC, optimization part in linker

Current status

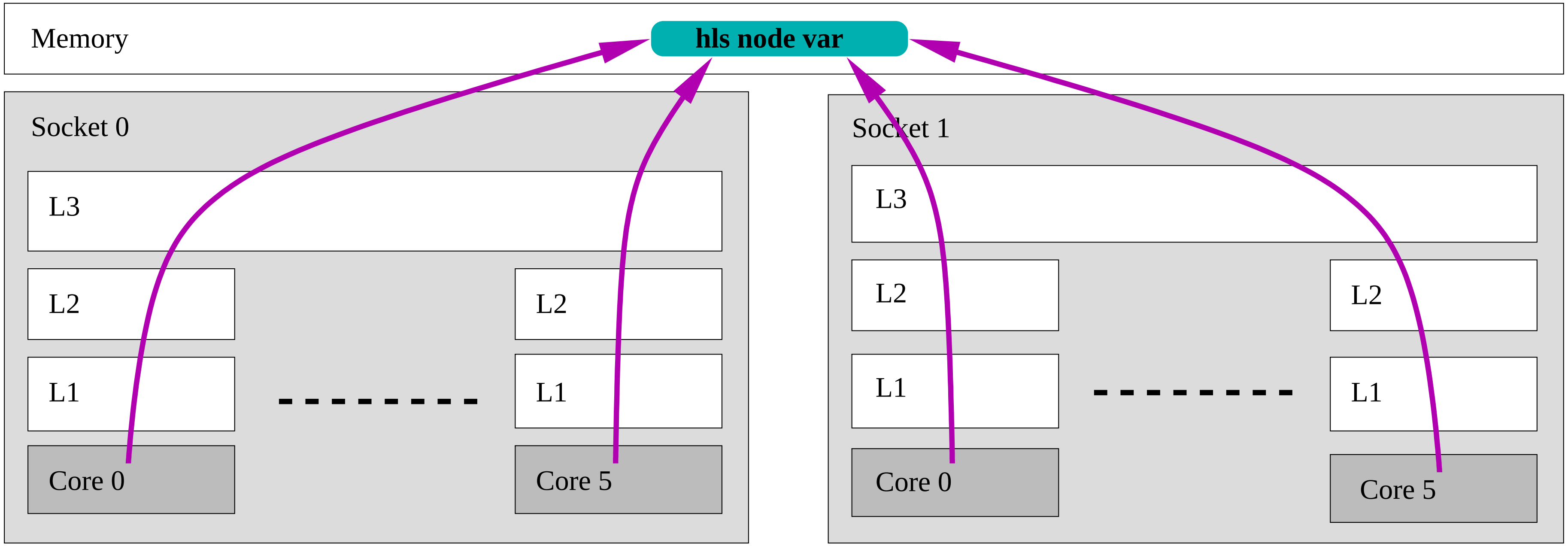
- Available since MPC 2.3.0
- Directive specification

  #pragma hls scope(variable1, …)

  #pragma hls single(variable1, …) [nowait]
- Complete implementation in GCC, Binutils and MPC
- Application porting: easy on known applications

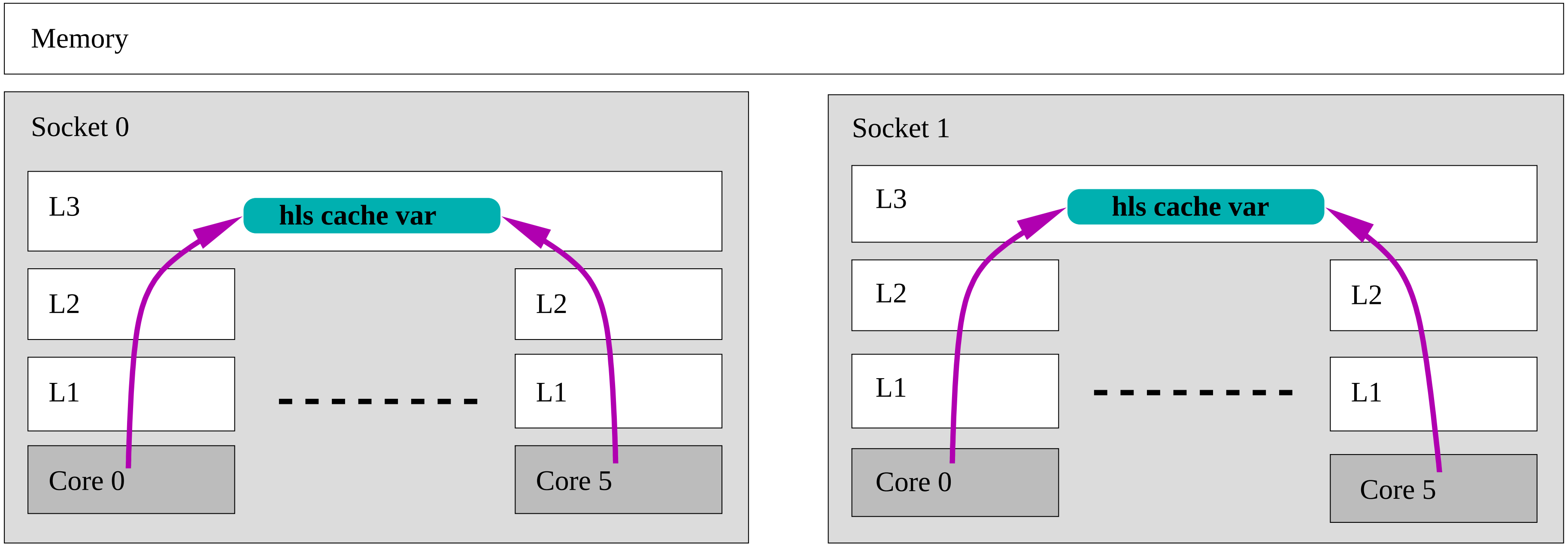
Example of HLS

- Example of one global variable named `a`
  - Duplicated in standard MPI environment
  - May be shared to save memory with directive
    - `#pragma hls node(a)`



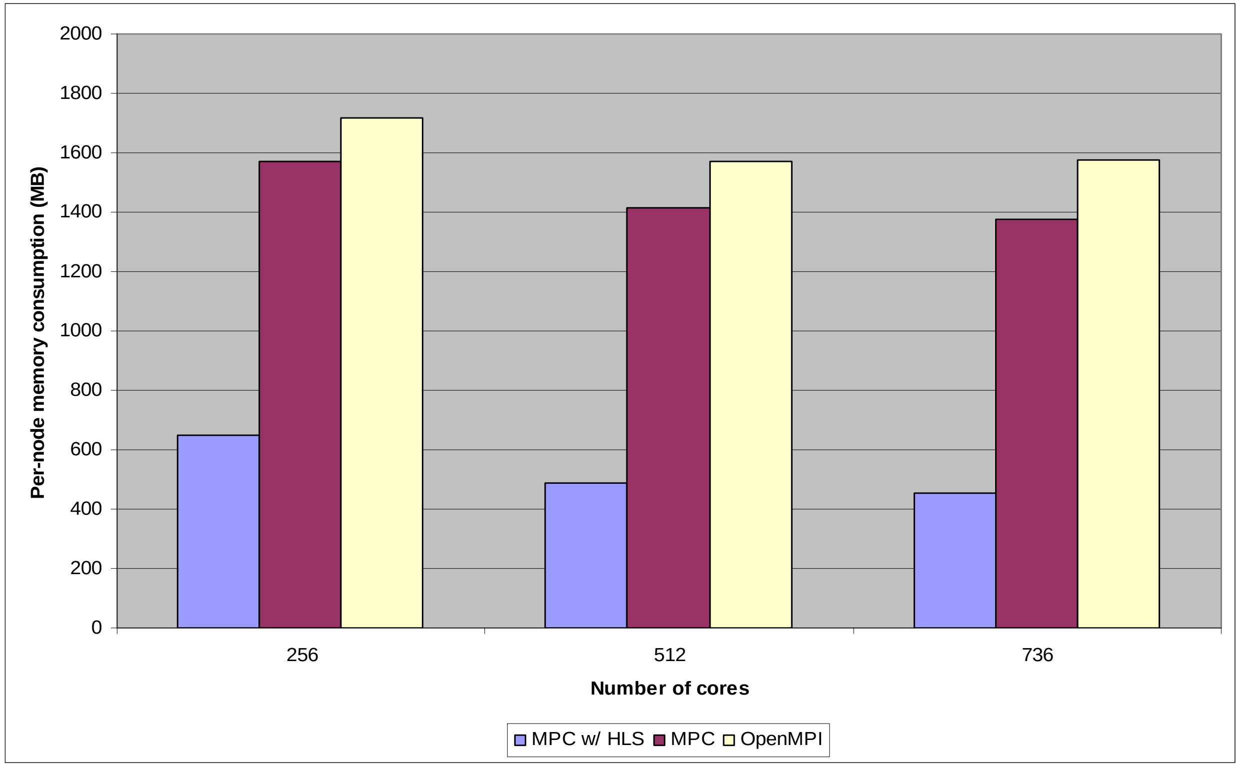
Example of HLS

- Multiple level available
  - Example of cache level 3
    - `#pragma hls cache(a) level(3)`



HLS Experiments

- EulerMHD 4096x4096 with **128 MB** of physics constants per MPI task
  - Up to 3.5 less memory consumed than OpenMPI
  - On 2-socket 4-core Core2Quad

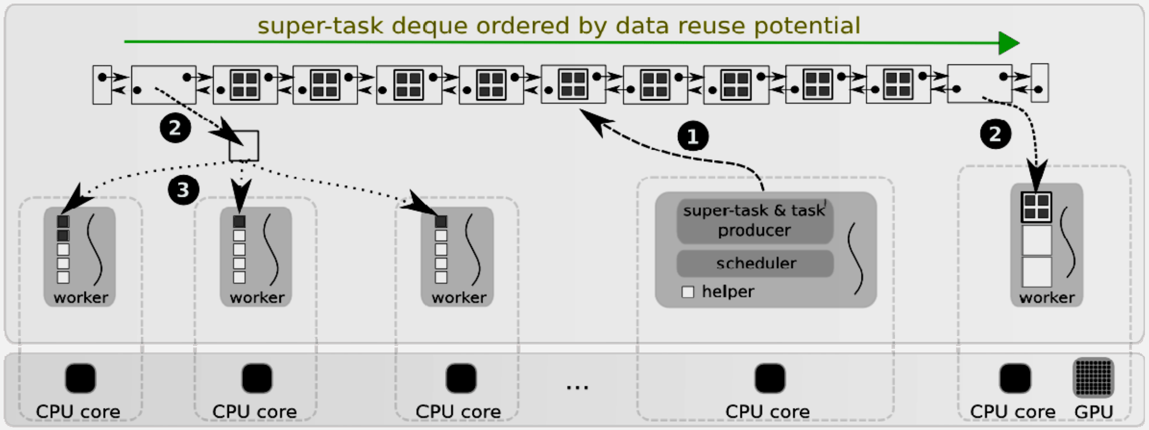


DEMO 2

Heterogeneous Task Scheduler

- Goal: Harness at the same time CPUs and accelerators in the context of irregular numerical computations

- Balance workload between each architecture by introducing a two-level work stealing mechanism:

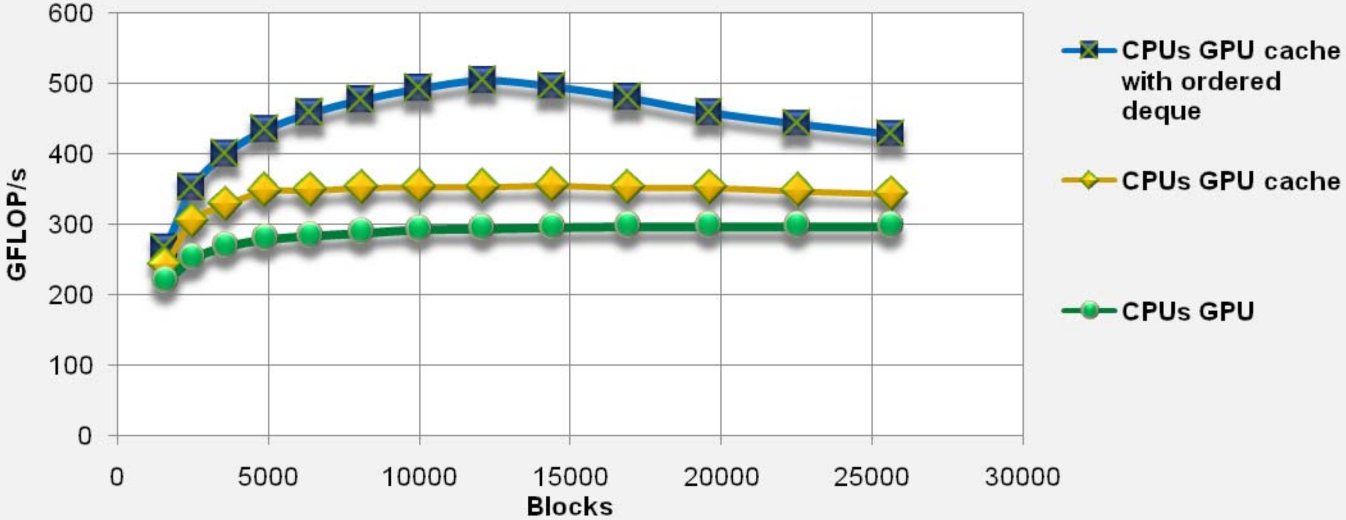


Hierarchical scheduling scheme

- Improve locality with a software cache strongly coupled to the scheduler
  - Designed to reduce memory transfers by retaining data in off-chip memory
  - Scheduler guided by cache affinity to avoid unnecessary transfers

Heterogeneous Task Scheduler

LU Decomposition with Dense & Sparse Blocks,
cumulated perf. of step 3** (SGEMM –> MKL & CUBLAS) **vs Matrix size



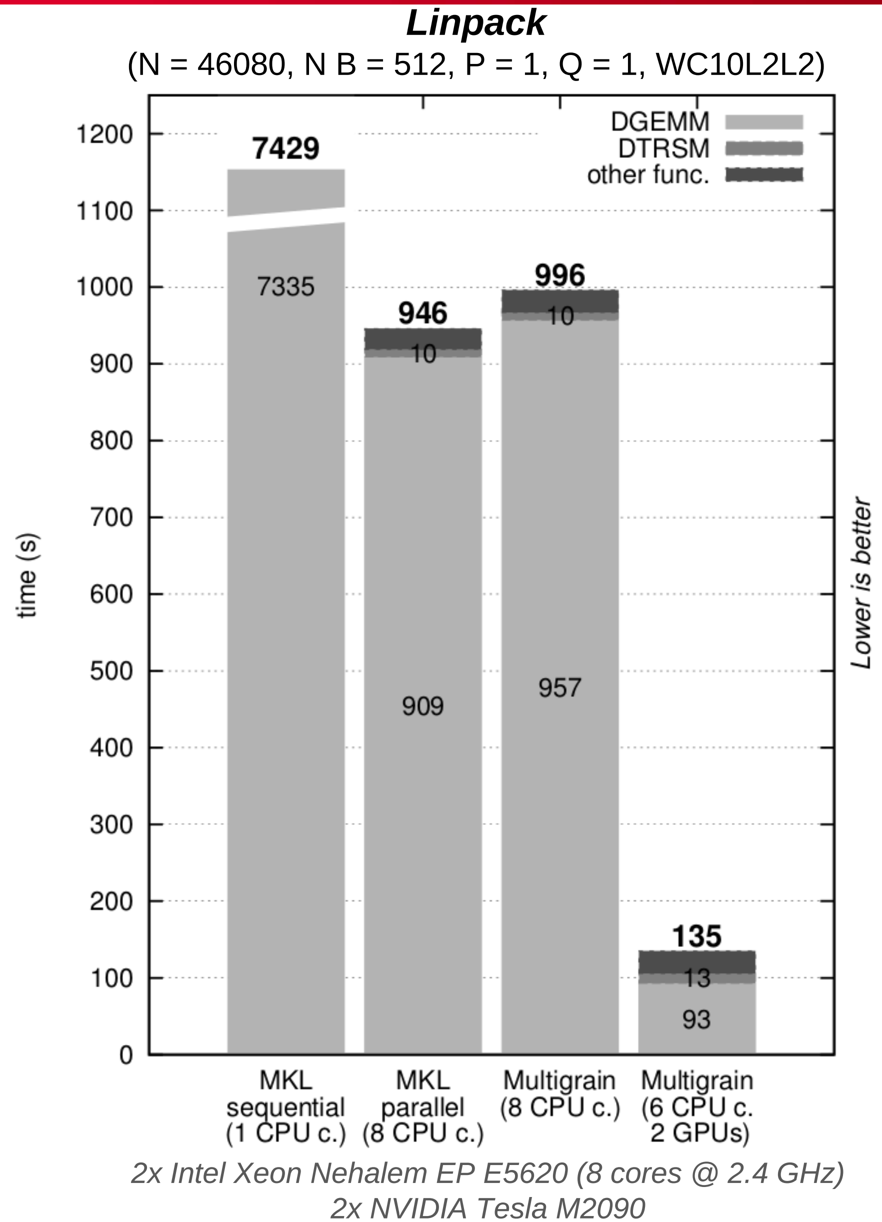
2x AMD 6164HE (24 cores @ 1.7 GHz)
1x Nvidia Geforce GTX 470 (448 cores @ 1.215GHz)

> Paper presented at MULTIPROG (January 2012)

Jean-Yves Vet, Patrick Carribault, Albert Cohen, **Multigrain Affinity for Heterogeneous Work Stealing**, MULTIPROG ‘12

> Could be used to exploit several types of many-core processors (Nvidia GPUs, AMD GPUs/APUs, Intel MIC, …)

Heterogeneous Task Scheduler



Heterogeneous BLAS Library

- Based on Intel MKL and NVIDIA CUBLAS optimised kernels
- Transparent for users
- Internal decomposition into super-tasks and tasks
- LINPACK: Homogeneous performance close to parallel MKL (**62.11** vs **68.94 GFLOP/s**)
- LINPACK: Heterogeneous performance reaches **482.4 GFLOP/s**

DE LA RECHERCHE À L'INDUSTRIE

Heterogeneous Task Scheduler

PN

Numerical flux

/* Part 1: Large matrix multiplications 1 */
GEMM [in: $A_{X(136\times136)}$, $B_{X(1M\times136)}$] [out: $C_{X(1M\times136)}$]
GEMM [in: $A_{Z(136\times136)}$, $B_{Z(1M\times136)}$] [out: $C_{Z(1M\times136)}$]

/* Part 2: Small matrix multiplications */
GEMM [in: $A_{X(136\times136)}$, $BL_{X(1K\times136)}$] [out: $CL_{X(1K\times136)}$]
GEMM [in: $A_{Z(136\times136)}$, $BL_{Z(1K\times136)}$] [out: $CL_{Z(1K\times136)}$]
GEMM [in: $A_{X(136\times136)}$, $BR_{X(1K\times136)}$] [out: $CR_{X(1K\times136)}$]
GEMM [in: $A_{Z(136\times136)}$, $BR_{Z(1K\times136)}$] [out: $CR_{Z(1K\times136)}$]
BARRIER

/* Part 3: Tasks */
TASK [in: $D_{X(1024)}$, $C_{X(1M\times136)}$] [out: $E_{X(1M\times136)}$]
TASK [in: $D_{Z(1024)}$, $C_{Z(1M\times136)}$] [out: $E_{Z(1M\times136)}$]
BARRIER
TASK [in: $D_{X(1024)}$, $CL_{X(1K\times136)}$] [in-out: $E_{X(1M\times136)}$]
TASK [in: $D_{Z(1024)}$, $CL_{Z(1K\times136)}$] [in-out: $E_{Z(1M\times136)}$]
BARRIER
TASK [in: $D_{X(1024)}$, $CR_{X(1K\times136)}$] [in-out: $E_{X(1M\times136)}$]
TASK [in: $D_{Z(1024)}$, $CR_{Z(1K\times136)}$] [in-out: $E_{Z(1M\times136)}$]
BARRIER

/* Part 4: Large matrix multiplications 2 */
GEMM [in: $F_{X(136\times136)}$, $E_{X(1M\times136)}$] [out: $F_{X(1M\times136)}$]
GEMM [in: $F_{Z(136\times136)}$, $E_{Z(1M\times136)}$] [out: $F_{Z(1M\times136)}$]
BARRIER

numerical_flux function

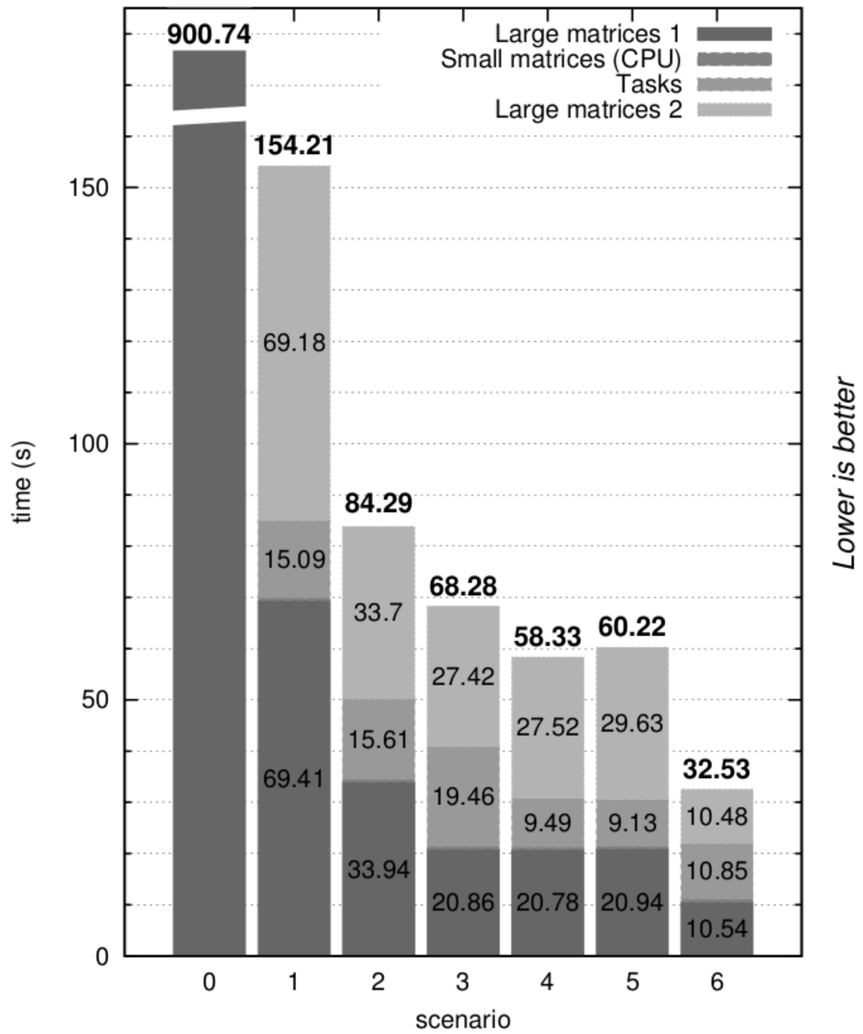
1. two large matrix multi-plications,
2. four independent small matrix multiplications
3. several tasks taking as input data generated by previous operations
4. two other large matrix multiplications exploiting data generated by the preceding step

How to avoid coslty data transfers ?

Heterogeneous Task Scheduler

PN

(N = 15, Z=X=1536)



2x Intel Xeon Nehalem EP E5620 (8 cores @ 2.4 GHz)
2x NVIDIA Tesla M2090

Data centric scheduling scheme

Scenarios

0: sequential (CPU)
1: homogeneous (CPUs)
2: heterogeneous
small tasks on CPUs only
3: heterogeneous
small tasks (**perf centric mode**)
4-5: heterogeneous
small tasks (**data centric mode**)
6: heterogeneous w/o data transfer

→ Program clearly limited by data transfers (via PCIe)

→ Reasoning on data locality for some tasks, and hampering transfers for load balancing gives additional performance

Language evaluations

- UPC
  - Berkeley UPC on the top of MPC
- Cilk
  - Cilk on the top of MPC
  - Evaluation of mix MPI + OpenMP + Cilk
- OpenACC
  - Evaluation of an OpenACC implementation (compiler part in GCC with CUDA backend)
- OpenCL
  - Evaluation of language capabilities
- OpenMP tasks
  - Prototype a task engine
  - How to mix multiple task models?

Tools: Debug/Profiling

Tools: Debug/Profiling

Debugging tools

- User-level thread debugger
- Help the conception and the maintainability of MPI + X applications
- Provide tools to solve bugs occurring during nights and week-ends on large number of cores
- Static/dynamic communications schemes checking

Profiling tools

- Tools adapted to MPC
- Tools for very large executions

Compiler support

- Help the programmer to move from MPI-everywhere to MPI + X
- Integration of our solutions in production compiler
- Dynamic analysis for potential HLS

Debugging

- Static analysis
  - Use GCC compiler to analyze
    - MPI, OpenMP, MPI + OpenMP
    - Detect wrong usage of MPI (collective communications with control flow)
- Dynamic (based on traces)
  - Use traces to debug large scale applications
  - Crash-tolerant trace engine
  - Parallel trace analyzer
- User level thread debugging [MTAAP 10]
  - Provide a generic framework to debug user-level thread
    - Evaluated on MPC, Marcel, GNUPth
  - Provide a patched version of GDB
  - Collaboration with Allinea DDT
    - MPC support in Allinea DDT

Profiling

- Application profiling
  - Unable to reduce the test case due to network topology impact on performance
  - Unable to store very large traces
    - Huge impact on the execution
    - Stress up the file system
  - In situ analysis

- Collaboration with other profiling tools
  - TAU is now MPC compliant
    - Thanks to Extended TLS

Compiler support

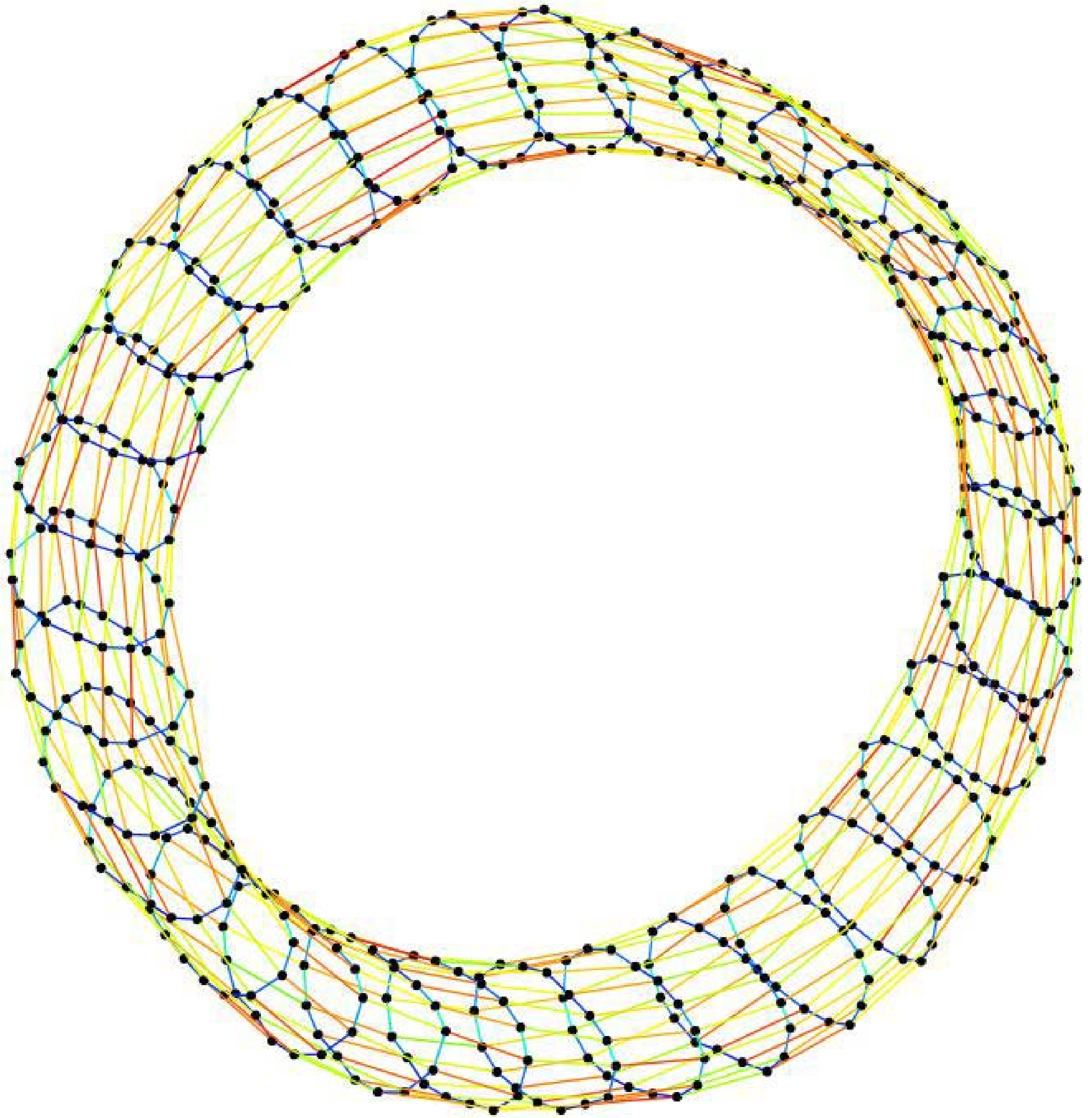
- ## Global variables
  - Expected behavior: duplicated for each MPI task
  - Issue with thread-based MPI: global variables shared by MPI tasks located on the same node

- ## Solution: **Automatic privatization**
  - Automatically convert any MPI code for thread-based MPI compliance
  - Duplicate each global variable

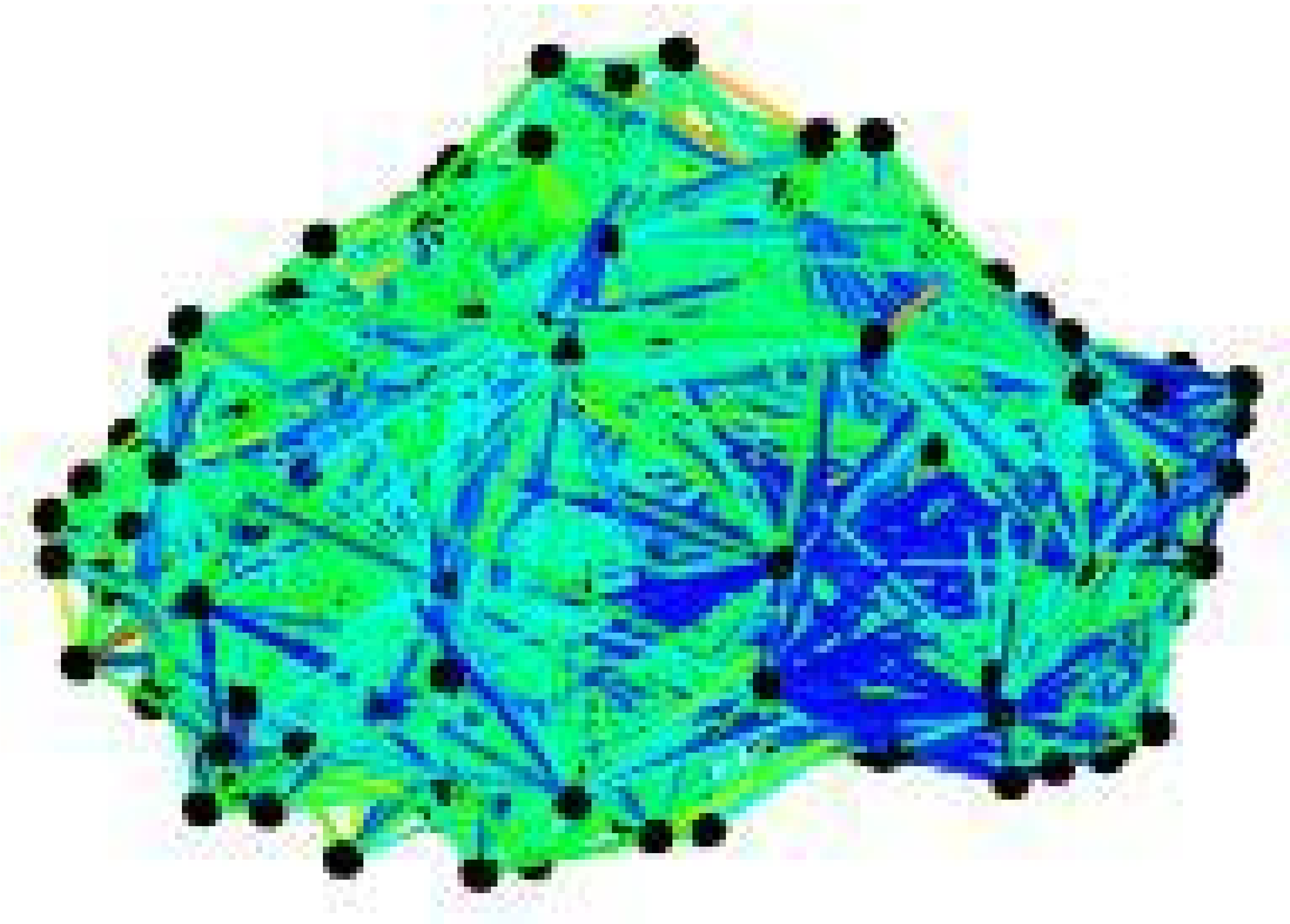
- ## Design & Implementation
  - Completely transparent to the user
  - New option to GCC C/C++/Fortran compiler (`-fmpc_privatize`)
  - When parsing or creating a new global variable: flag it as thread-local
  - Generate runtime calls to access such variables (extension of TLS mechanism)
  - Linker optimization for reduce overhead of global variable access

Discutions autour des applications

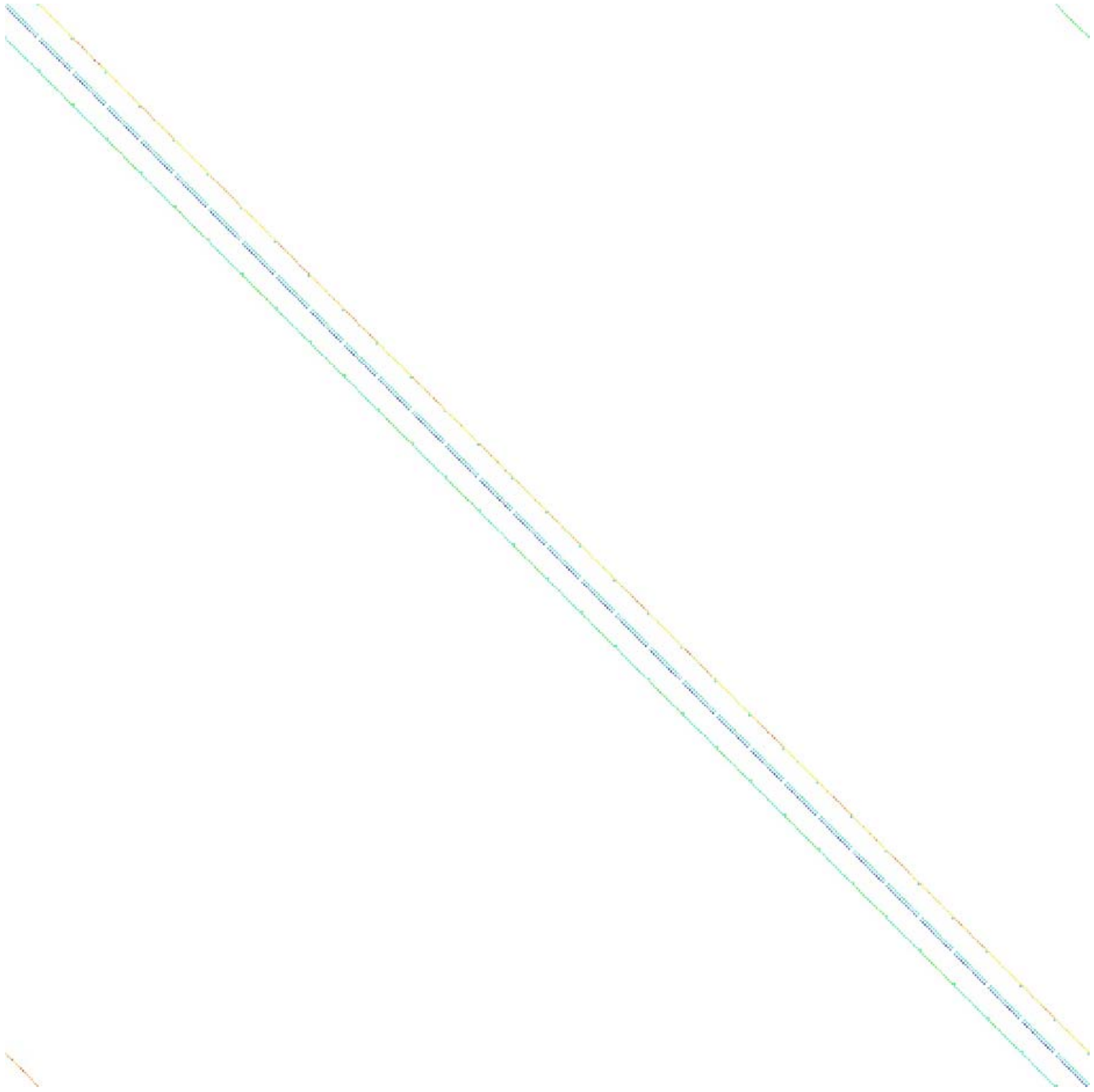
Les mini-apps (topologie)



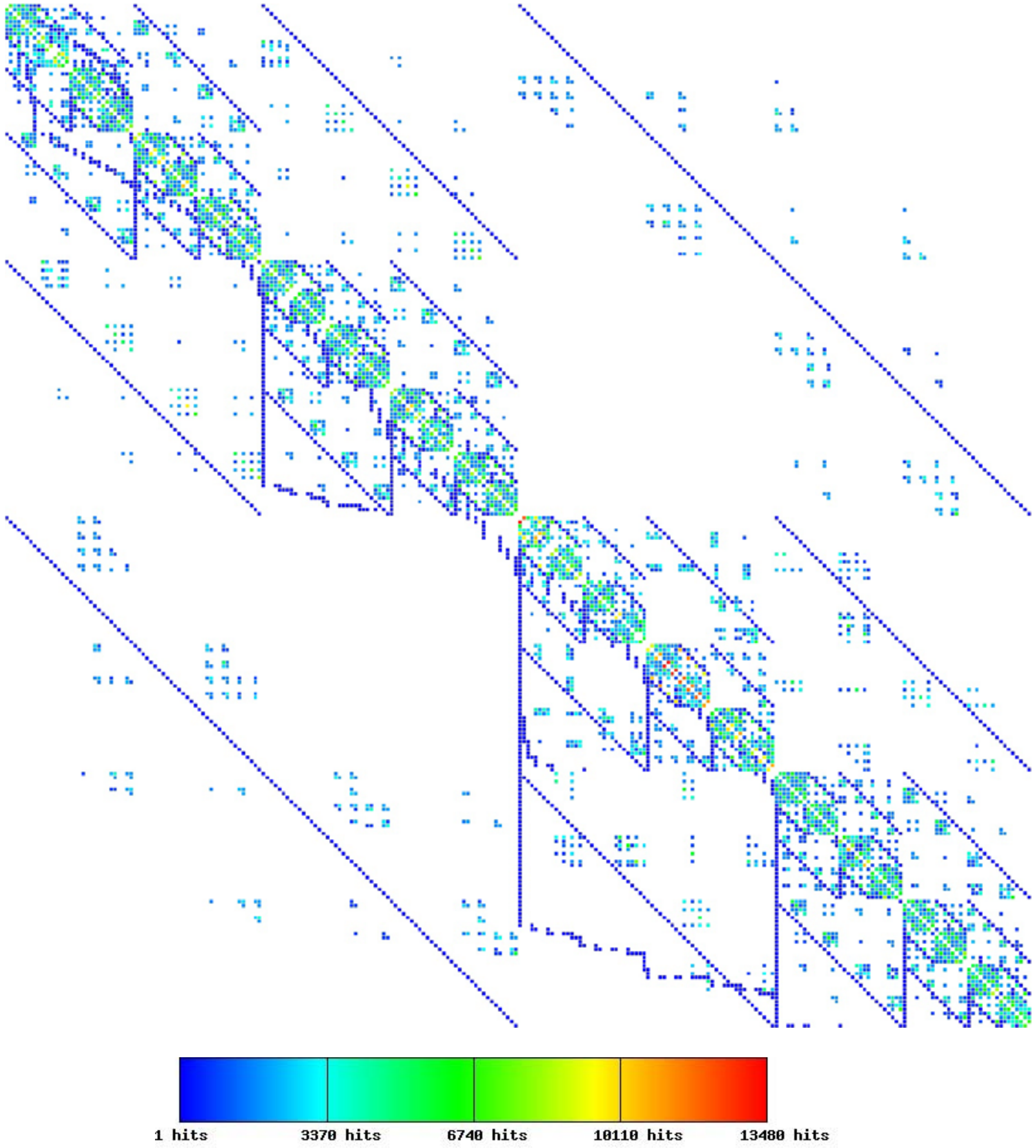
Une vraie application (topologie)



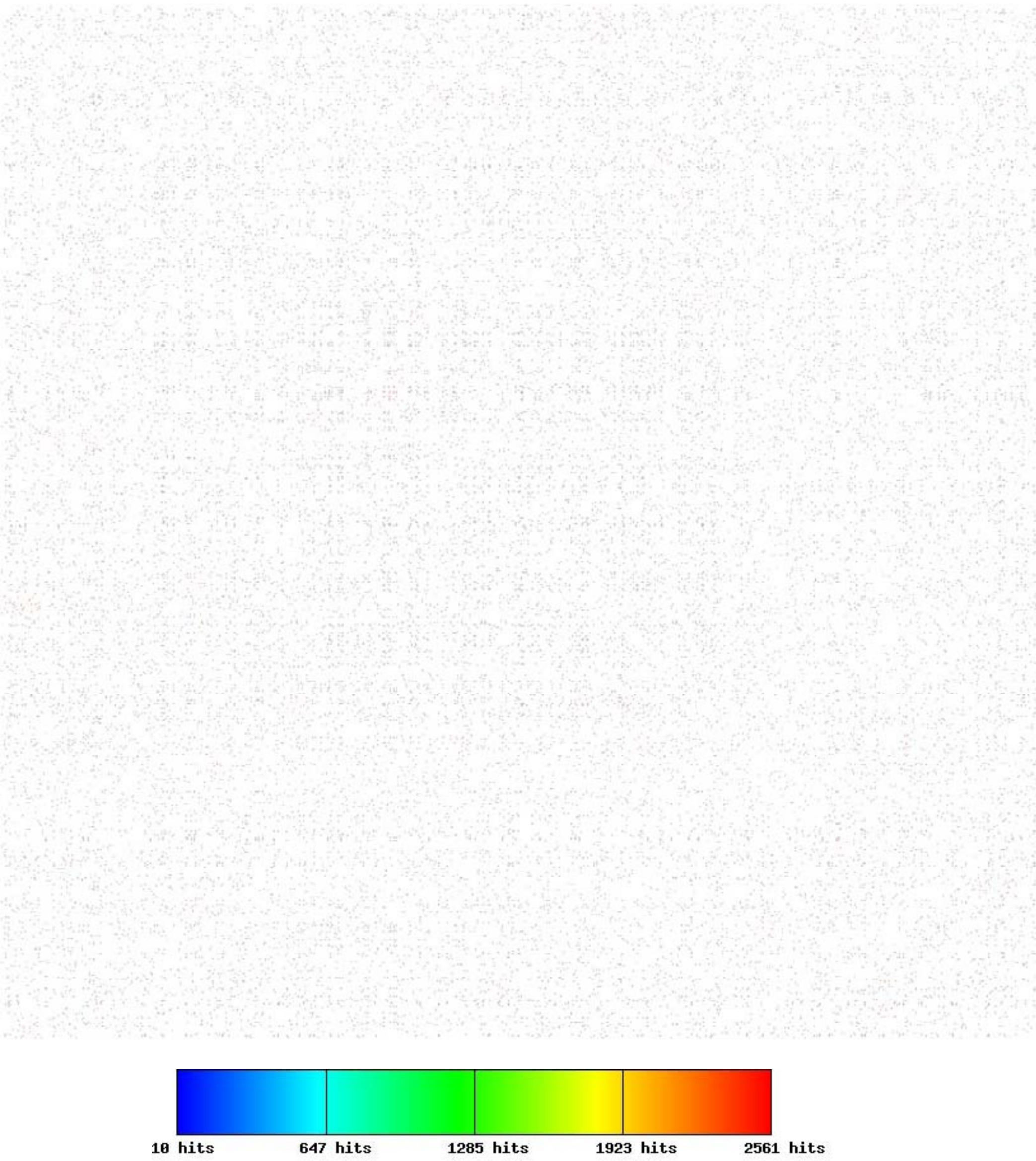
Les mini-apps (matrice des communications)



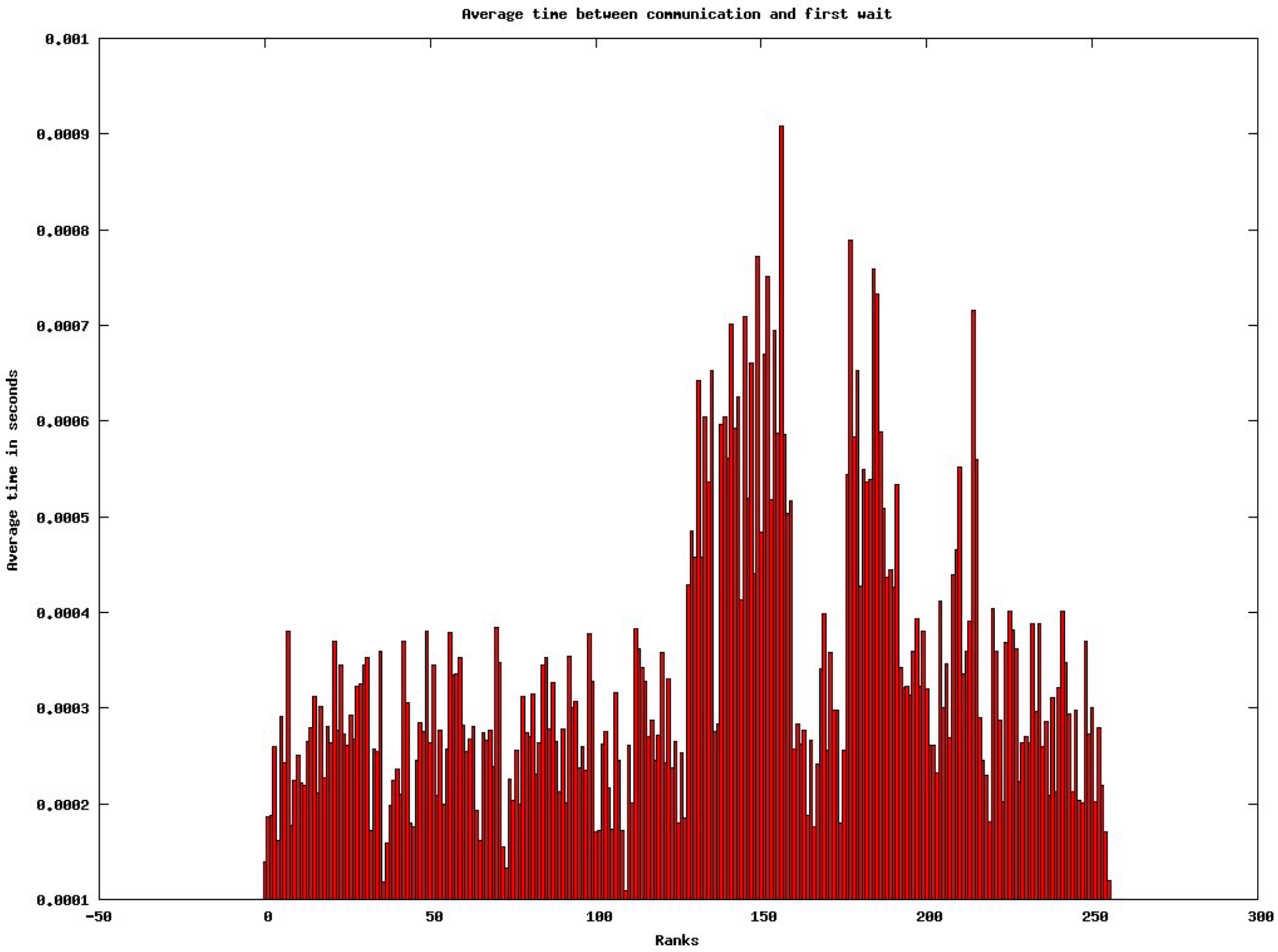
Une vraie application (matrice des communications)



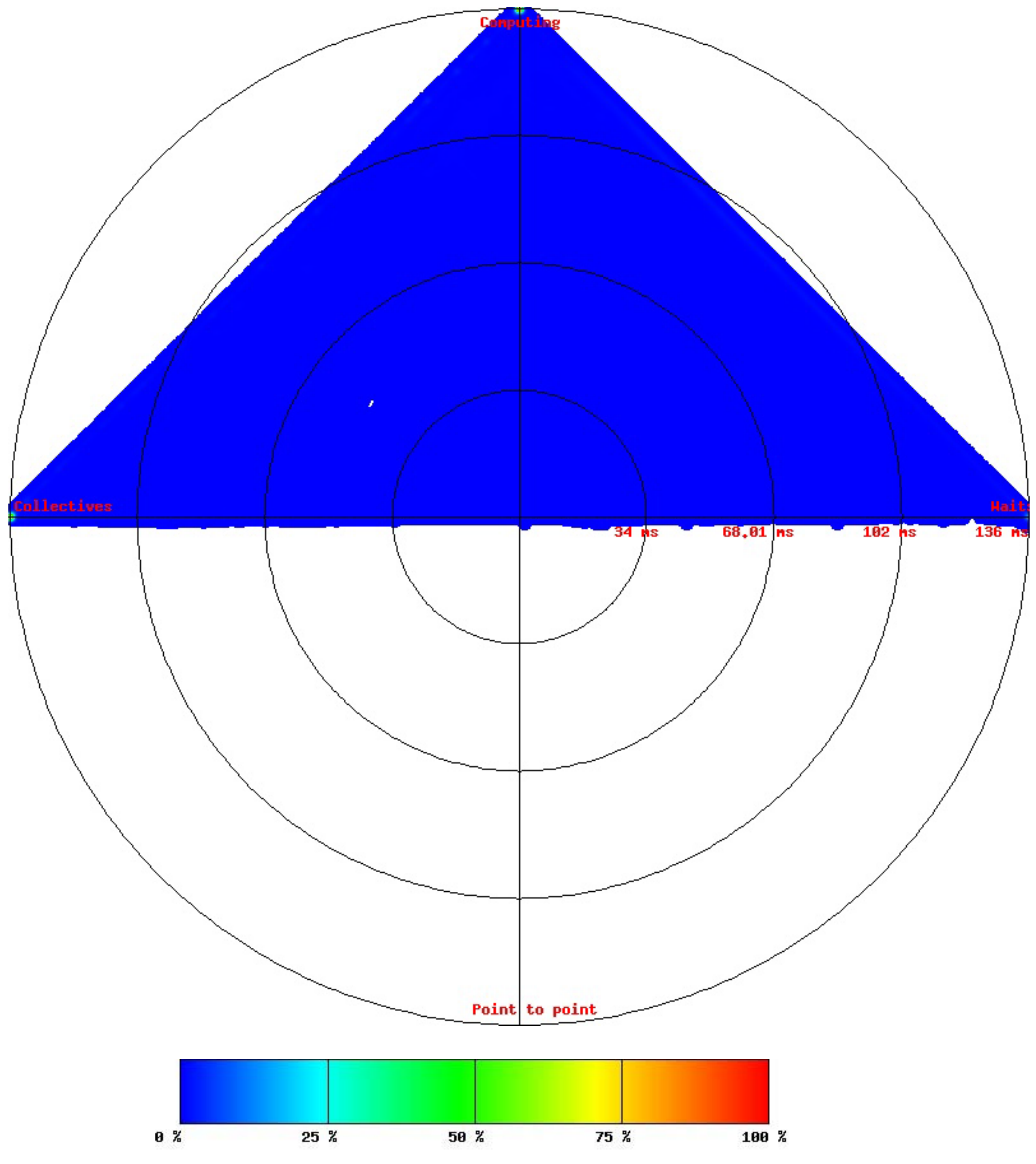
Une vraie application (matrice des communications)



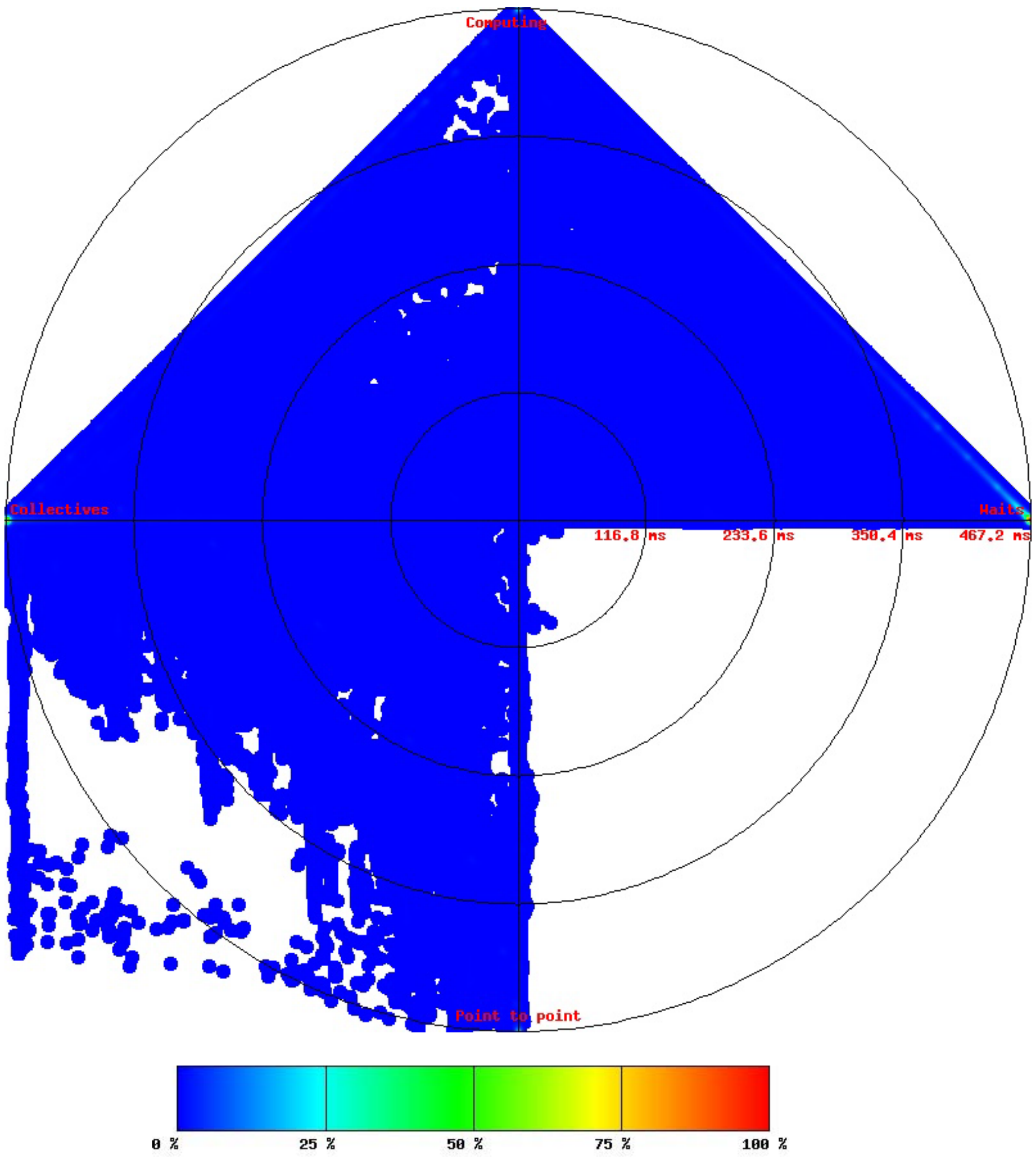
Possibilité d’asynchronisme



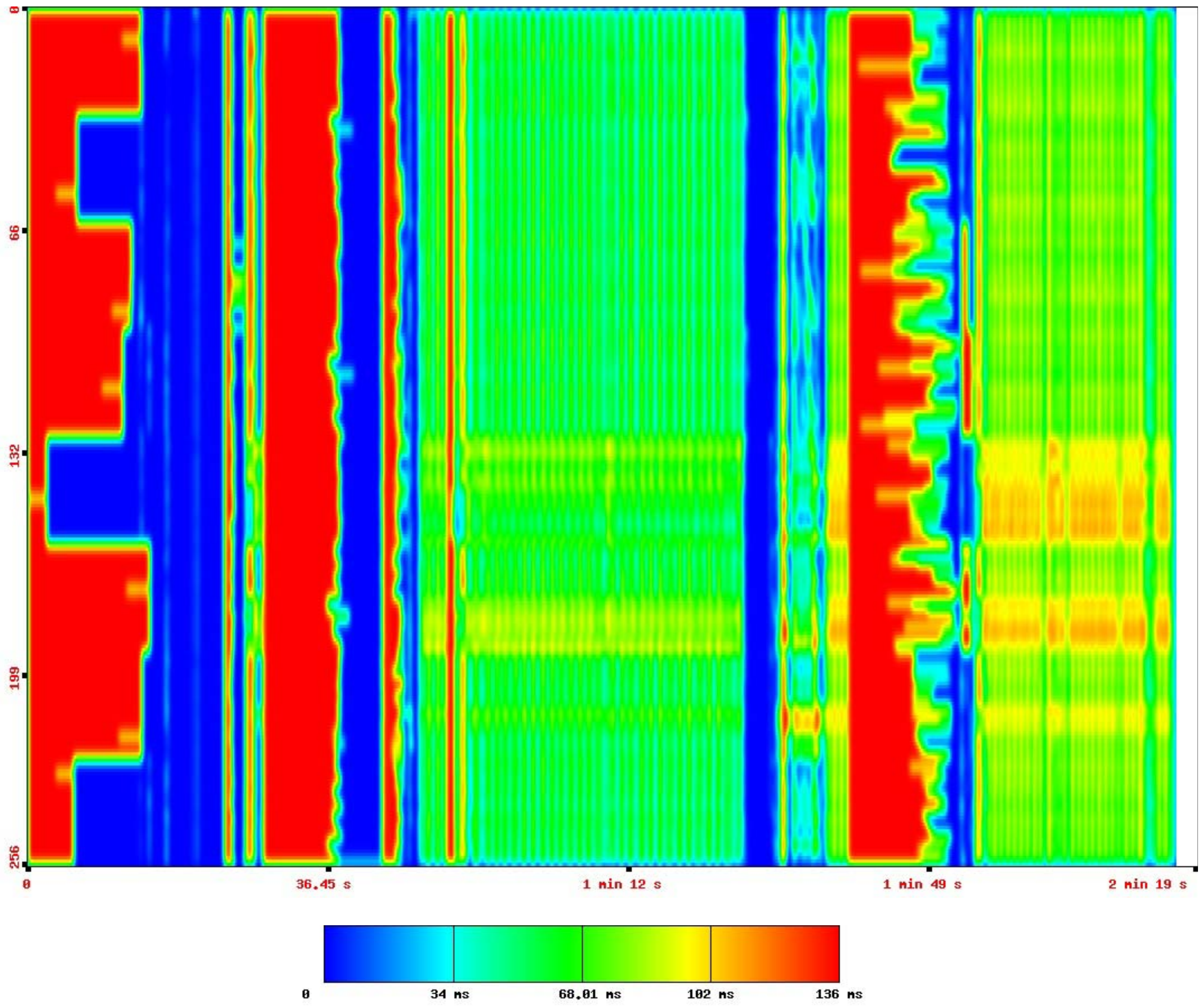
Sensibilité du code



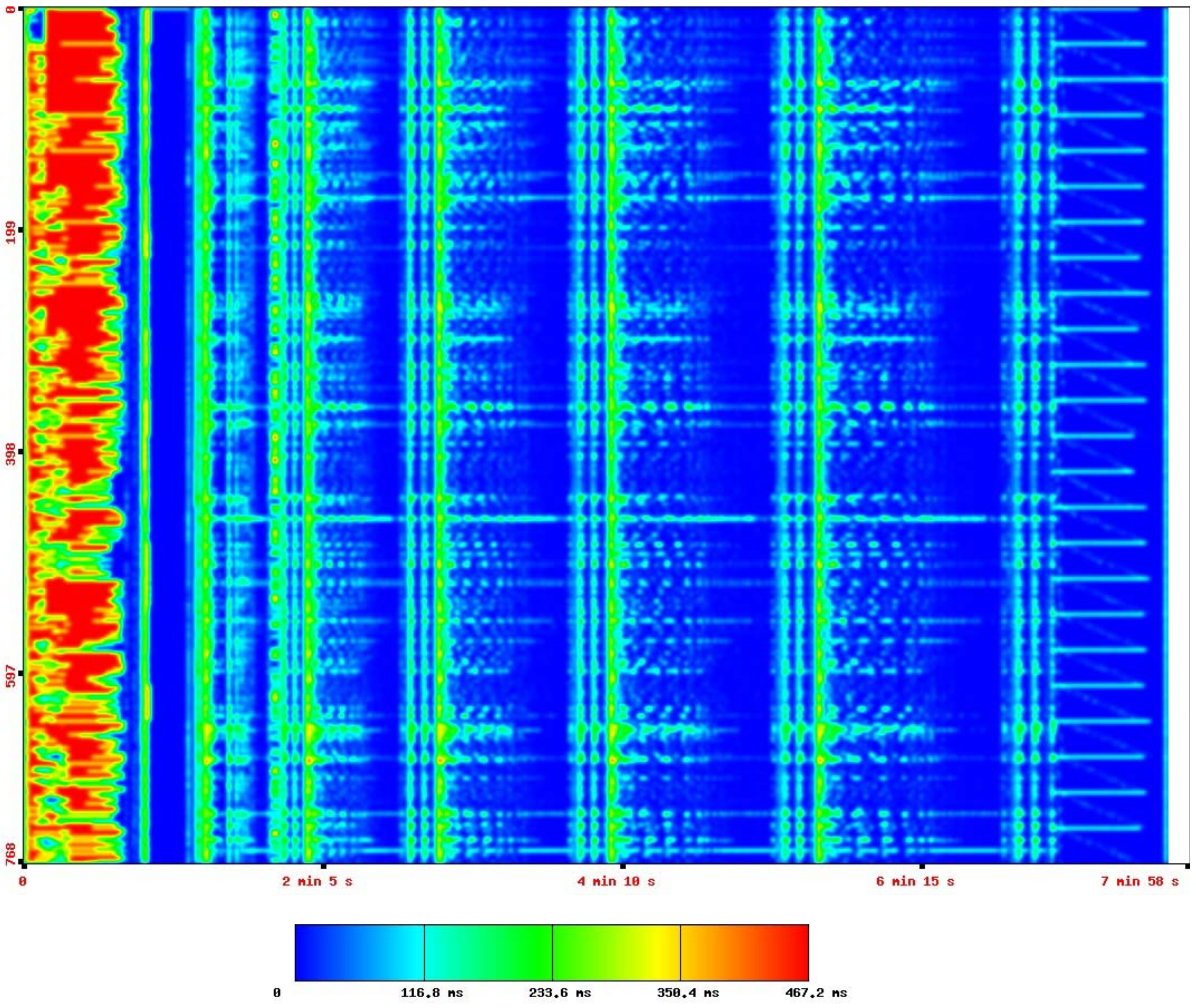
Sensibilité du code



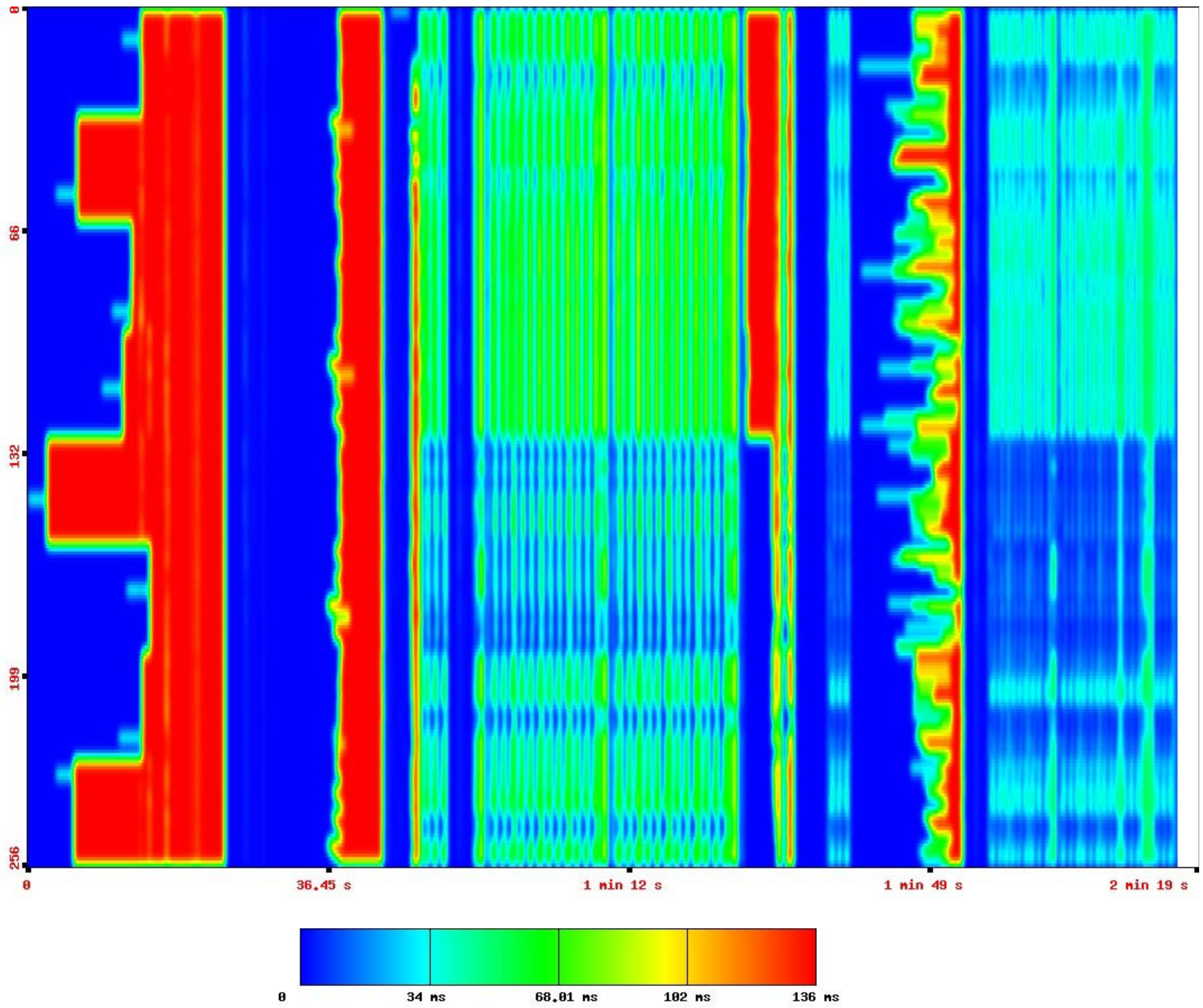
Temps non MPI



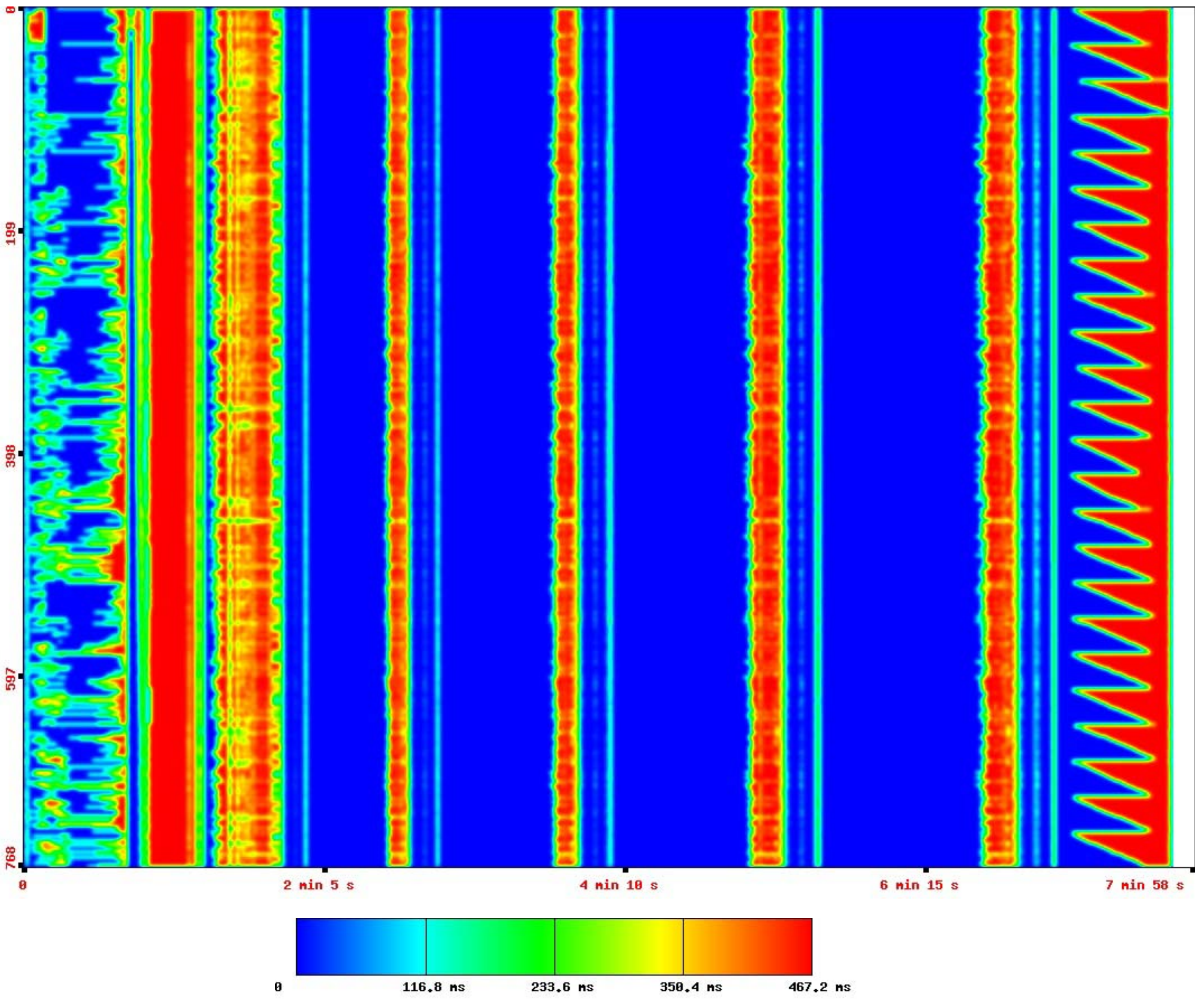
Temps non MPI



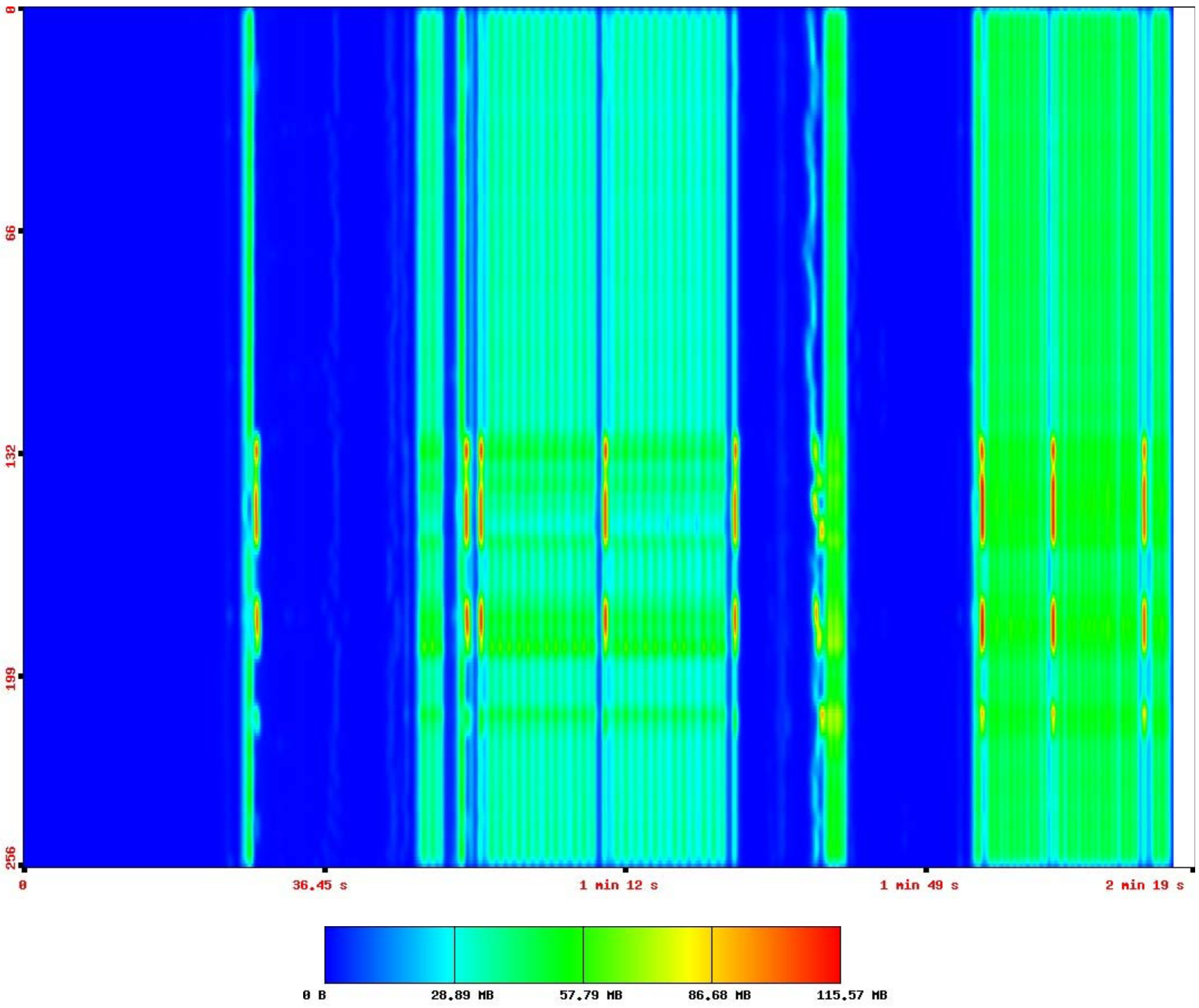
Attente dans les collectives



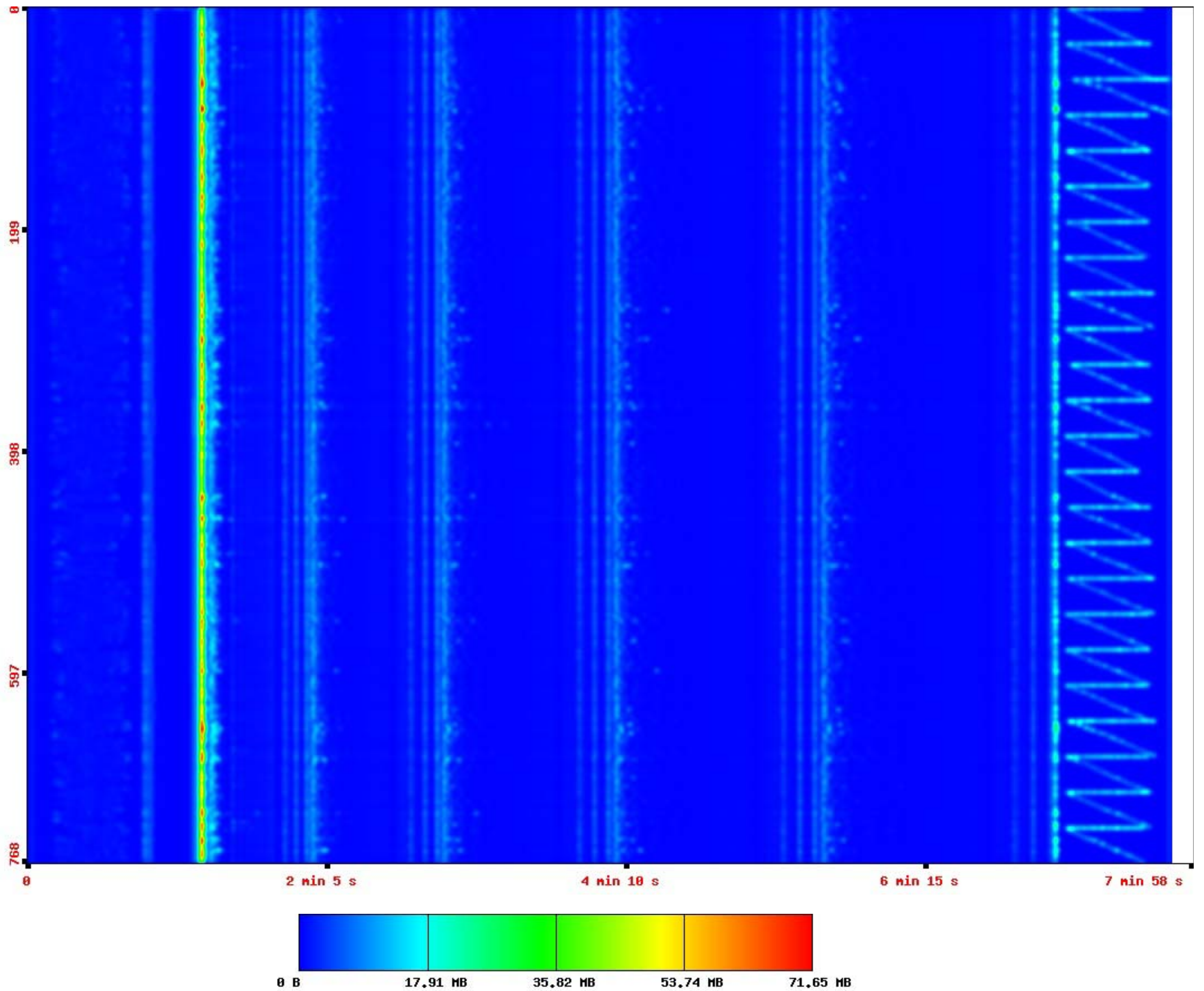
Attente dans les collectives



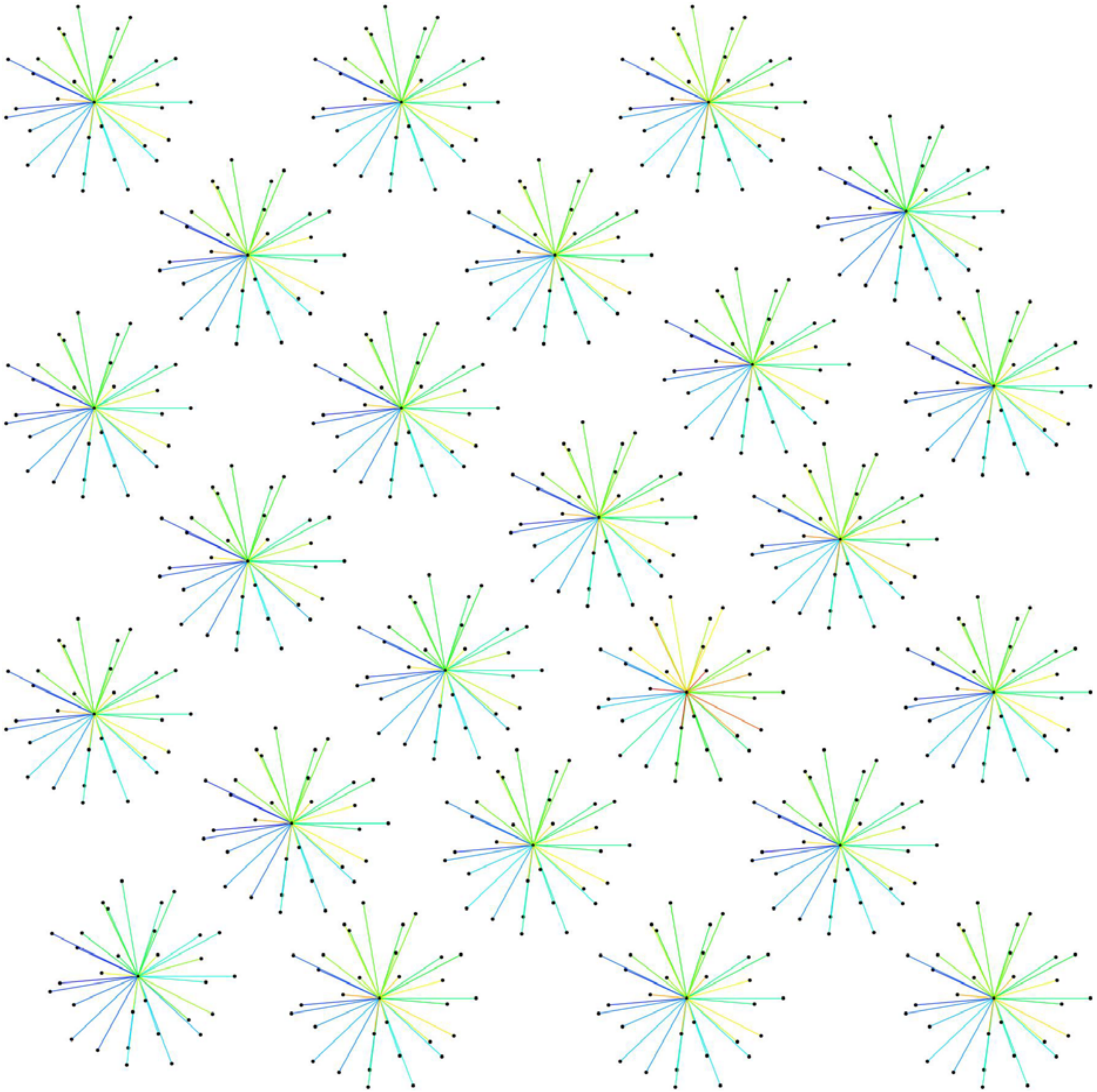
Allocation mémoire



Allocation mémoire



Que faire???



CONCLUSION/FUTURE WORK

Conclusion

- ## Runtime optimization
  - Provide widely spread standards
  - MPI 1.3, OpenMP 2.5, PThread
  - Available at http://mpc.sourceforge.net
  - Optimized for manycore and NUMA architectures
- ## Programming models
  - Provide unified runtime for MPI + X applications
  - Evaluation of new programming models
- ## Tools
  - Debugger support
  - Profiling
  - Compiler support

Bibliography

2012

- S. Didelot, P. Carribault, M. Pérache, W. Jalby, *Improving MPI Communication Overlap With Collaborative Polling,* (EUROMPI'12)
- A. Maheo, S. Koliai, P. Carribault, M. Pérache, W. Jalby, *Adaptive OpenMP for Large NUMA Nodes*, (IWOMP'12)
- M. Tchiboukdjian, P. Carribault, M. Pérache, *Hierarchical Local Storage: Exploiting Flexible User-Data Sharing Between MPI Tasks*, (IPDPS'12)
- J.-Y. Vet, P. Carribault, A. Cohen, *Multigrain Affinityfor Heterogeneous Work Stealing*, (MULTIPROG'12)

2011

- P. Carribault, M. Pérache, H. Jourdren, *Thread-Local Storage Extension to Support Thread-Based MPI/OpenMP Applications* (IWOMP'11)

2010

- P. Carribault, M. Pérache, H. Jourdren, *Enabling Low-Overhead Hybrid MPI/OpenMP Parallelism with MPC* (IWOMP'10)
- K. Pouget, M. Pérache, P. Carribault, H. Jourdren, *User Level DB: a Debugging API for User-Level Thread Libraries (MTAAP'10)*

2009

- M. Pérache, P. Carribault, H. Jourdren, *MPC-MPI: An MPI Implementation Reducing the Overall Memory Consumption* (EuroPVM/MPI'09)

2008

- F. Diakhaté, M. Pérache, H. Jourdren, R. Namyst, *Efficient shared-memory message passing for inter-VM communications* (VHPC'08)
- M. Pérache, H. Jourdren, R. Namyst, *MPC: A Unified Parallel Runtime for Clusters of NUMA Machines* (EuroPar'08)