NewMadeleine & StarPU/nmad
Résultats récents et perspectives

Alexandre Denis – Alexandre.Denis@inria.fr
Philippe Swartvagher – Philippe.Swartvagher@inria.fr

Inria Bordeaux – Sud-Ouest
France

Kickoff Solharis – Décembre 2019

Agenda

- Introduction NewMadeleine
- Résultats préliminaires StarPU/NewMadeleine
- Perspectives

1

Introduction NewMadeleine

Clusters of multicore

- Multicore era
- Cluster nodes typically comprise tens of cores
  - Tomorrow: hundreds of cores (Xeon Phi, AMD Threadripper)
- One (or two) network interface
  - Not hundreds
- Network is shared accross threads

Multithreaded Communications

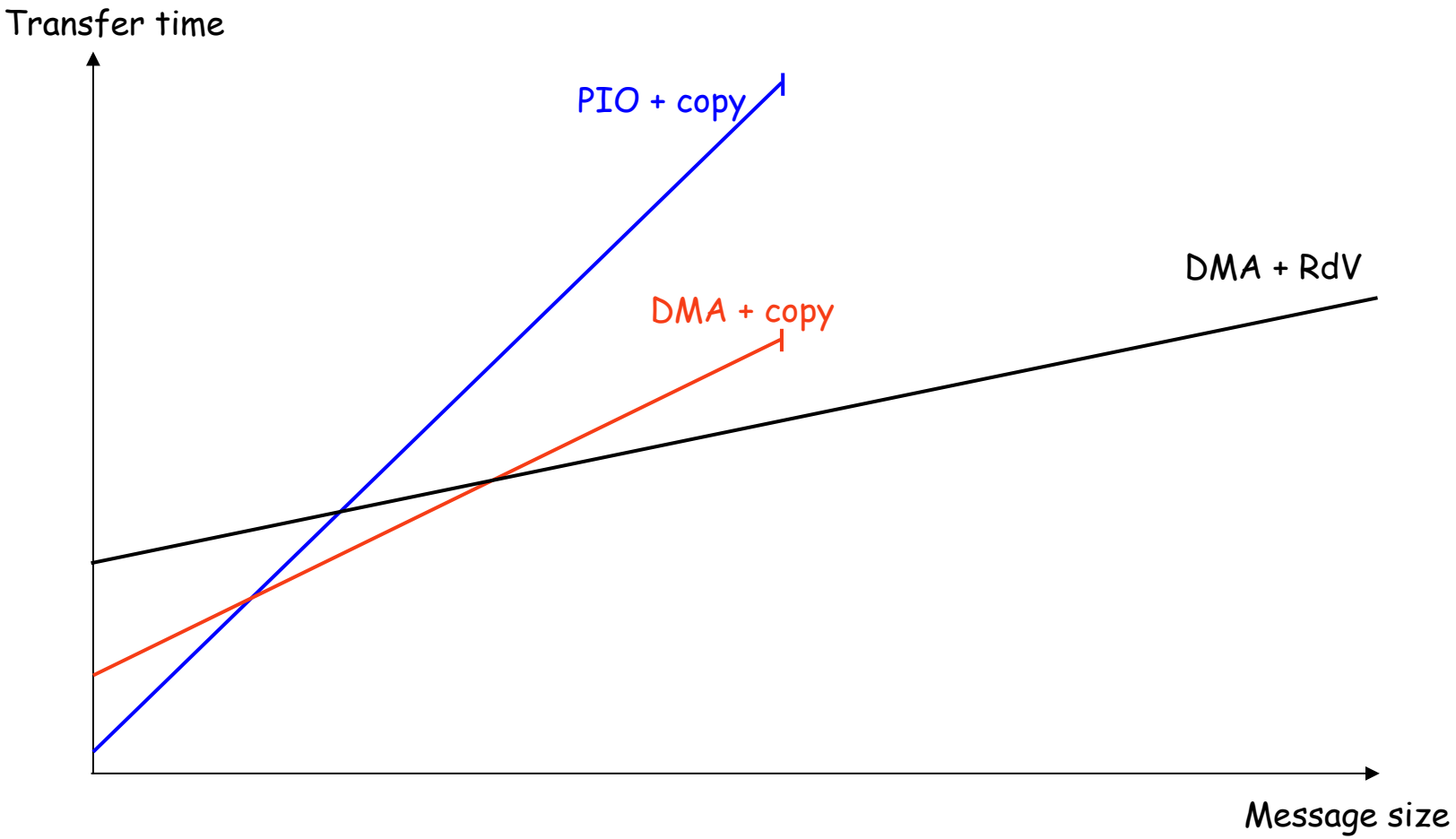
- **Programming models** for clusters of multicore
  - *Pure MPI* do not scale (1 process per core)
  - MPI + X (pthread, OpenMP, TBB, OpenCL, ...)
  - PGAS (MPI used internally)
  - Multithreaded MPI, i.e. MPI_THREAD_MULTIPLE
- Multi-threading has to be taken into account in communications
  - Has an impact on actual communication performance
  - Multi-threading is no more an "*exotic feature*"
  - Communication layer designed for multi-threaded communications
  - Not added as an afterthought
- **Multithreading is a core feature**- deal with it!

A thread-aware approach for communications

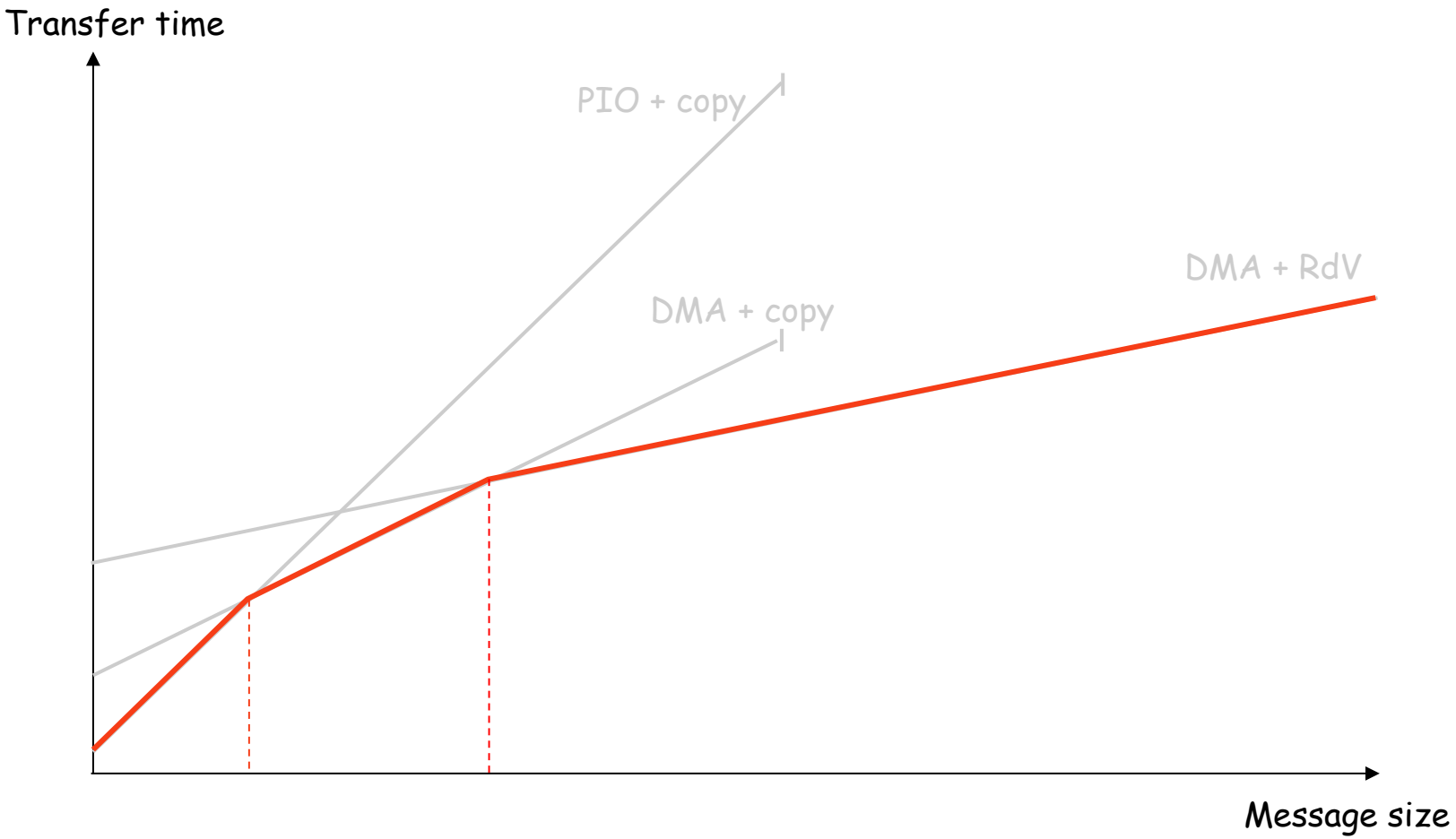
- Impact of multi-core on communications
  - Communication library must support multi-threading
  - Multiple flows from multiple threads share a NIC
  - **Decouple** `MPI_Send/Recv` from NIC send/recv

- New opportunities for optimization!
  - Aggregate packets from different threads
  - The “best” optimization strategy depends on:
    - the capabilities and performance of the underlying network
    - host architecture
    - applications requirements
  - Opportunistically use idle cores
  - **Decide optimization at run time**

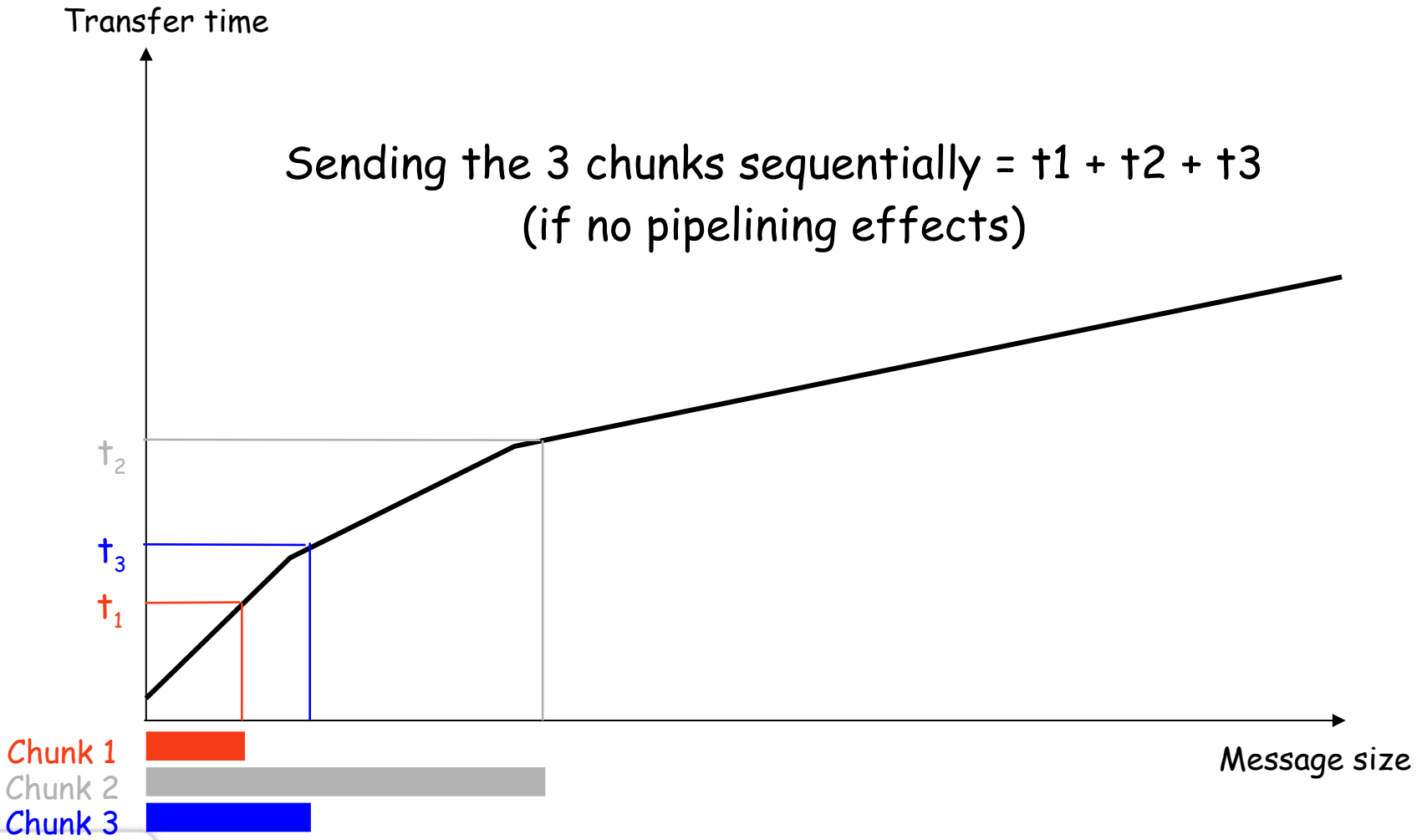
Implementing data transfers efficiently



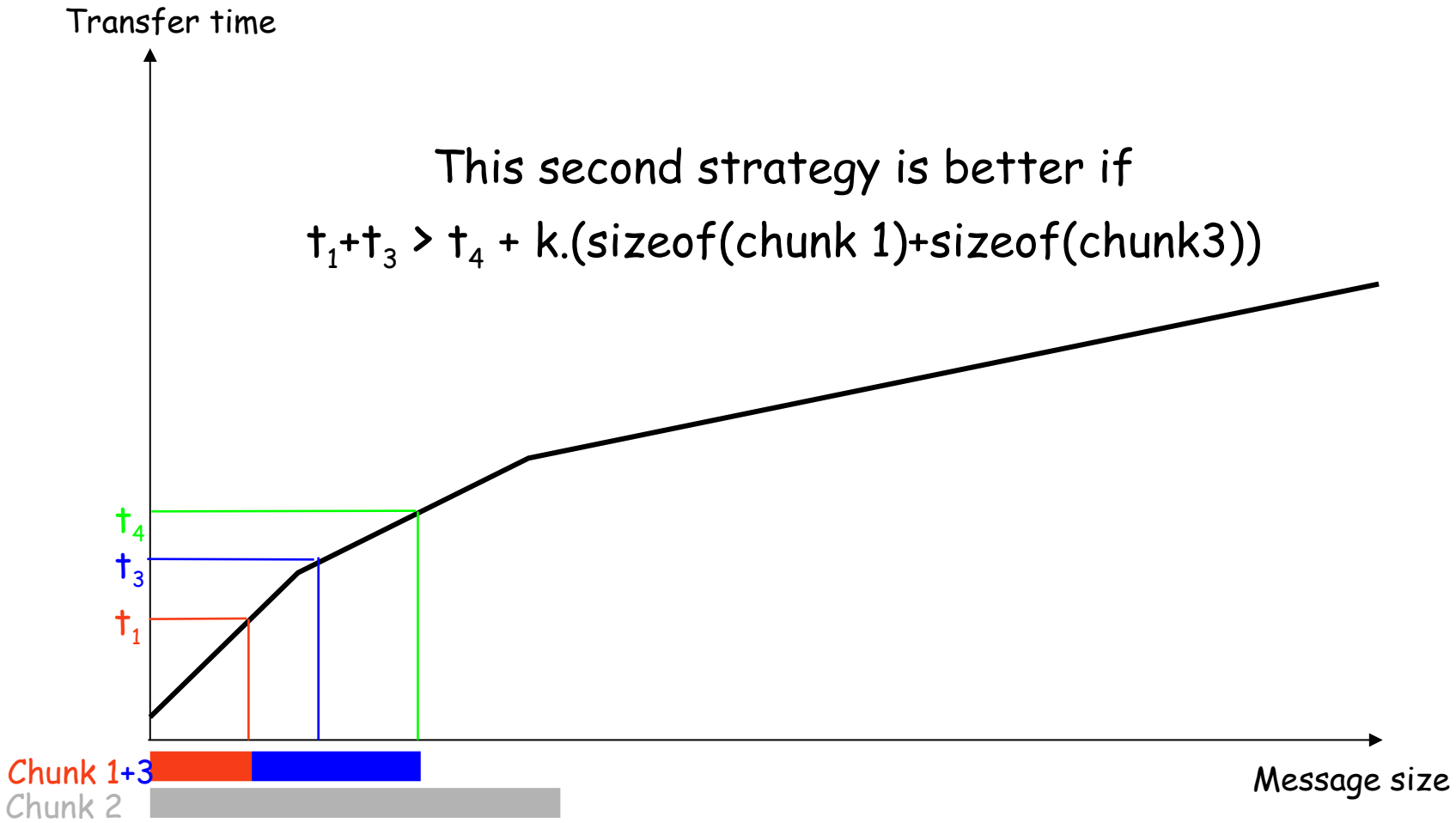
Implementing data transfers efficiently



Implementing data transfers efficiently

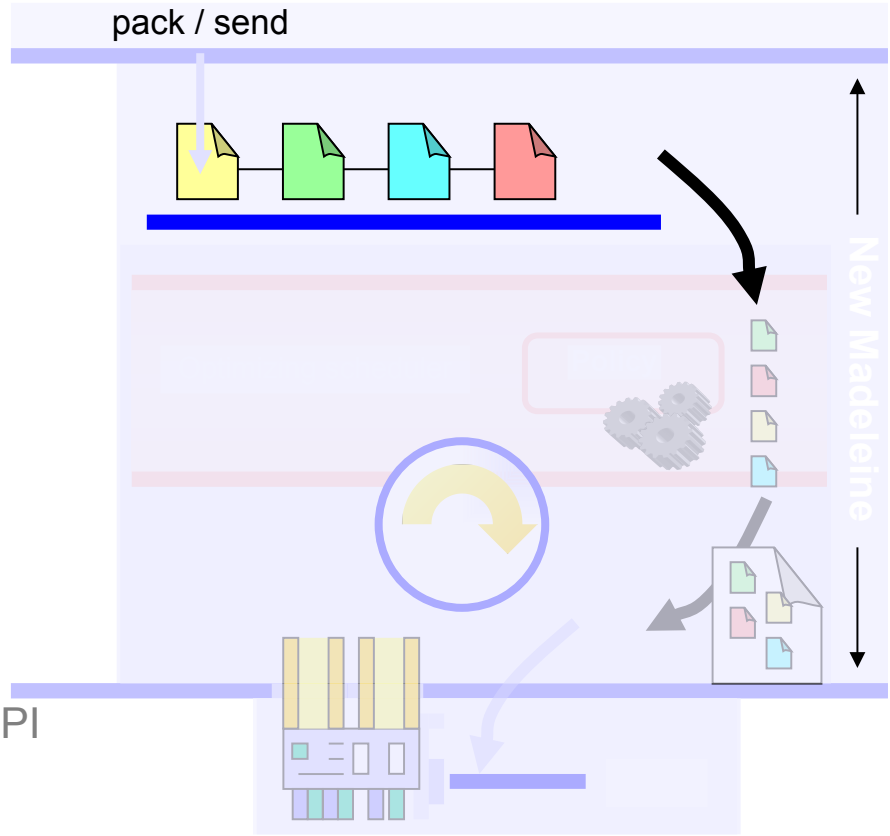


Implementing data transfers efficiently



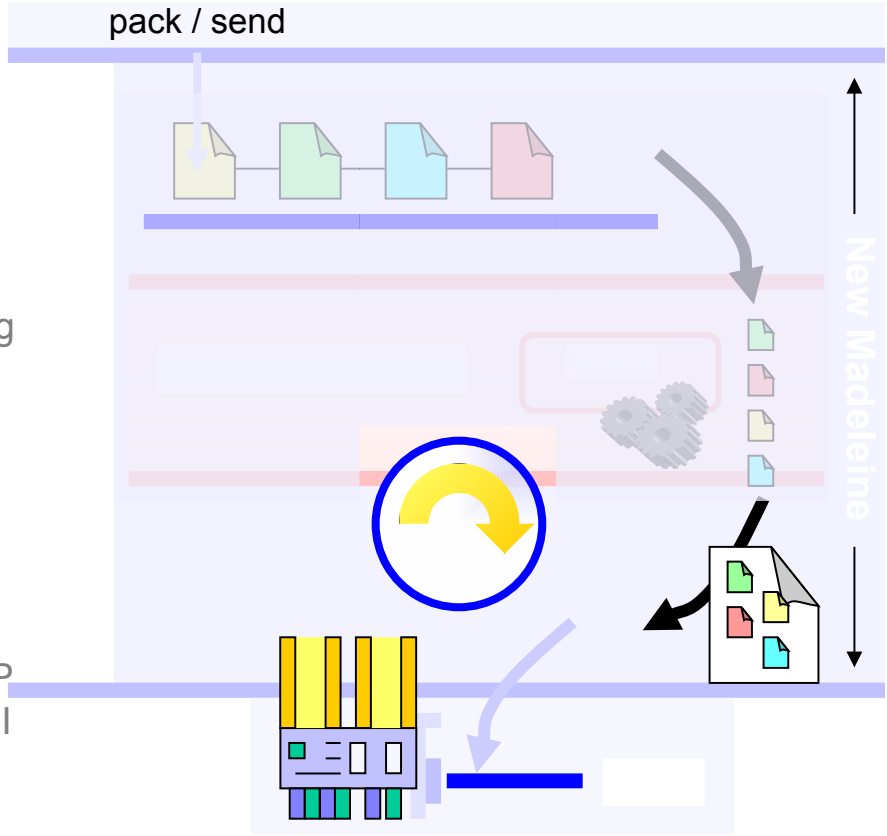
NewMadeleine architecture: the interface layer

- Application submits requests
  - Non-blocking operation
- Meta-data are associated to data
  - Reordering constraints
  - Tag, seq number, etc
- User communication interfaces
  - Incremental message building interface
  - Send-receive interface
  - Simple MPI implementation: MadMPI



NewMadeleine architecture: the network layer

- NewMadeleine activity triggered by NIC
  - **Busy NIC** → gather application requests
  - **Idle NIC** → invoke the **scheduler**
    - Analysis of the user request backlog
    - Application of a strategy
    - Synthesis of a network request
- Available drivers
  - Infiniband verbs, libfabric (OFI), PSM (InfiniPath), PSM2 (OmniPath), shm, TCP sockets, (Myrinet MX, Quadrics Elan, SCI Sisci)

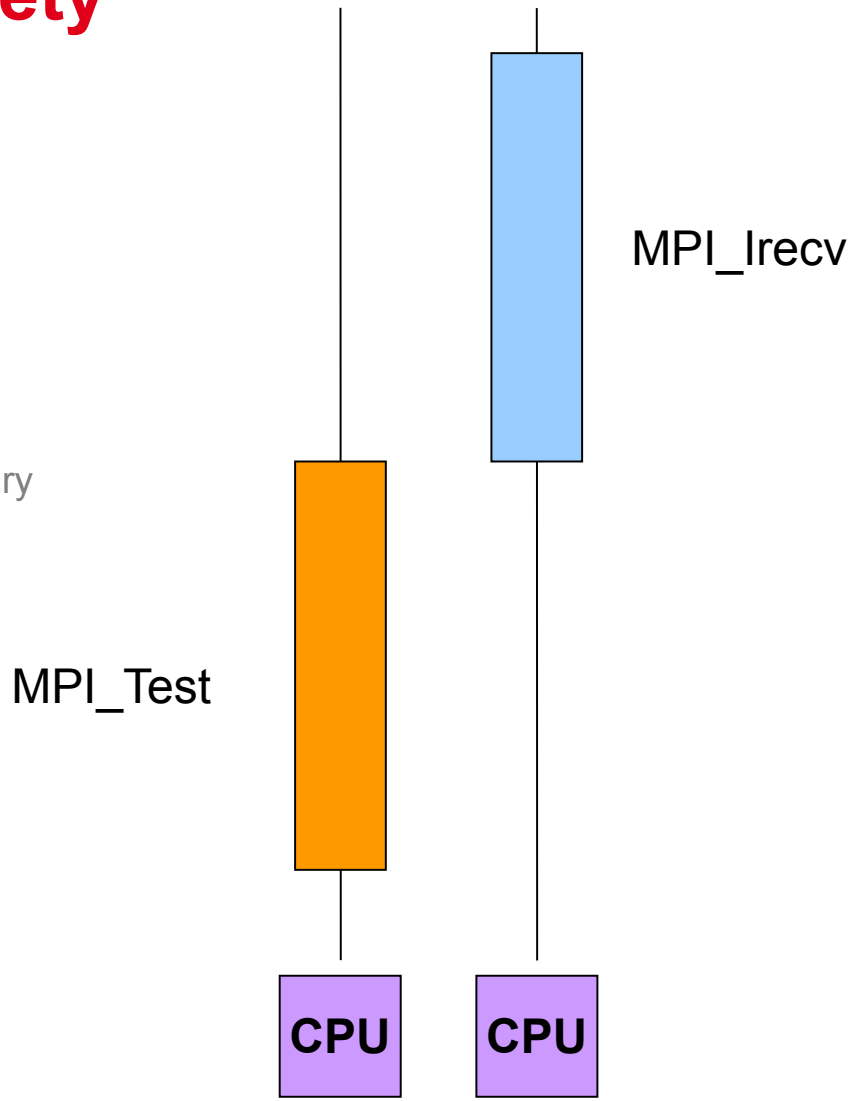


Mixing network and threads

- Multiple levels of threads support in communication subsystem
- Level 1: **Thread safety**
  - Allow simultaneous access to the library
- Level 2: **Background progression** of communication
  - Make communications progress in background (rendez-vous, non-blocking)
- Level 3: **Parallel processing**
  - Use several cores to process communication

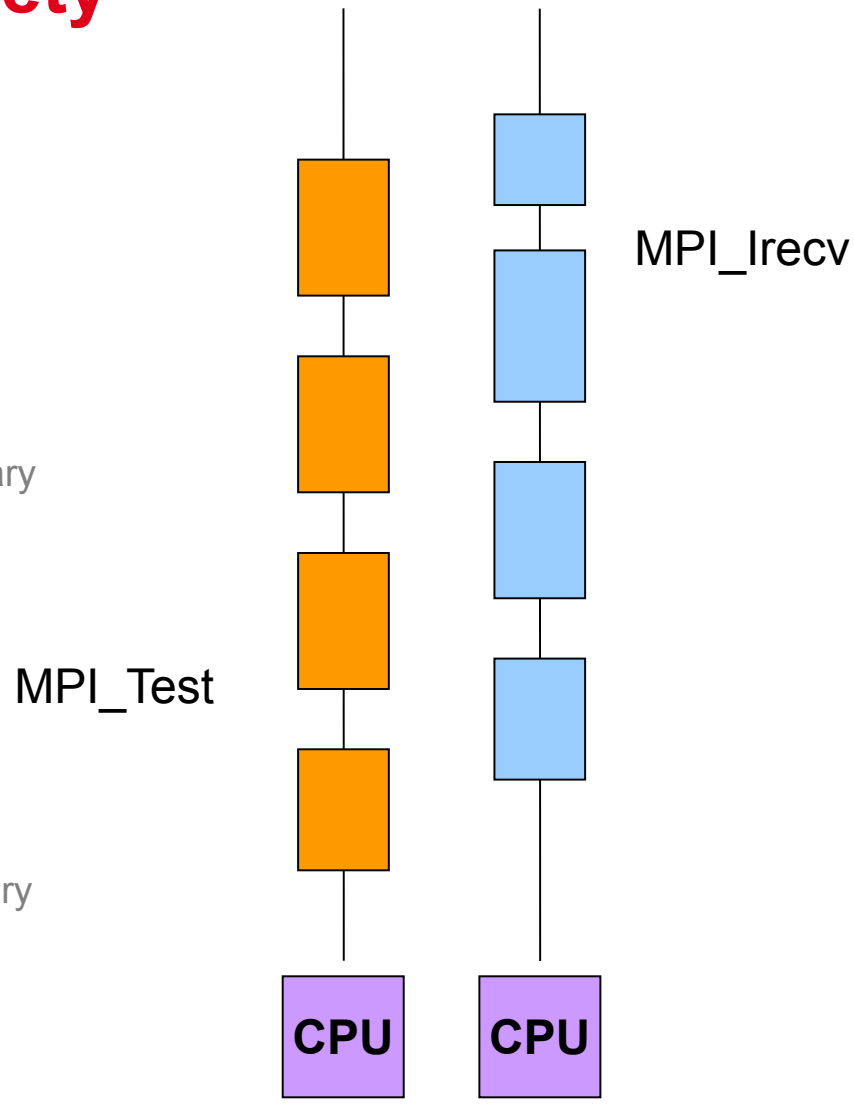
Ensuring thread-safety

- Level 1: thread safety
  - **Coarse grain**
    - Library-wide mutex
    - Avoids simultaneous access to the library
    - Overhead = 140ns
      - negligible



Ensuring thread-safety

- Level 1: thread safety
  - Coarse grain
    - Library-wide mutex
    - Avoids simultaneous access to the library
    - Overhead = 140ns
  - **Fine grain**
    - Action-wide mutexes
    - Local thread-safety
    - Allows simultaneous access to the library
    - Overhead = 230ns

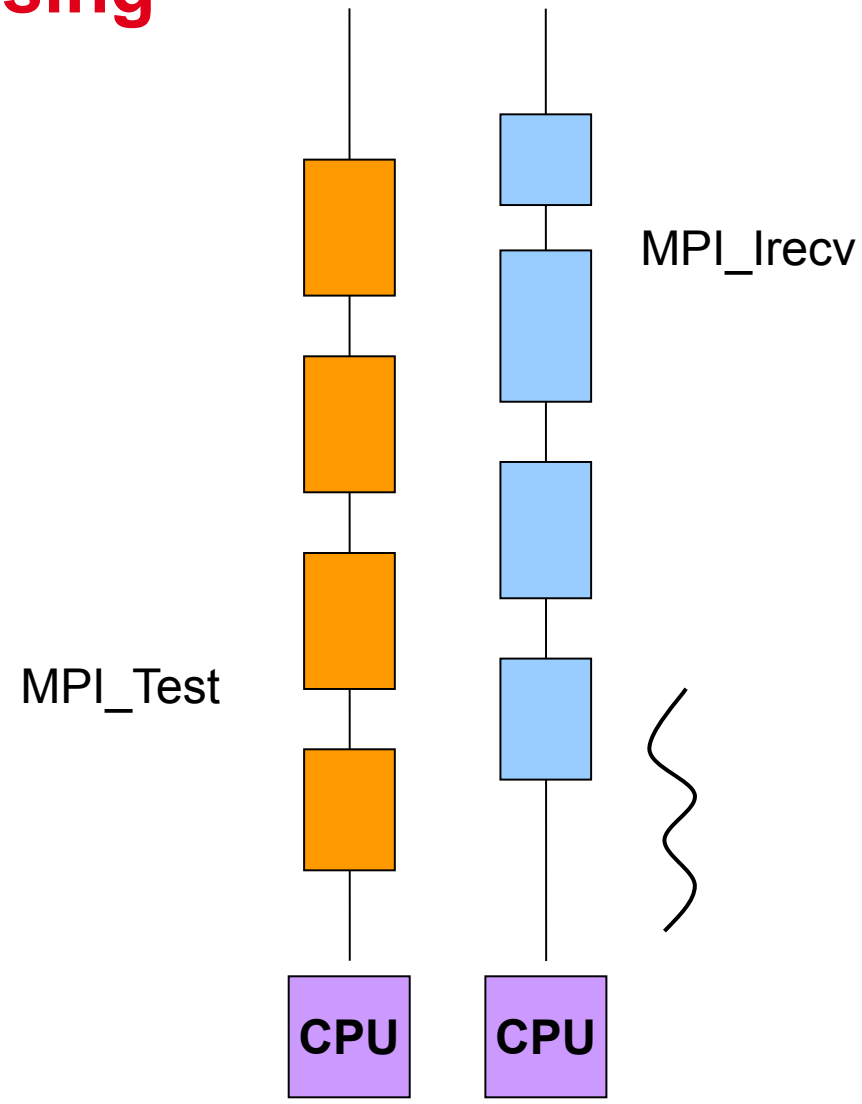


Active waiting vs. Passive waiting

- **Active Waiting**
  - High reactivity
  - No context switch
  - Contention on polling
  - Waste CPUs
- **Passive Waiting**
  - Efficient concurrent waiting
  - Possibility to schedule application threads
  - High overhead (context switches)
- **Fixed-spin waiting**
  - Busy wait for a fixed duration (10µs), then Passive wait
  - Low overhead
  - Efficient concurrent waiting
  - Possibility to schedule application threads

Background processing

- Level 2: **background processing**
- A **progression** thread per NIC
  - Rendezvous handshake progression
    - MX/Myrinet, OpenMPI/TCP
  - Priority issue on overloaded systems
  - Overhead :
    - 400ns (inter-core, same chip synchronization)
    - 2-3us. (inter-chip synchronization)
    - Depends on thread placement

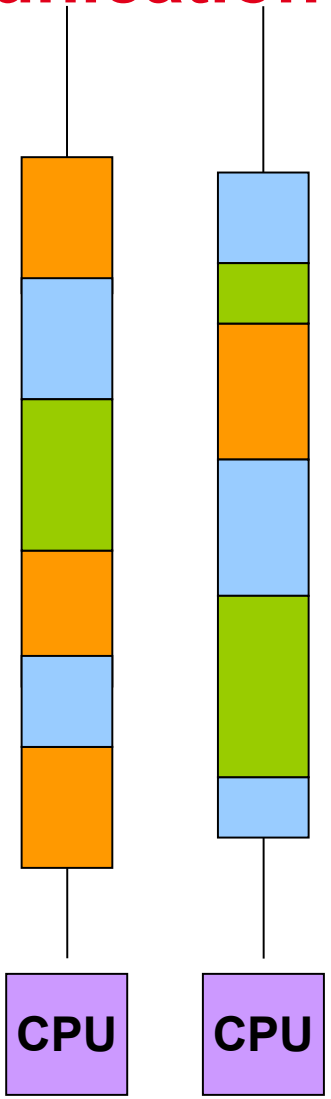


Parallel processing of communication flows

- Level 3: take benefit from multi-core:

The **PIOMan** communication manager

- Communication processing seen as a sequence of operations scheduled through *hooks* in thread scheduler: idle, timer, context switch
  - Operations may be scheduled on any core
  - Load balancing of processing
    - Idle cores 'help' working cores
  - Offloading of asynchronous operations
  - No priority issue
  - Reduced overhead : 400ns
    - Placement is controlled
    - Less context switches



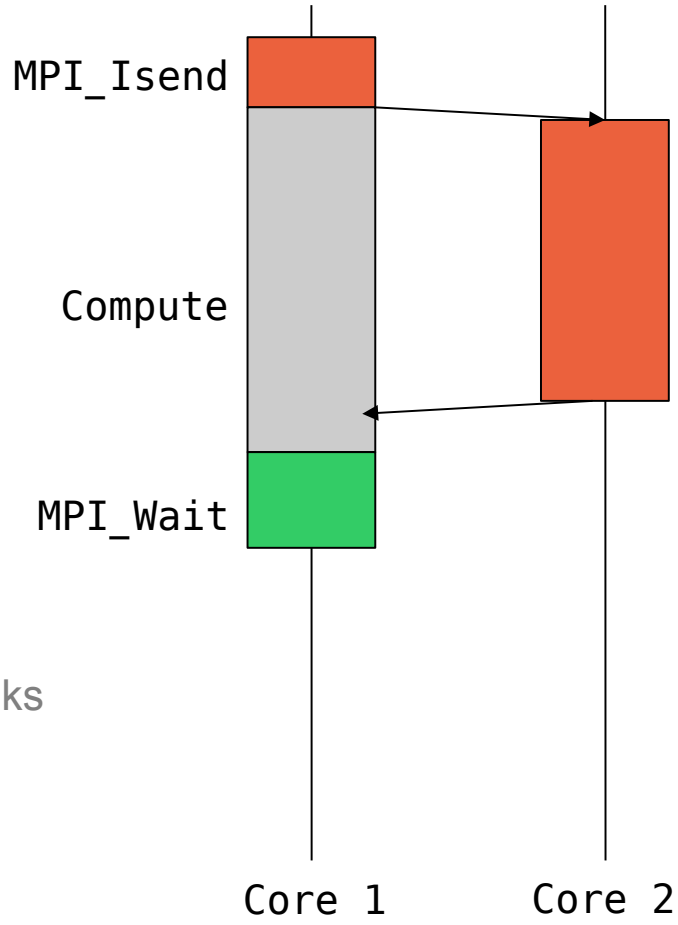
Asynchronous progression

- Communication/computation overlap
- MPI asynchronous progression
  - Communications progress in background
    - e.g. rendez-vous protocol
- User assumes communications progress automagically

→ Communication library is not sequential

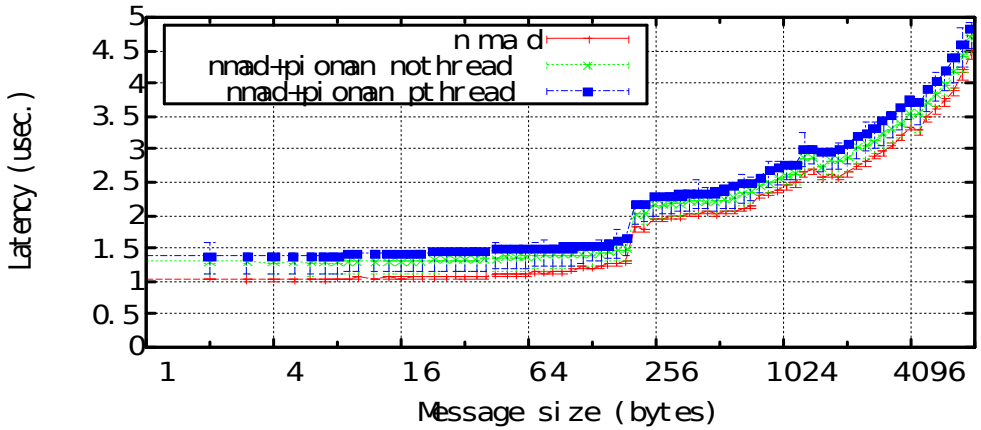
→ **Decompose** communications in basic tasks

→ **Schedule** tasks on cores for progression



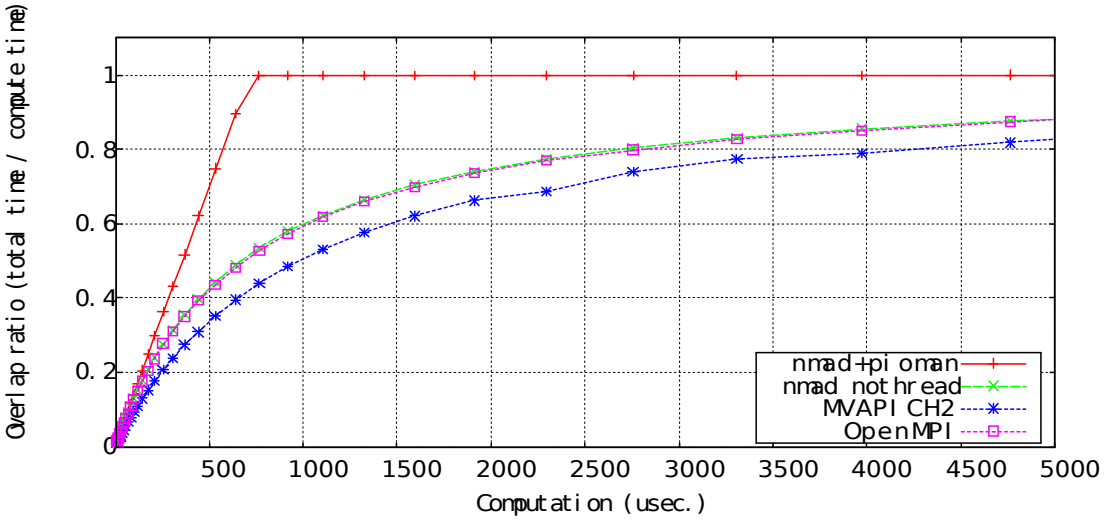
Overhead evaluation - latency

- Latency overhead = ~300-400ns
- Caused by:
  - Tasklets management
  - Atomic ops (*memory fence*)
  - More work done in polling functions
    - e.g. poll all tasks instead of waiting for a single request

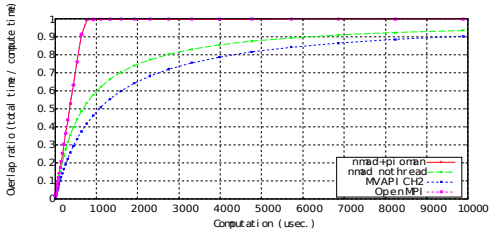


Progression Benchmarks - ratio

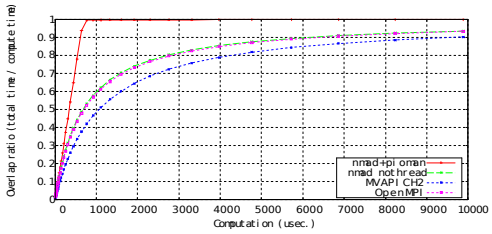
- Overlap ratio = T(computation) / T(computation + communication)
  - Warning! Converges towards 1 even whith no overlap!
  - Metrics from OSU MPI benchmark
- Benchmark: 4MB message, computation on sender and receiver sides
- MadMPI + Pioman gets **100% overlap**
  - As soon as communication is a as long as computation



Both sides overlap



Sender side overlap



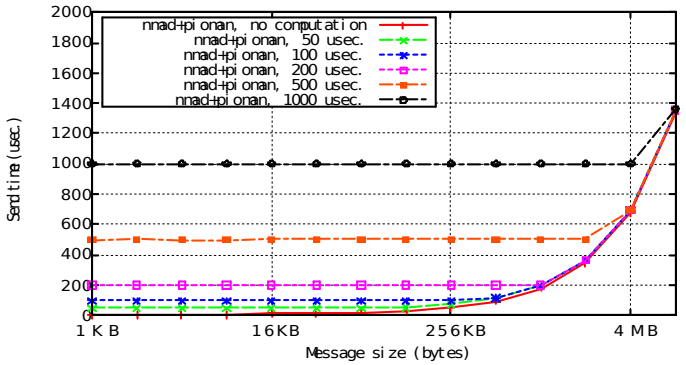
Receiver side overlap

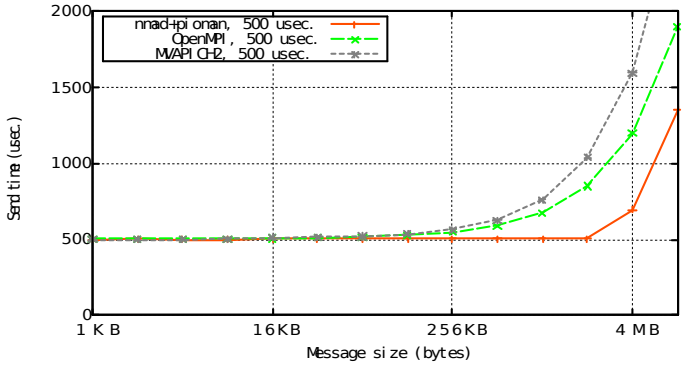
Progression Benchmarks – send time

- Overlap MPI send and computation

```
MPI_Isend(...);
Computation();
MPI_Wait(...);
```

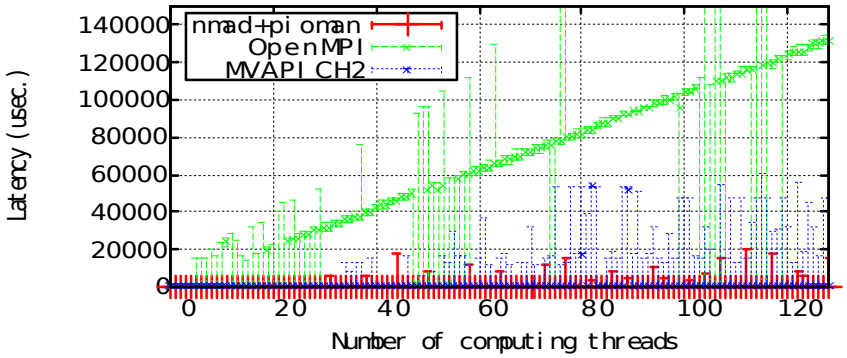
  - Measure total time on sender side
- MadMPI + Pioman
  - total = max(computation, send)
  - **100% overlap**
- OpenMPI, MVAPICH2
  - total = computation + send
  - Serialized!





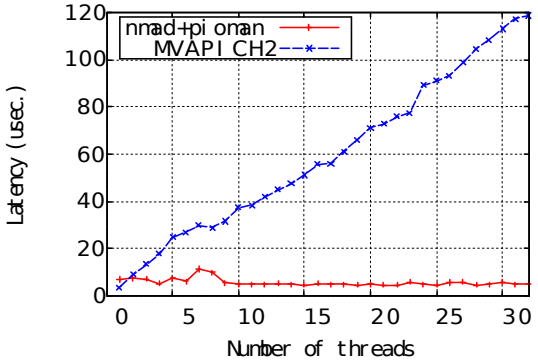
Multithreaded benchmarks

- Multi-threaded applications
- Only 1 thread performs MPI communications
  - i.e. **MPI_THREAD_FUNNELED**
- Competition between communications and computation
- Benchmark: 1MB ping-pong, 32 cores, plot min/max/median
  - **Bounded reactivity** even with massive overload

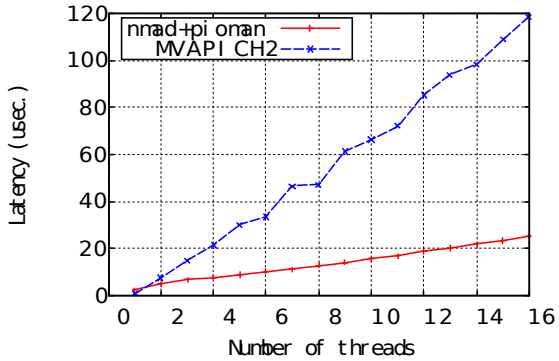


Multithreaded Benchmarks

- Multi-threaded communications
  - i.e. **MPI_THREAD_MULTIPLE**
  - MadMPI+PIOMan v.s. MVAPICH2
  - OpenMPI does not support threads+IB
- 1 byte ping-pong
  - 1 thread to N threads
  - N threads to N threads
- **Constant-time reactivity**
  - Tested up to 200 threads



1-to-N



N-to-N

2

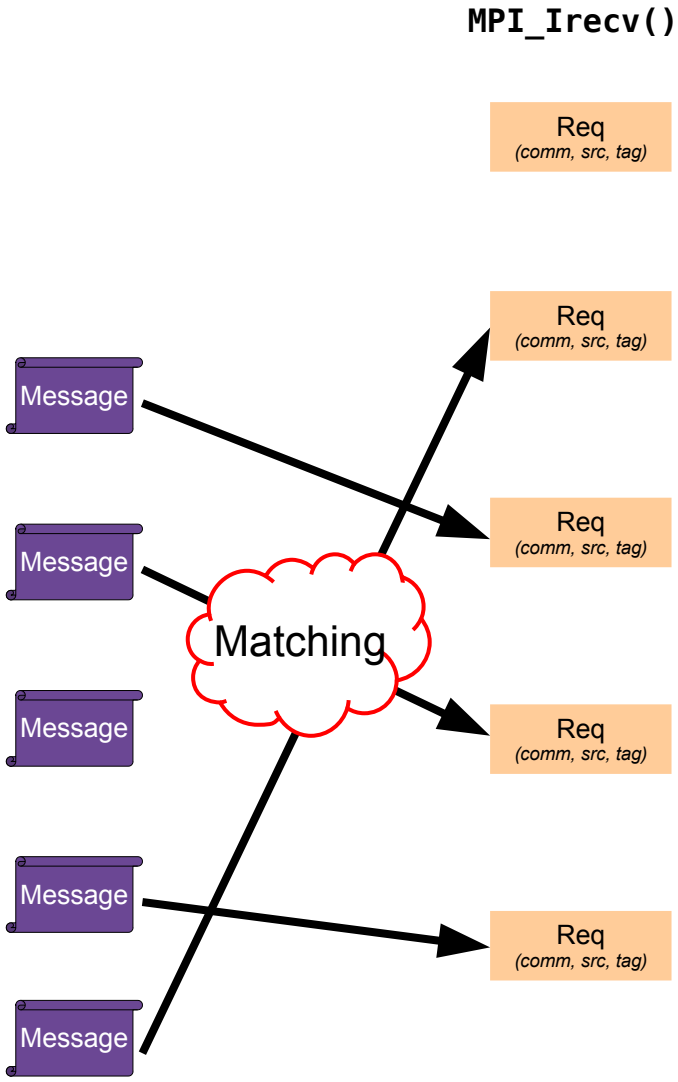
Premiers résultats StarPU/NewMadeleine

Problématiques

- **MPI peu adapté** au schéma de communication de StarPU
  - Tout en point-à-point, pas de collective
  - Communications irrégulières
  - Réactivité, multi-thread
- Meilleure adéquation avec NewMadeleine
- Problématiques
  - Exploitation directe dans StarPU des **fonctionnalités natives** de NewMadeleine
  - Développement dans NewMadeleine de fonctionnalités dédiées à StarPU : **broadcast** dynamique
  - **Compréhension des paramètres** qui influent sur la performance
  - **Passage à l'échelle** de NewMadeleine en nombre de noeuds

Tag matching

- Match an **incoming message** with its corresponding **receive request**
- Criteria
  - Communicator
  - Source (emitter)
  - Tag (32 bits integer given by user)
  - Message order
- Upon **incoming message**
  - Find a pending receive request
- When a **receive request is posted**
  - Find an unexpected message already arrived
- Naive algorithm : **linear complexity**



A constant-time tag matching algorithm

- Classification of receive requests
  - **Fully-specified** requests: given tag and source
  - **Wildcard** requests: `MPI_ANY_SOURCE` & `MPI_ANY_TAG`, tag and source not known in advance
  - **Partially-specified** requests: either `MPI_ANY_SOURCE` or `MPI_ANY_TAG`, but not both at the same time
- Rationale of proposed algorithm:
  - **Different data structures** for each kind of request, with constant-time access
  - **Sequence numbers to interleave** different kinds properly

Constant-time algorithm

Algorithm 1 Matching algorithm upon packet arrival

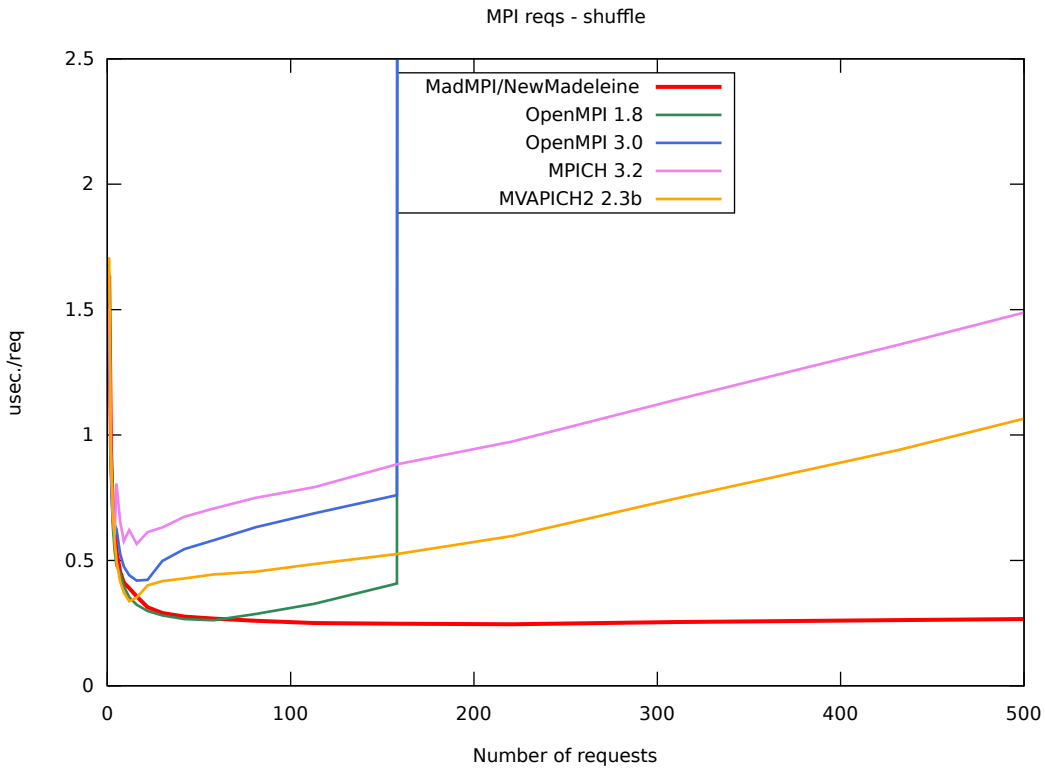
packet arrival from $source$ on tag tag
$spec\_entry \leftarrow$ lookup($specs.reqs$, $(source, tag)$)
$spec\_req \leftarrow$ head($spec\_entry.reqs$)
$req \leftarrow spec\_req$
$w\_req \leftarrow$ head($wildcards.reqs$)
if $req = \emptyset \vee (w\_req \neq \emptyset \wedge w\_req.seq < req.seq)$ **then**
  $req \leftarrow wildcard\_req$
end if
$tags\_entry \leftarrow$ lookup($tags.reqs$, tag)
$tags\_req \leftarrow$ head($tags\_entry.reqs$)
if $req = \emptyset \vee (tags\_req \neq \emptyset \wedge tags\_req.seq < req.seq)$ **then**
  $req \leftarrow tags\_req$
end if
$src\_entry \leftarrow$ lookup($src.reqs$, $(source, comm)$)
$src\_req \leftarrow$ head($src\_entry.reqs$)
if $req = \emptyset \vee (src\_req \neq \emptyset \wedge src\_req.seq < req.seq)$ **then**
  $req \leftarrow src\_req$
end if
{No matching request; enqueue packet as unexpected}
if $req = \emptyset$ **then**
  pushback($packet$, $spec\_entry.unexpected$)
  pushback($packet$, $wildcards.unexpected$)
  pushback($packet$, $tags\_entry.unexpected$)
  pushback($packet$, $src\_entry.unexpected$)
end if
return req

Algorithm 2 Matching algorithm upon receive request posted

receive request posted for $source$ and tag
if $source \neq \emptyset \wedge tag \neq \emptyset$ **then**
  $spec\_entry \leftarrow$ lookup($spec.unexpected$, $(source, tag)$)
  if $\neg$ empty($spec\_entry.unexpected$ **then**
    return head($spec\_entry.unexpected$)
  else
    pushback($request$, $spec\_entry.reqs$)
  end if
else if $source = \emptyset \wedge tag = \emptyset$ **then**
  if $\neg$ empty($wildcards.unexpected$) **then**
    return head($wildcards.unexpected$)
  else
    pushback($request$, $wildcard\_reqs$)
  end if
else if $source = \emptyset$ **then**
  $src\_entry \leftarrow$ lookup($src.unexpected$, $(source, comm)$)
  if $\neg$ empty($src\_entry.unexpected$ **then**
    return head($src\_entry.unexpected$)
  else
    pushback($request$, $src\_entry.reqs$)
  end if
else if $tag = \emptyset$ **then**
  $tags\_entry \leftarrow$ lookup($tags.unexpected$, tag)
  if $\neg$ empty($tags\_entry.unexpected$ **then**
    return head($tags\_entry.unexpected$)
  else
    pushback($request$, $tags\_entry.reqs$)
  end if
end if

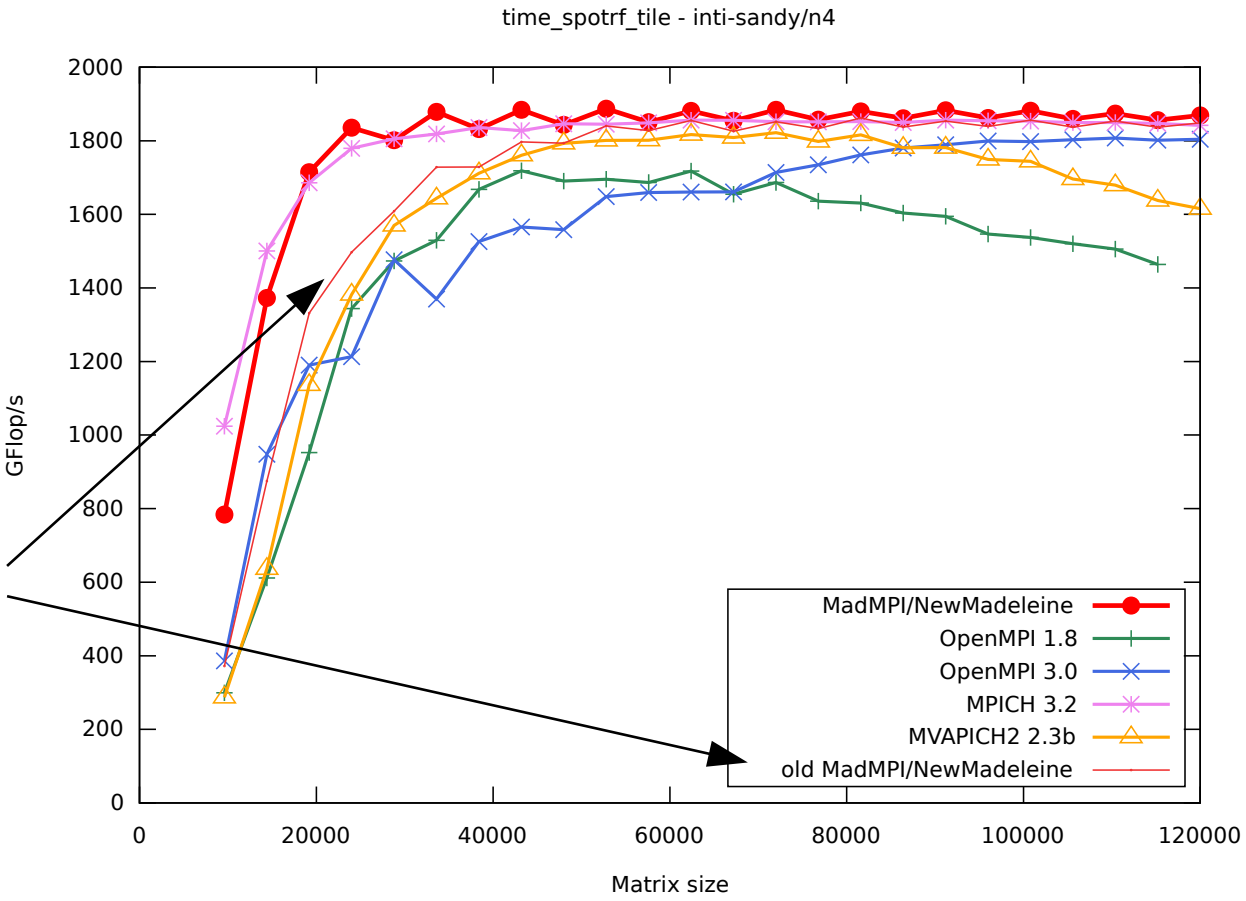
Micro-benchmark - shuffled

- Graph: time per request
  - Lower is better
- Machine: `inti`, IB QDR
- Tags are **shuffled**
- NewMadeleine
  - **bounded time** per req
- Other MPI libraries
  - Time **linear** with number of requests
  - OpenMPI blows up for more than ~160 simultaneous requests



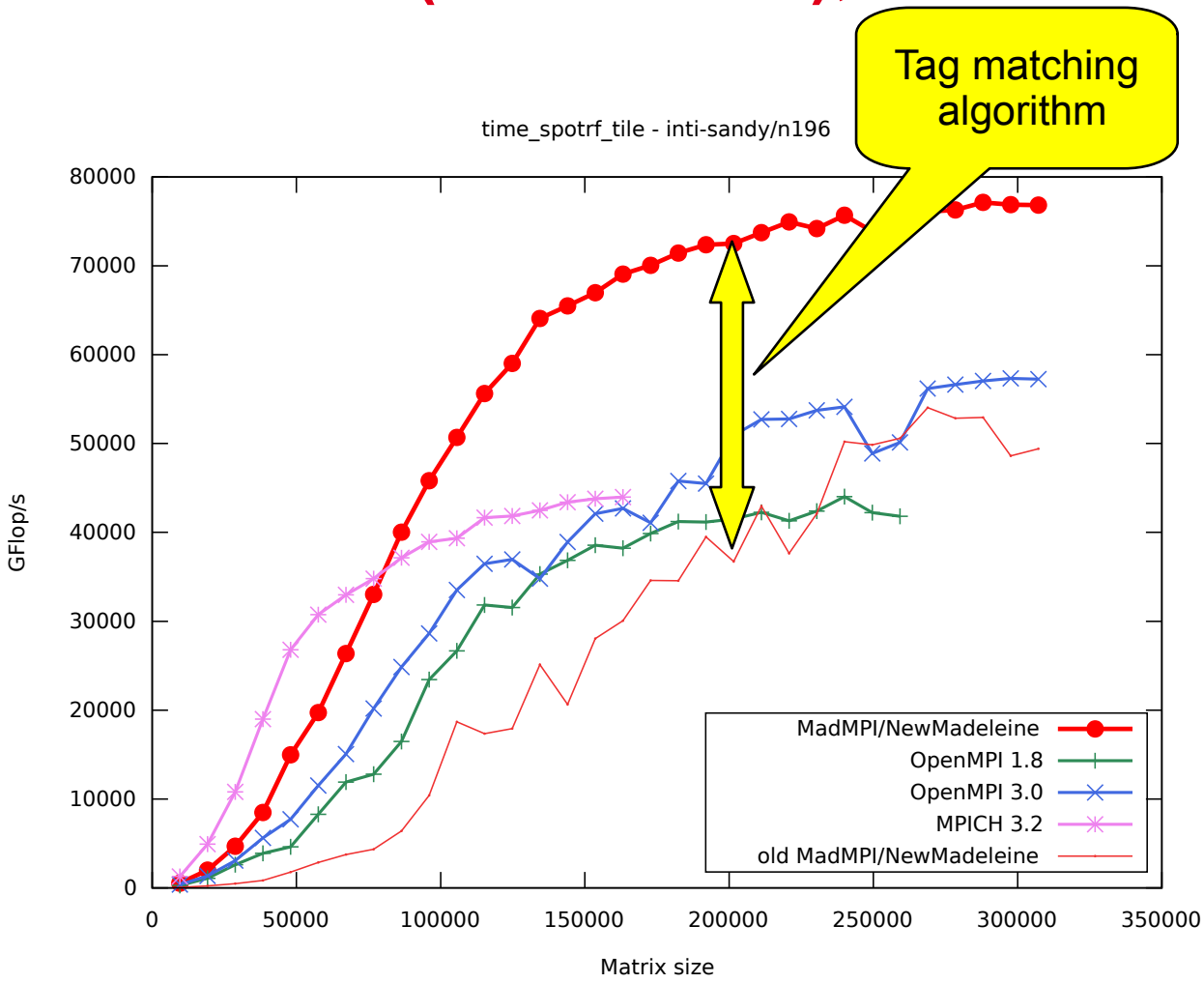
Cholesky – 4 nodes (64 cores), IB

- Graph: gigaflop/s
  - Higher is better
- Machine: `inti`, IB QDR
- Good performance for small & large matrix size
- Compared against previous NewMadeleine version, without constant-time matching
  - Really **check** the performance **impact of matching**
- **MPI library impacts global application performance**



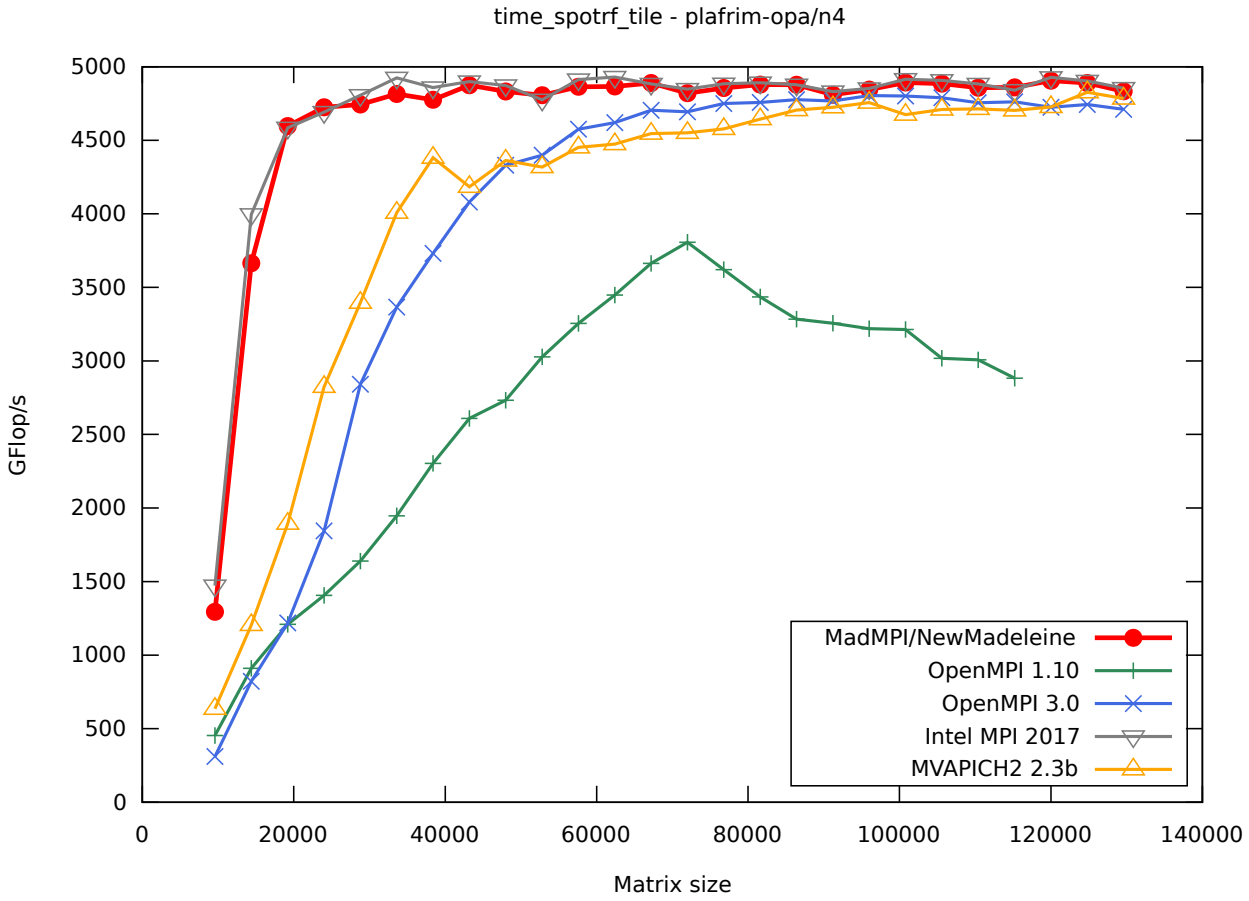
Cholesky – 196 nodes (3136 cores), IB

- Graph: gigaflop/s
  - Higher is better
- Machine: `inti`, IB QDR
- NewMadeleine reaches **+35% Gflop/s** than other MPI libraries
- **MPI library impacts global application performance**



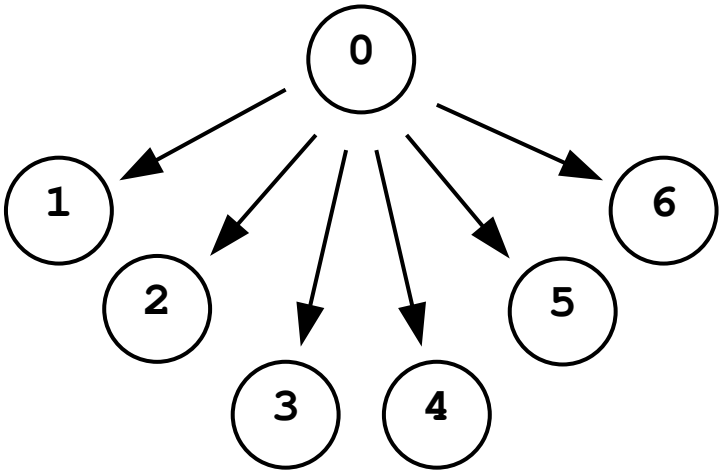
Cholesky – 4 nodes (96 cores), OmniPath

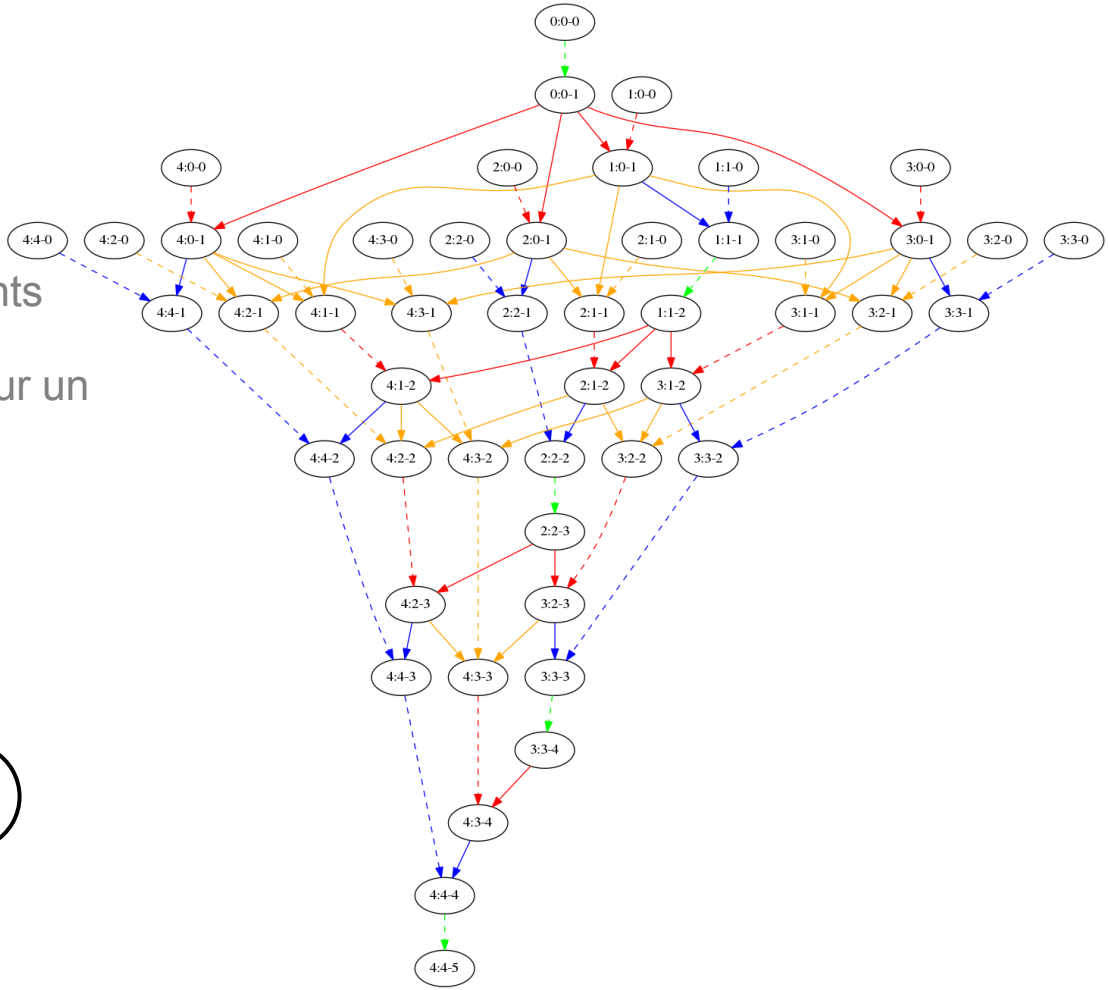
- Graph: gigaflop/s
  - Higher is better
- Machine: `plafrim`, OmniPath
- NewMadeleine reaches good performance for small & large matrix size



Collectives dans StarPU

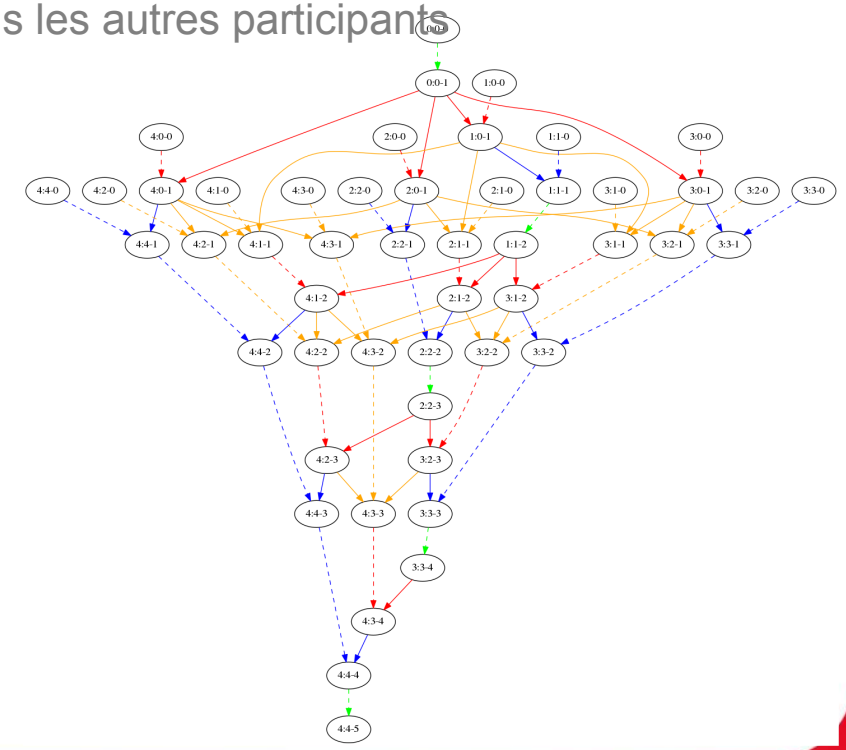
- Une donnée est dépendance de plusieurs tâches
  - sur plusieurs noeuds différents
  - = **multicast** (ou *broadcast* sur un sous-communicateur)





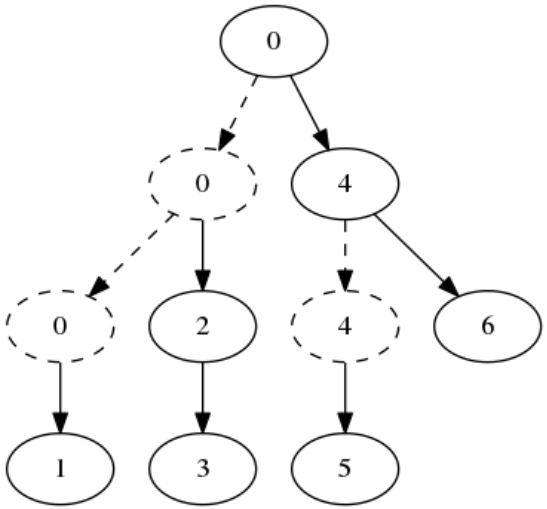
Collectives dans StarPU

- MPI_Bcast
  - Synchronisation : tous les noeuds participent **en même temps**
  - Tous les noeuds doivent **connaître** tous les autres participants
  - Créer un communicateur dédié
    = **barrière** sur le communicateur père
- Incompatible avec le modèle **asynchrone**
  & décentralisé de StarPU

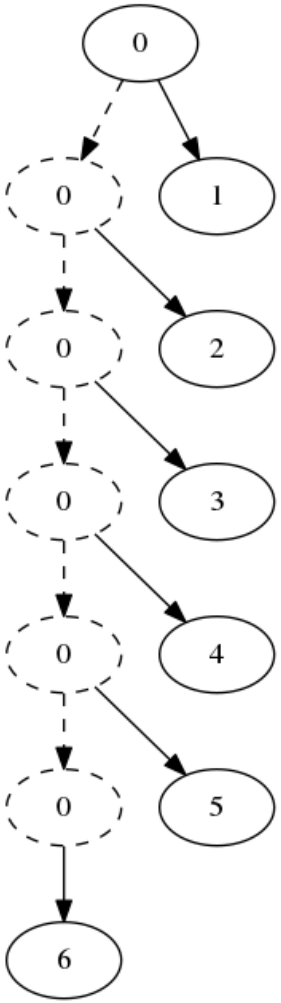


Collectives dans StarPU

- Algorithme naïf
  - La racine envoie à toutes les feuilles
  - Complexité **linéaire**
- Algorithme en arbre (binomial)
  - Complexité **logarithmique**
  - Tous les noeuds doivent **connaître les destinataires**



Arbre binomial



Algorithme naïf

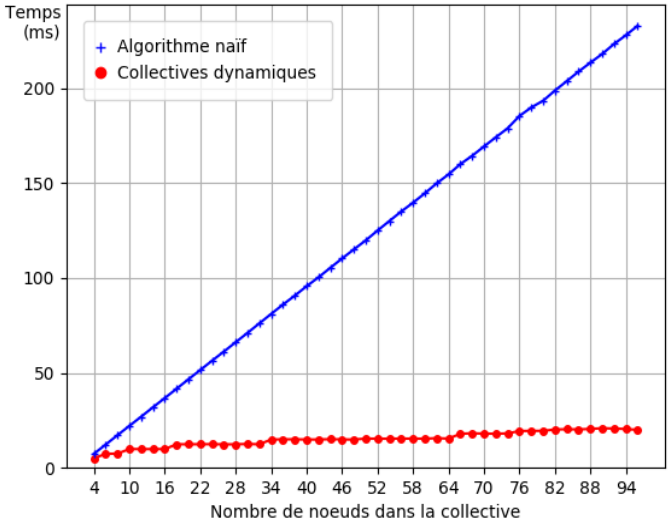
Temps

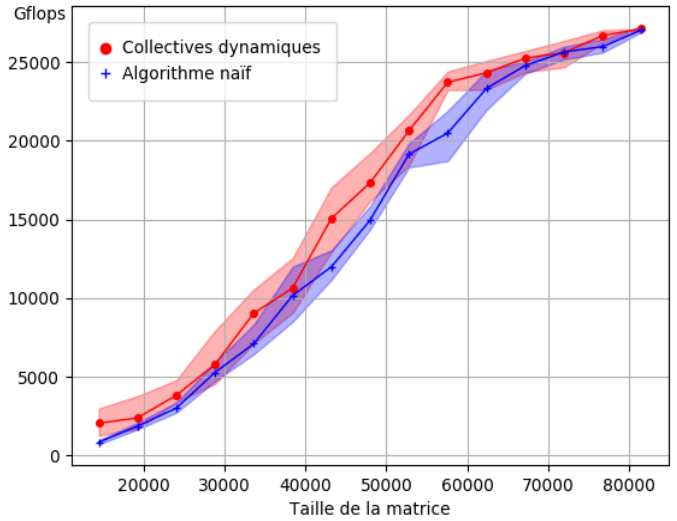
Collectives dynamiques

- **Détection** qu'une même donnée est à envoyer à plusieurs destinataires
- Algorithme de collective avec participation **passive** des noeuds intermédiaires
  - **Routage** à la source
    - Le message contient toutes les instructions de routage, les feuilles n'ont pas besoin de se connaître entre elles
  - **Messages actifs** NewMadeleine
    - Les noeuds n'ont pas besoin de savoir qu'ils vont recevoir une collective et non un simple send

Collectives dynamiques

- Benchmark sur machine *inti* (CEA)
- Microbenchmark
  - Performance similaire au MPI_Bcast()
- Benchmark Chameleon (Cholesky)
  - Sur 64 noeuds, collectives vers 7 noeuds maximum
  - Amélioration performance moyenne de 21%
- À confirmer sur de plus grands cas





3

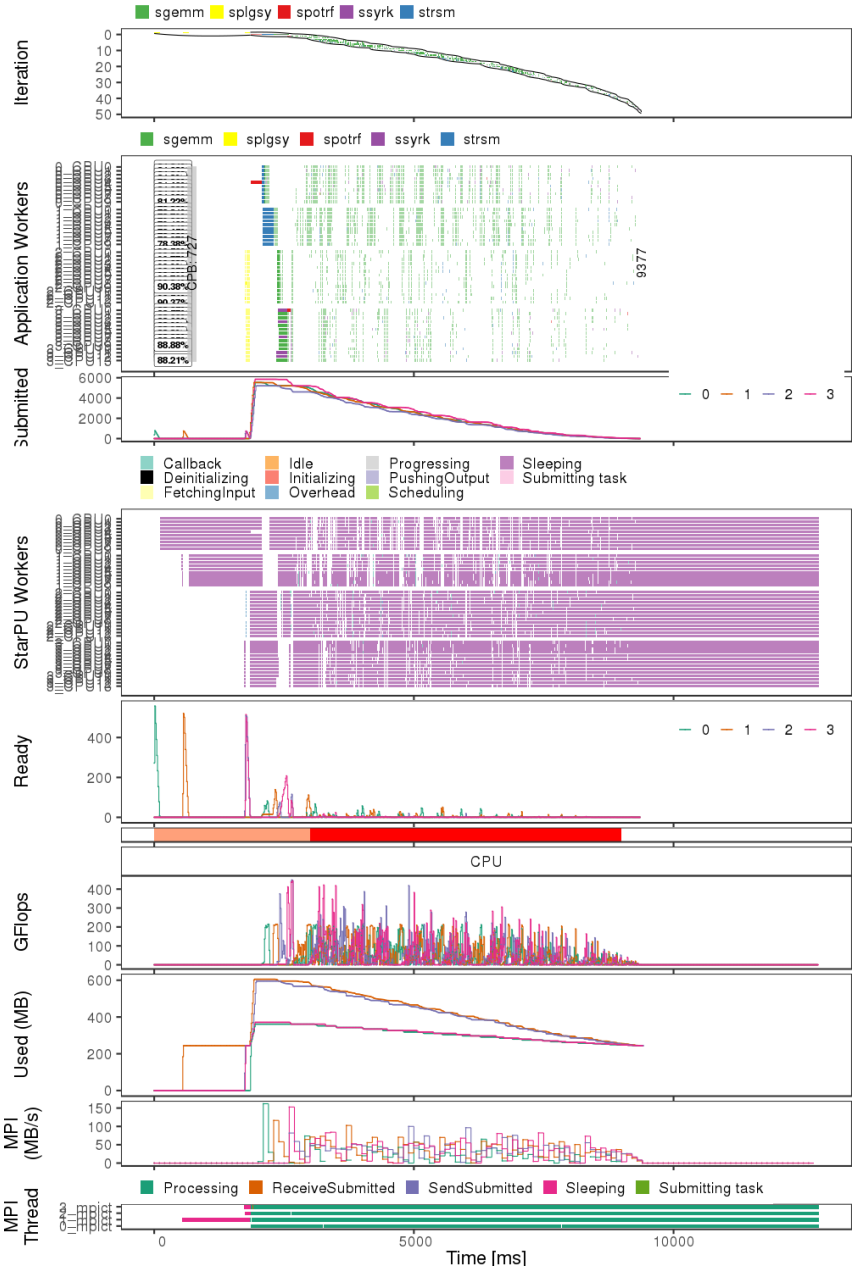
Work in progress

Positionnement

- ADT Gordon
  - Action de consolidation de la pile logicielle HPC de l'Inria Bordeaux
    - NewMadeleine (comms) + StarPU (tâches) + Chameleon (GEMM) + Diodon (appli)
    - Consolidation de chacune des bibliothèques
    - Intégration NewMadeleine + StarPU
  - Ingénieur : Adrien Guilbaud
- Projet région hpc-scalable-ecosystem
  - Action scientifique dans la pile logicielle HPC en Aquitaine
  - NewMadeleine + StarPU + Chameleon + autres
  - Thèse : Philippe Swartvagher
    - Diffusion dynamique dans StarPU/nmad
    - Problématique des communications dans les runtimes à tâches

Perspectives

- **Interactions** entre StarPU et NewMadeleine
  - StarPU connaît (un peu) le **futur**
- Localiser et comprendre les endroits où les communications réseau sont le goulet d'étranglement
- Identifier les **métriques** pertinentes ayant un impact sur la performance globale



Thank you

http://pm2.gforge.inria.fr/newmadeleine/