DE LA RECHERCHE À L'INDUSTRIE

Communication-Aware Task Scheduling Strategy in Hybrid MPI+OpenMP applications

IWOMP 2021 – 17th International Workshop on OpenMP – 16th September 2021

Romain PEREIRA [1,2] – Adrien ROUSSEL[1] - Patrick CARRIBAULT [1] – Thierry GAUTIER [2]

[1] Commissariat à l'énergie atomique et aux énergies alternatives (CEA, DAM, DIF, F-91297 Arpajon, France) - www.cea.fr

[2] Institut national de recherche en sciences et technologies du numérique (INRIA, Grenoble Rhône-Alpes, France) – www.inria.fr

Introduction (1/4)

Context

- HPC applications, **distributed** machines
  - Message Passing Interface (**MPI**)

Introduction (1/4)

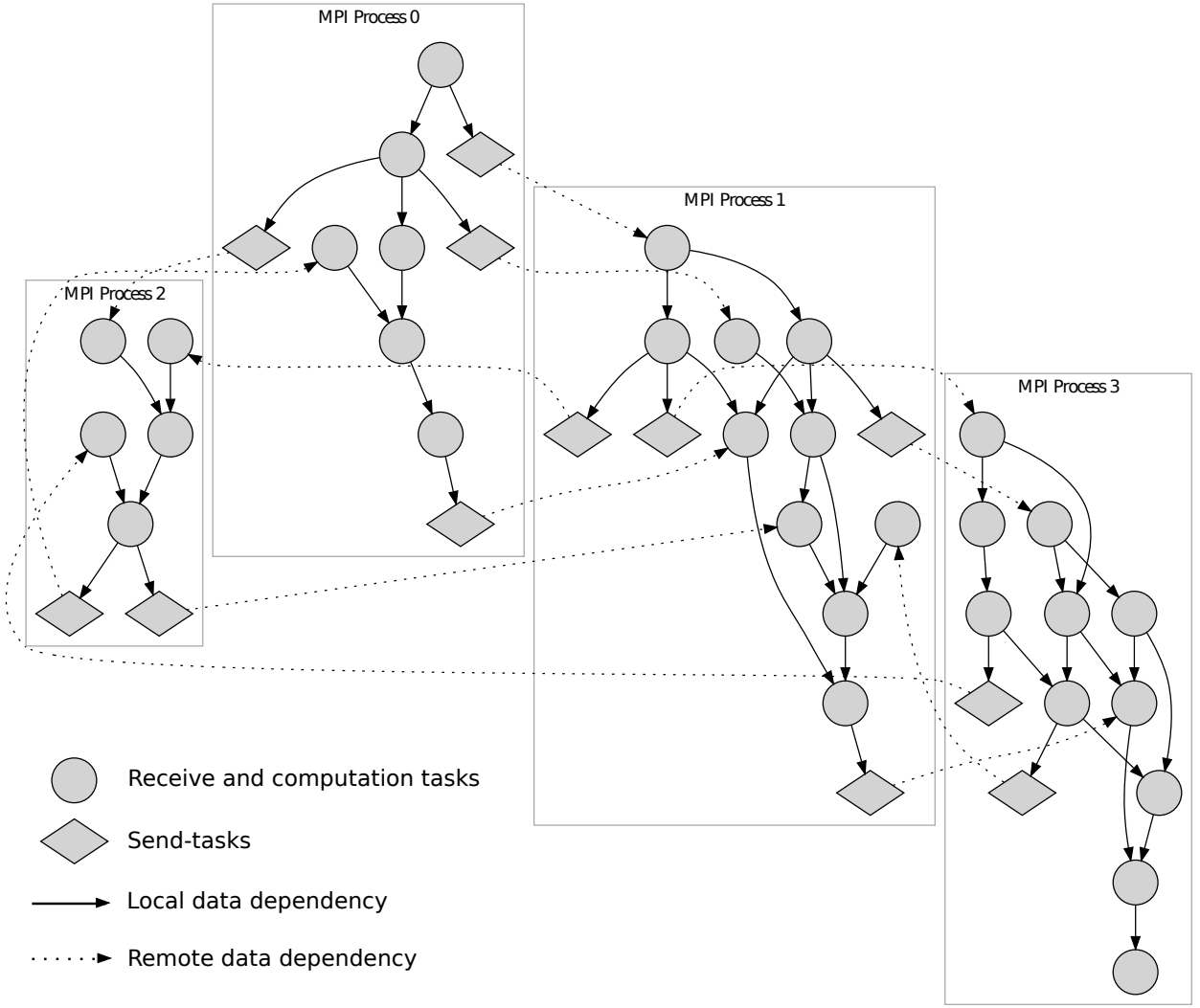
Context

- HPC applications, **distributed** machines
  - Message Passing Interface (**MPI**)

- On **compute nodes**, number of cores is increasing
  - **Task programming model** - efficiency and asynchronism
  - Introduced in OpenMP version 3.0, 2008

Context

- HPC applications, **distributed** machines
  - Message Passing Interface (**MPI**)
- On **compute nodes**, number of cores is increasing
  - **Task programming model** - efficiency and asynchronism
  - Introduced in OpenMP version 3.0, 2008
- **MPI+OpenMP(tasks)** applications
  - Distributed data-flow graph
  - Scheduler knowledge limited to local subgraphs

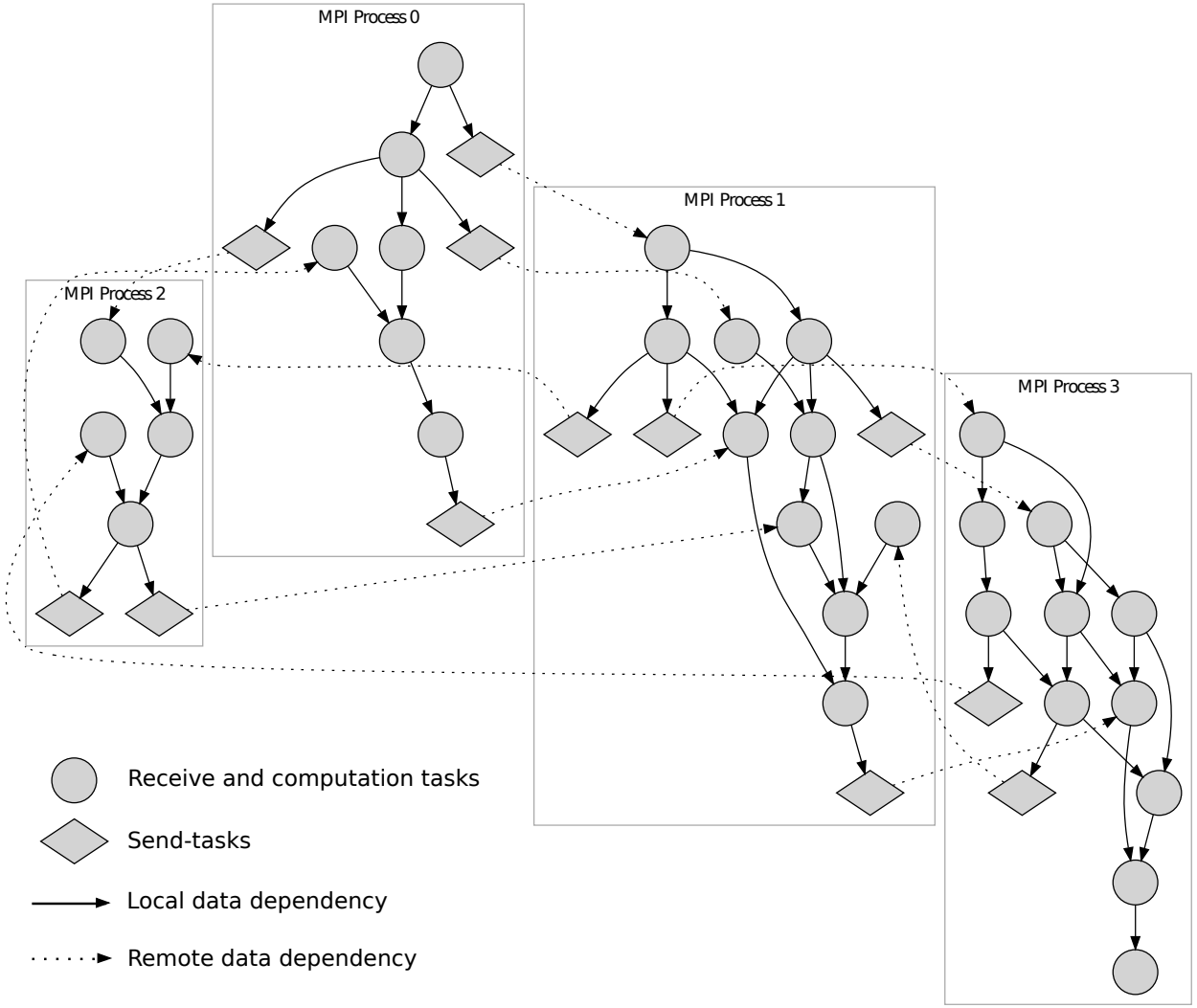


A data-dependency task graph distributed across 4 processes

Context

- HPC applications, **distributed** machines
  - Message Passing Interface (**MPI**)
- On **compute nodes**, number of cores is increasing
  - **Task programming model** - efficiency and asynchronism
  - Introduced in OpenMP version 3.0, 2008
- **MPI+OpenMP(tasks)** applications
  - Distributed data-flow graph
  - Scheduler knowledge limited to local subgraphs

➡ **Scheduling** issues

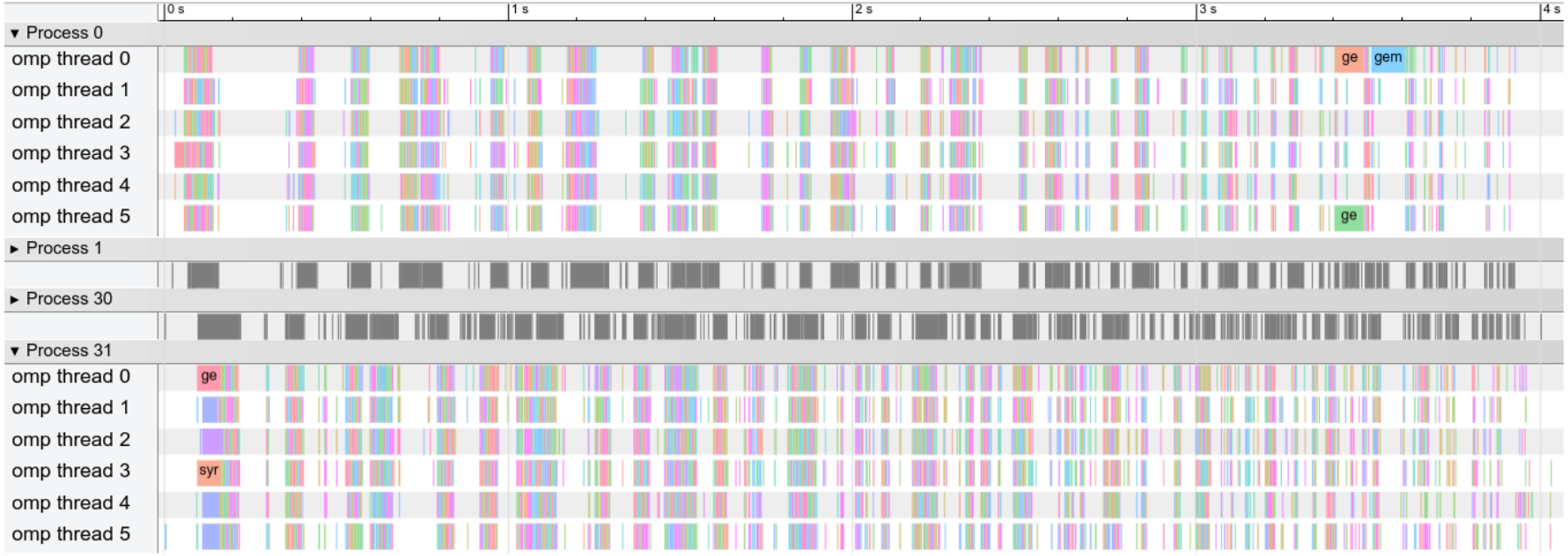


A data-dependency task graph distributed across 4 processes

Motivation – scheduling issues

- FIFO task scheduling
- Task-based Cholesky matrix factorization (MKL) **[TASKYIELD, CHOLESKY]**

- Skylake nodes
  - Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
  - 2 sockets (NUMA) – 24 cores per socket



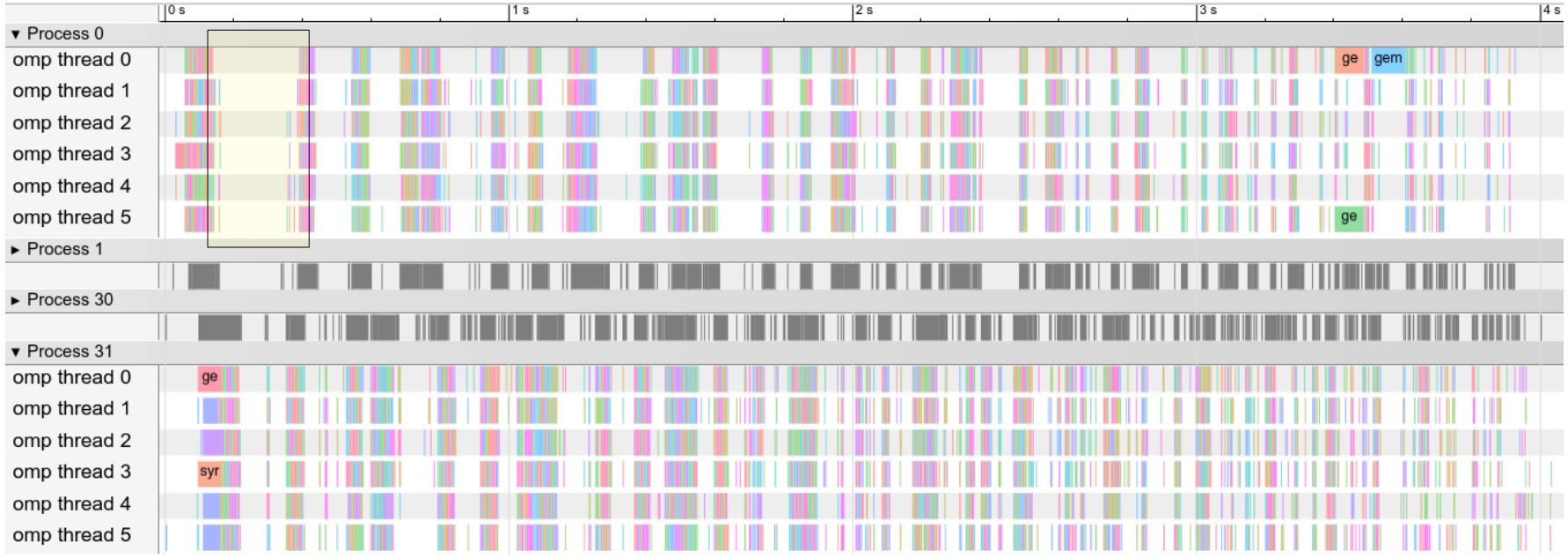
Task scheduling on a matrix of size 32,768 with blocks of size 512 on 4 Skykake nodes with 8 processes per nodes, of 6 threads each

[TASKYIELD] Joseph Schuchart, Keisuke Tsugane, José Gracia, Mitsuhisa Sato. (IWOMP 2018). The Impact of Taskyield on the Design of Tasks Communicating Through MPI
[CHOLESKY] – Cholesky factorization - https://gitlab.inria.fr/ropereir/iwomp2021 - forked from **[TASKYIELD]**

Motivation – scheduling issues

- FIFO task scheduling
- Task-based Cholesky matrix factorization (MKL) **[TASKYIELD, CHOLESKY]**

- Skylake nodes
  - Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
  - 2 sockets (NUMA) – 24 cores per socket



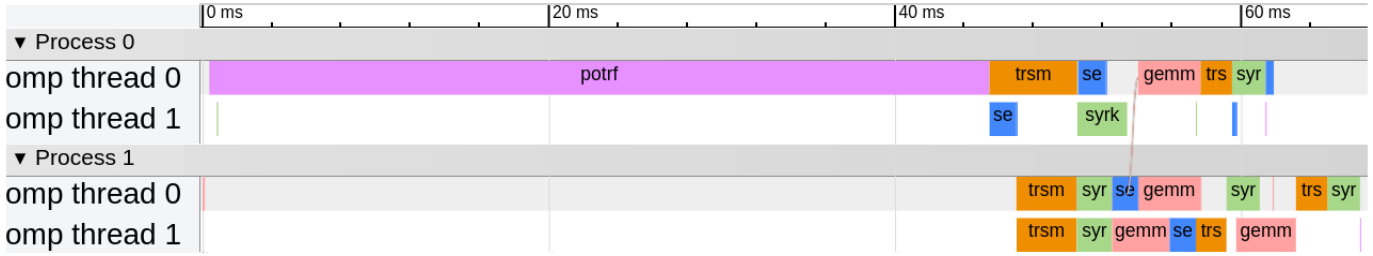
Task scheduling on a matrix of size 32,768 with blocks of size 512 on 4 Skykake nodes with 8 processes per nodes, of 6 threads each

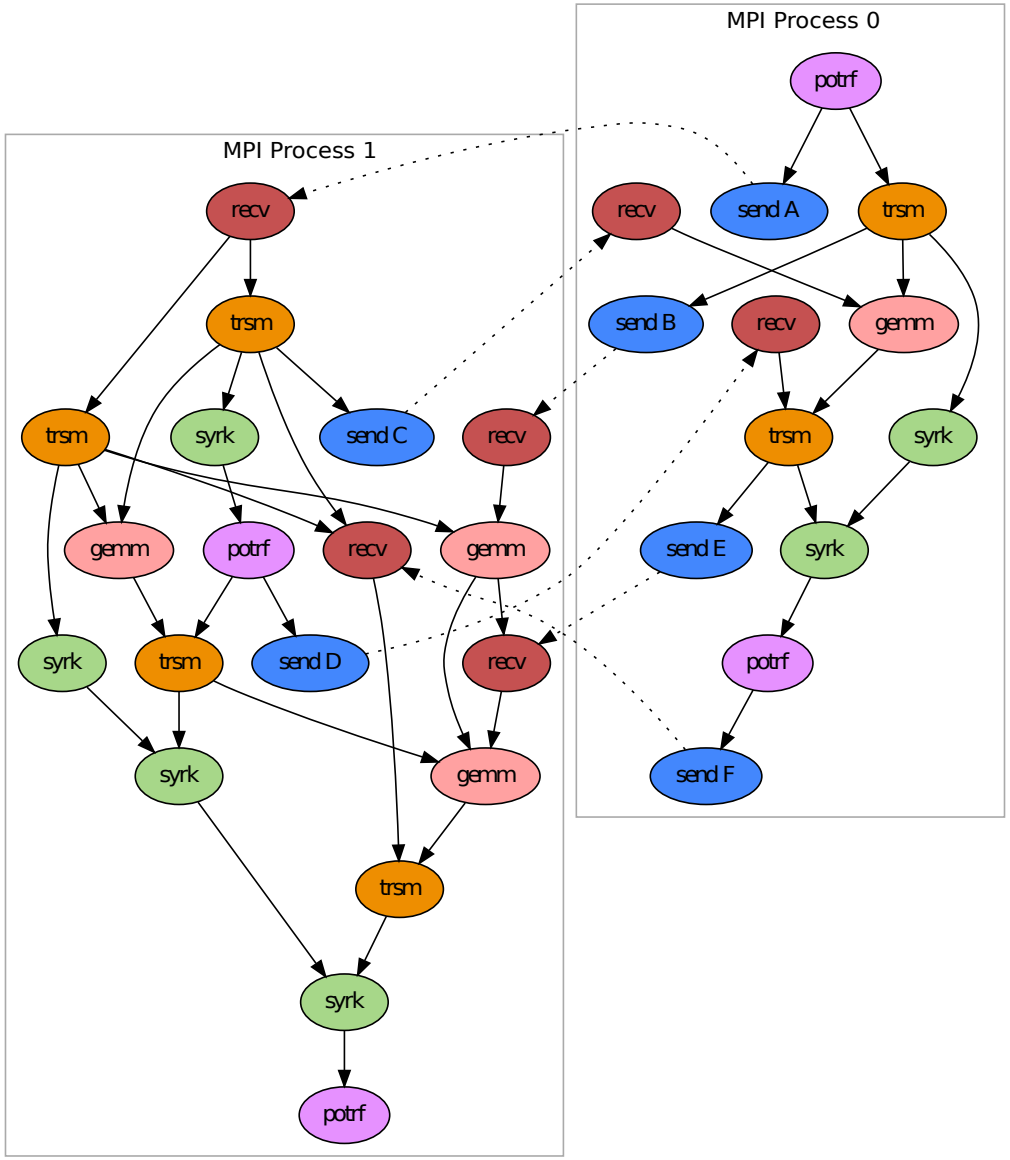
➡ Can we reduce **idle** periods with a better scheduling policy?

[TASKYIELD] Joseph Schuchart, Keisuke Tsugane, José Gracia, Mitsuhisa Sato. (IWOMP 2018). The Impact of Taskyield on the Design of Tasks Communicating Through MPI
[CHOLESKY] – Cholesky factorization - https://gitlab.inria.fr/ropereir/iwomp2021 - forked from **[TASKYIELD]**

Motivation - a minimal case

- Cholesky Factorization
  - Matrix of size 2,048 with blocks of size 512
  - 2 processes with 2 threads

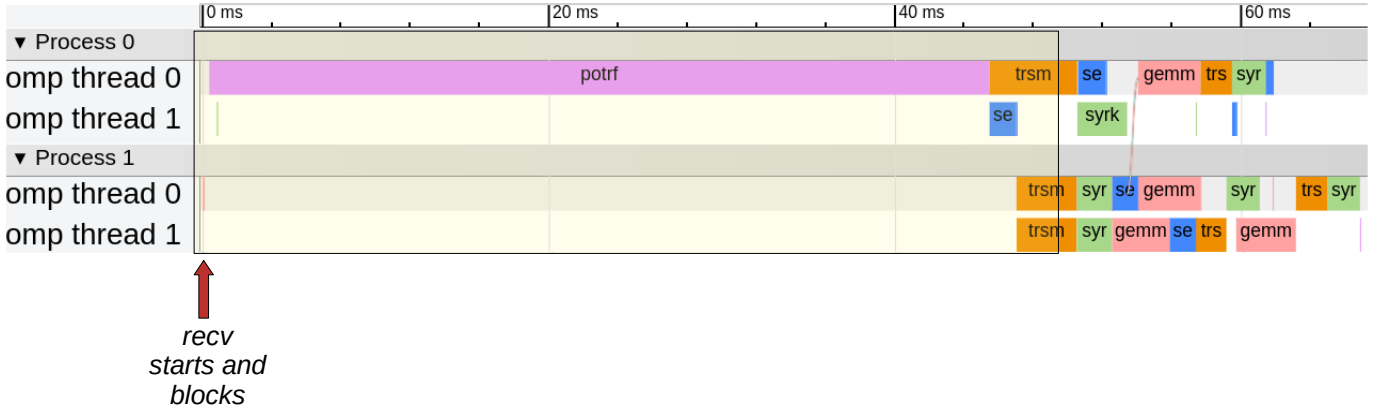




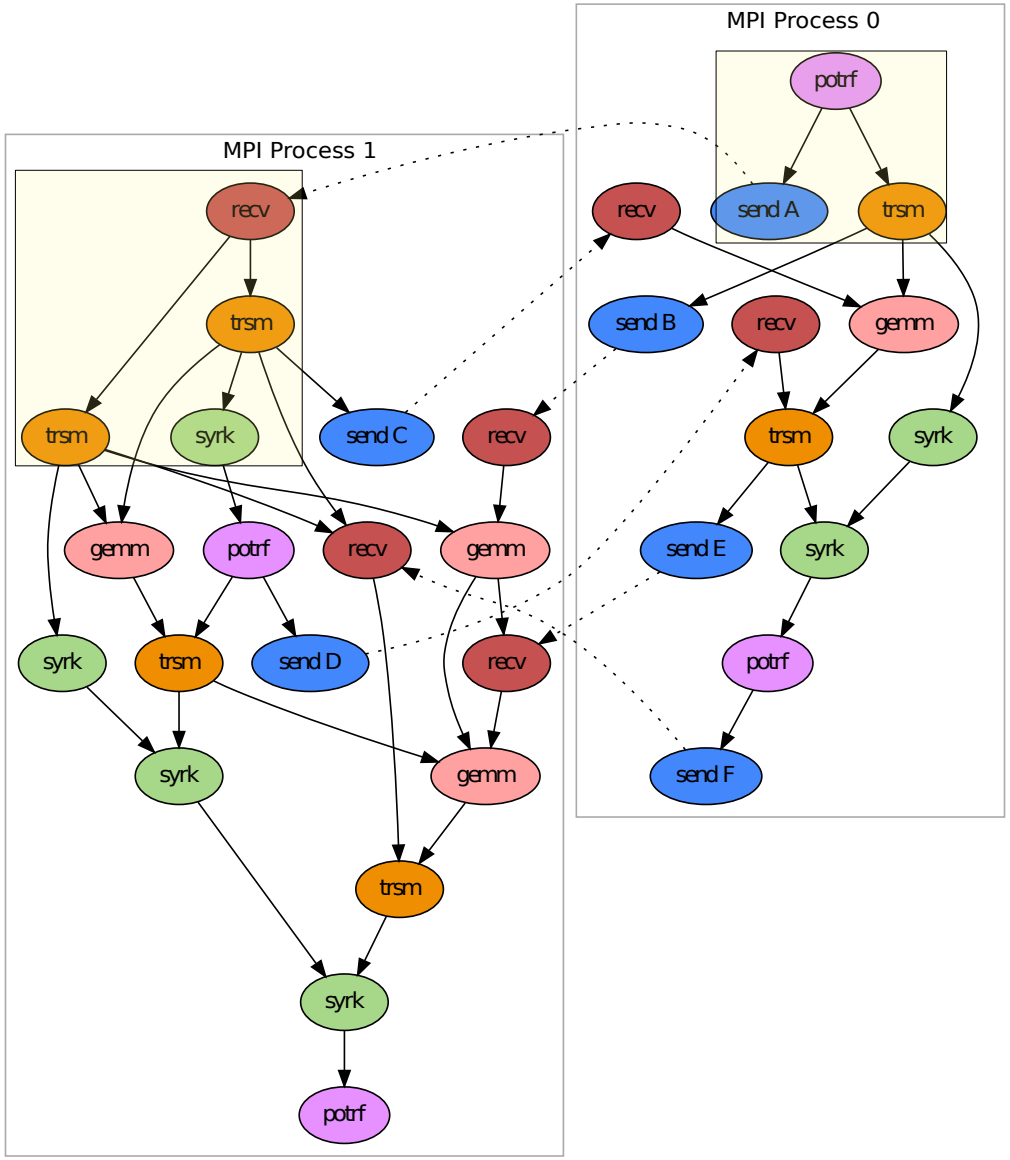
Data-dependency graph of the minimal case

Motivation - a minimal case

- Cholesky Factorization
  - Matrix of size 2,048 with blocks of size 512
  - 2 processes with 2 threads



Initially, **low parallelism** is expressed → no possible gain

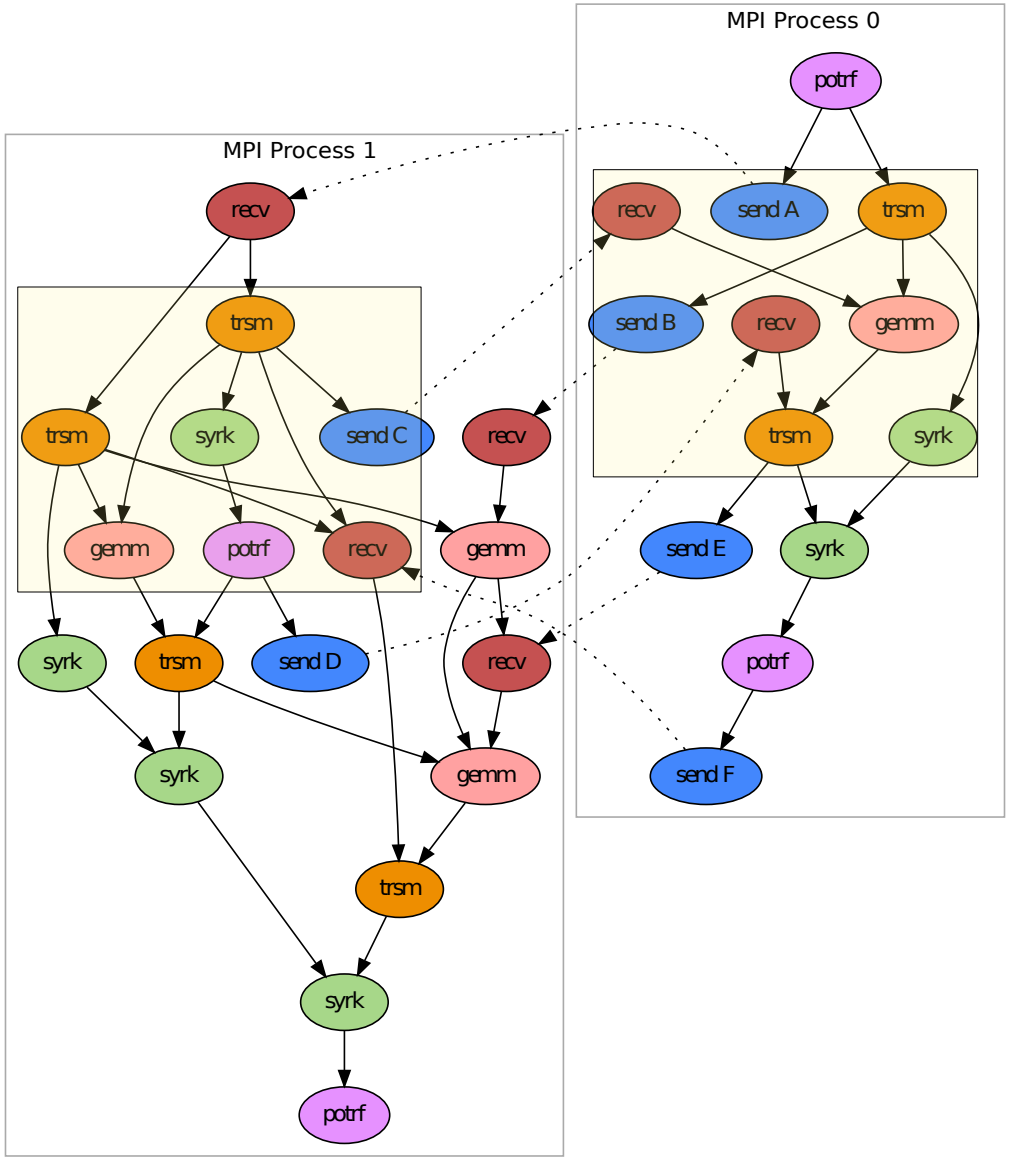


Data-dependency graph of the minimal case

Motivation - a minimal case

- Cholesky Factorization
  - Matrix of size 2,048 with blocks of size 512
  - 2 processes with 2 threads

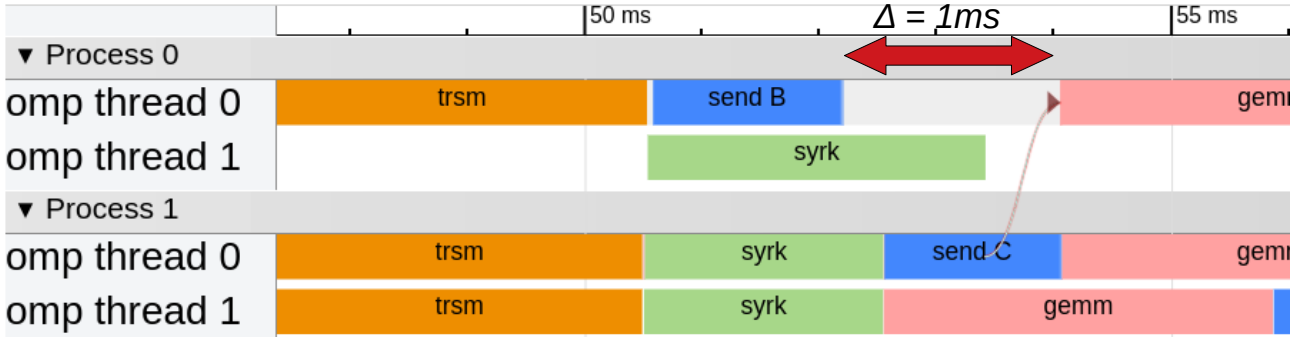
Idle time due to **scheduling decisions**



Data-dependency graph of the minimal case

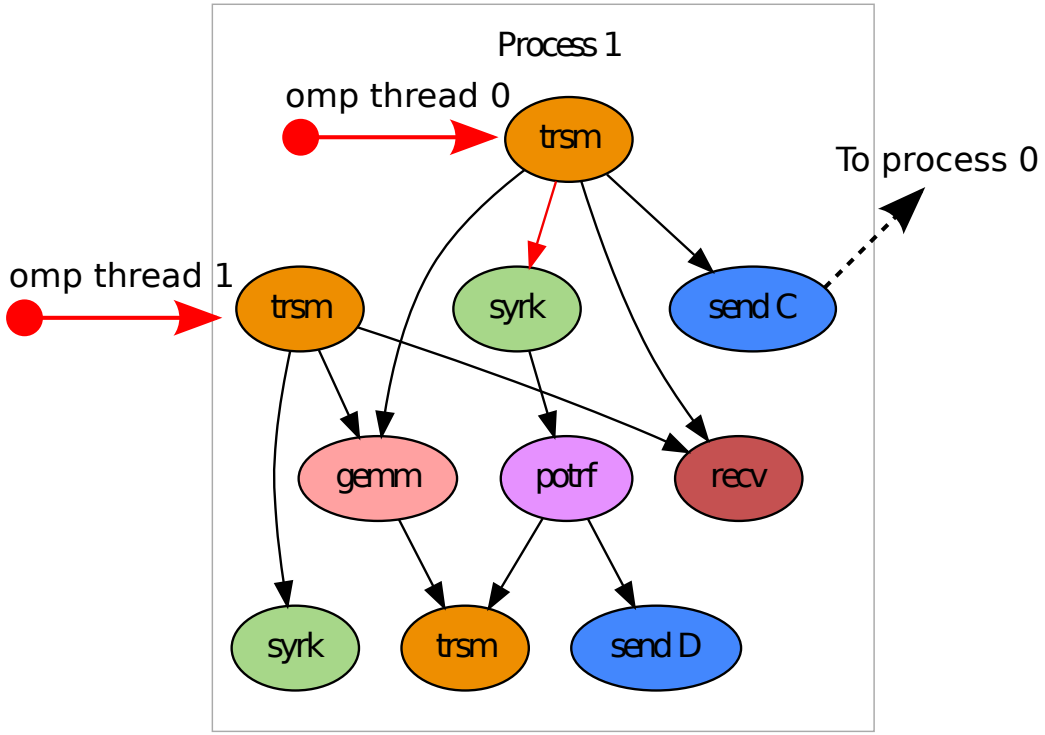
Motivation - a minimal case

- Cholesky Factorization
  - Matrix of size 2,048 with blocks of size 512
  - 2 processes with 2 threads



FIFO task scheduling on the minimal case

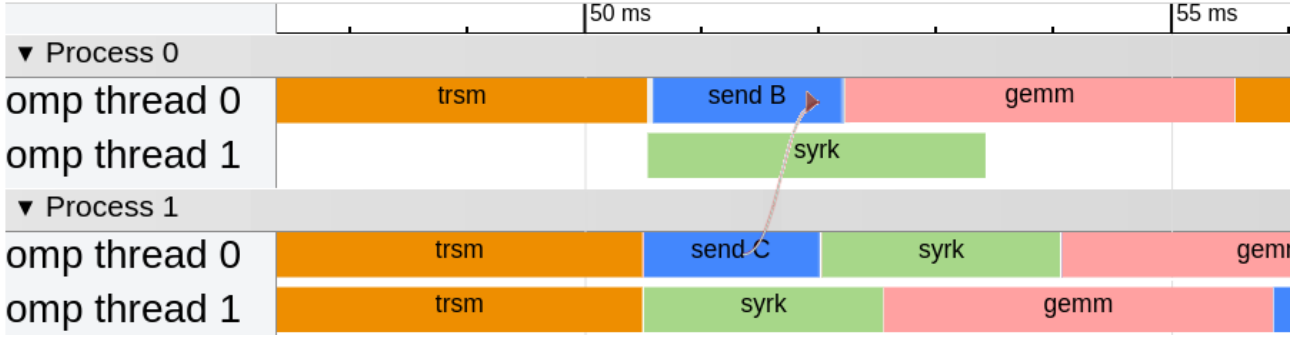
We **zoomed** in the idle period



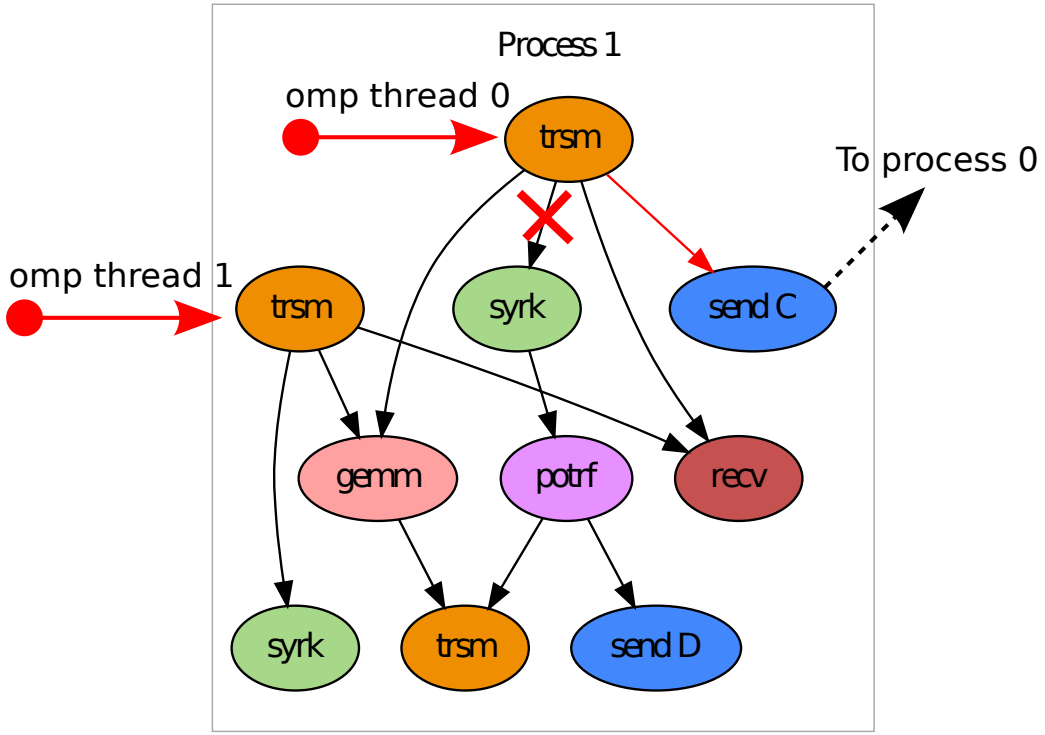
Subgraph of process 1 on the minimal case

Motivation - a minimal case

- Cholesky Factorization
  - Matrix of size 2,048 with blocks of size 512
  - 2 processes with 2 threads



FIFO task scheduling on the minimal case



Subgraph of process 1 on the minimal case

Problem

➡ Wrong scheduling decisions taken locally make remote threads **idle**

Related work

- MPI+OpenMP interoperability issues
  - Loss of threads (or even deadlocks) when nesting MPI blocking operations within OpenMP tasks – **[QCD benchmark]**

```
1    # pragma omp task depends(in: data)
2    {
3        […]
4        MPI_Send(data, …) ;
5        […]
6     }
```

Hybrid MPI+OpenMP(tasks) minimal code example

[QCD benchmark] - Meadows, Larry & Ishikawa, Ken-ichi. (2017). OpenMP Tasking and MPI in a Lattice QCD Benchmark.

Related work

- MPI+OpenMP interoperability issues
  - Loss of threads (or even deadlocks) when nesting MPI blocking operations within OpenMP tasks – **[QCD benchmark]**
- Several solutions - **[TASKYIELD] – [TAMPI] – [MPI-DETACH]**
  - Task switch to overlap waiting time
  - Enables working MPI+OpenMP(tasks) applications

```
1    # pragma omp task depends(in: data)
2    {
3        […]
4        MPI_Send(data, …) ;
5        […]
6     }
```

Hybrid MPI+OpenMP(tasks) minimal code example

[QCD benchmark] - Meadows, Larry & Ishikawa, Ken-ichi. (2017). OpenMP Tasking and MPI in a Lattice QCD Benchmark.

[TASKYIELD] - Joseph Schuchart, Keisuke Tsugane, José Gracia, Mitsuhisa Sato. (IWOMP 2018). The Impact of Taskyield on the Design of Tasks Communicating Through MPI

[TAMPI] - Kevin Sala, Jorge Bellón, Pau Farré, Xavier Teruel, Josep M. Perez, Antonio J. Peña, Daniel Holmes, Vicenç Beltran, and Jesus Labarta. 2018. Improving the Interoperability between MPI and Task-Based Programming Models. In Proceedings of the 25th European MPI Users' Group Meeting (EuroMPI'18). Association for Computing Machinery, New York, NY, USA, Article 6, 1–11.

[MPI-DETACH] – Protze, J., Hermanns, M.A., Demiralp, A., Müller, M.S., Kuhlen, T. : MPI detach – asynchronous local completion. In : 27th European MPI Users' Group Meeting. p. 71-80. EuroMPI/USA '20, Association for Computing Machinery, New York, NY, USA (2020)

Introduction (4/4)

Related work

- MPI+OpenMP interoperability issues
  - Loss of threads (or even deadlocks) when nesting MPI blocking operations within OpenMP tasks – **[QCD benchmark]**
- Several solutions - **[TASKYIELD] – [TAMPI] – [MPI-DETACH]**
  - Task switch to overlap waiting time
  - Enables working MPI+OpenMP(tasks) applications

➡ Scheduling issues remains

```
1    # pragma omp task depends(in: data)
2    {
3        […]
4        MPI_Send(data, …) ;
5        […]
6     }
```

Hybrid MPI+OpenMP(tasks) minimal code example

Contribution

- **Task scheduling strategy for hybrid MPI+OpenMP(tasks) applications**
  - Task re-ordering
  - Favoring local send operations to reduce remote idle times

[QCD benchmark] - Meadows, Larry & Ishikawa, Ken-ichi. (2017). OpenMP Tasking and MPI in a Lattice QCD Benchmark.

[TASKYIELD] - Joseph Schuchart, Keisuke Tsugane, José Gracia, Mitsuhisa Sato. (IWOMP 2018). The Impact of Taskyield on the Design of Tasks Communicating Through MPI

[TAMPI] - Kevin Sala, Jorge Bellón, Pau Farré, Xavier Teruel, Josep M. Perez, Antonio J. Peña, Daniel Holmes, Vicenç Beltran, and Jesus Labarta. 2018. Improving the Interoperability between MPI and Task-Based Programming Models. In Proceedings of the 25th European MPI Users' Group Meeting (EuroMPI'18). Association for Computing Machinery, New York, NY, USA, Article 6, 1–11.

[MPI-DETACH] – Protze, J., Hermanns, M.A., Demiralp, A., Müller, M.S., Kuhlen, T. : MPI detach – asynchronous local completion. In : 27th European MPI Users' Group Meeting. p. 71-80. EuroMPI/USA '20, Association for Computing Machinery, New York, NY, USA (2020)

Contents

Communication-Aware Task Re-ordering Strategy

- **Priority Policies**
- Implementation

Communication-Aware Task Re-ordering Strategy (1/5)

Task Ordering

- Ready tasks with the highest priority first
- OpenMP **priority hint** clause

Communication-Aware Task Re-ordering Strategy (1/5)

Task Ordering

- Ready tasks with the highest priority first
- OpenMP **priority hint** clause

Manual Approach (MA1)

- Hint to **1** on **send-tasks**
- No hint on other tasks (0 by default)

```
1   # pragma omp task depends(in: data) priority(1)
2   {
3       […]
4       MPI_Send(data, …) ;
5       […]
6    }
```

Hybrid MPI+OpenMP(tasks) minimal code example

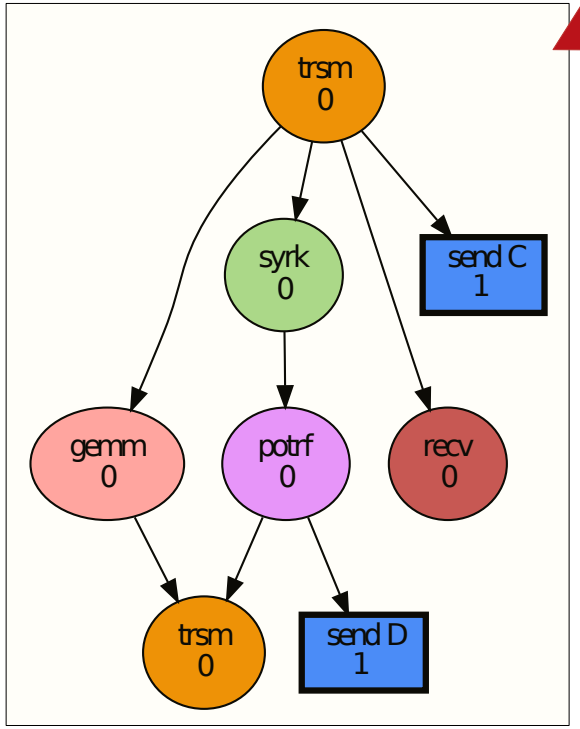
Communication-Aware Task Re-ordering Strategy (1/5)

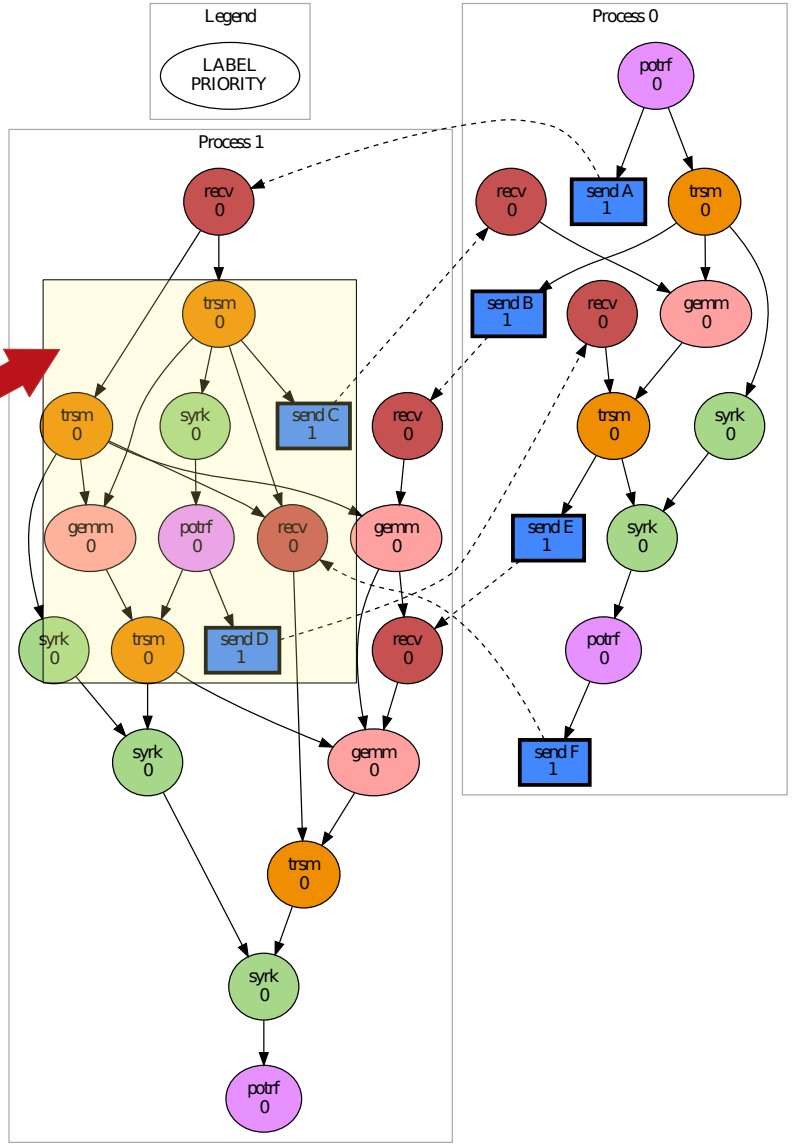
Task Ordering

- Ready tasks with the highest priority first
- OpenMP **priority hint** clause

Manual Approach (MA1)

- Hint to **1** on **send-tasks**
- No hint on other tasks (0 by default)





Data-dependency graph and MA1 priorities

Communication-Aware Task Re-ordering Strategy (1/5)

Task Ordering

- Ready tasks with the highest priority first
- OpenMP **priority hint** clause

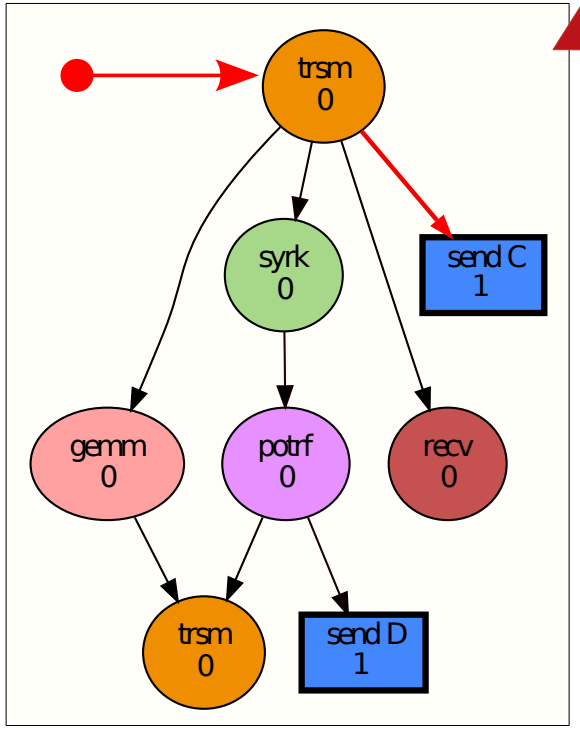
Manual Approach (MA1)

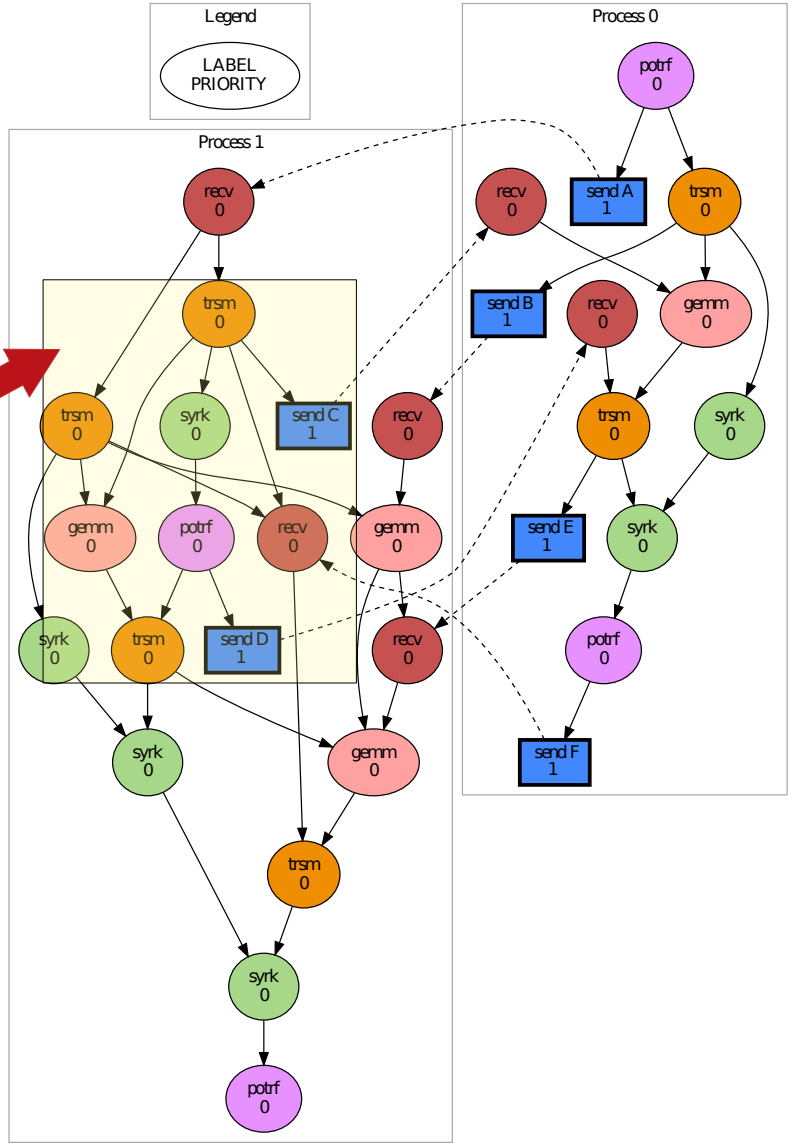
- Hint to **1** on **send-tasks**
- No hint on other tasks (0 by default)

Advantages

- Whenever a **send** is ready, it is scheduled

→ Once **trsm** is completed, **send C** is elected





Data-dependency graph and MA1 priorities

Communication-Aware Task Re-ordering Strategy (1/5)

Task Ordering

- Ready tasks with the highest priority first
- OpenMP **priority hint** clause

Manual Approach (MA1)

- Hint to **1** on **send-tasks**
- No hint on other tasks (0 by default)

Advantages

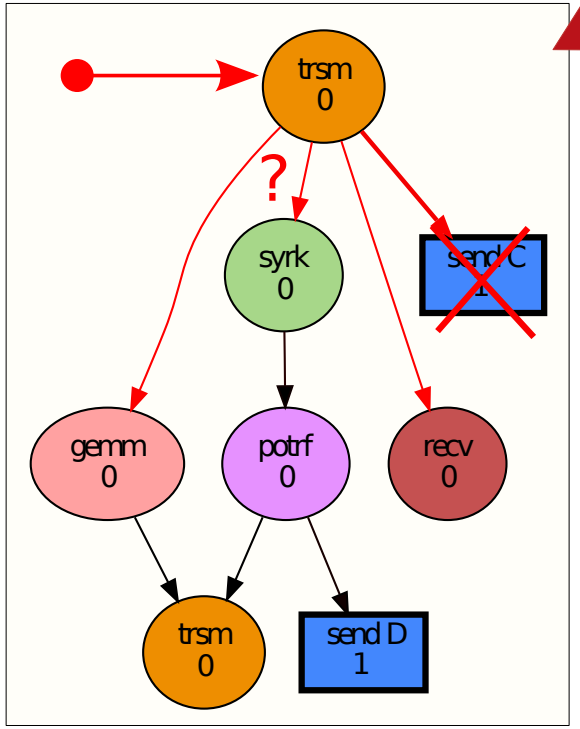
- Whenever a **send** is ready, it is scheduled

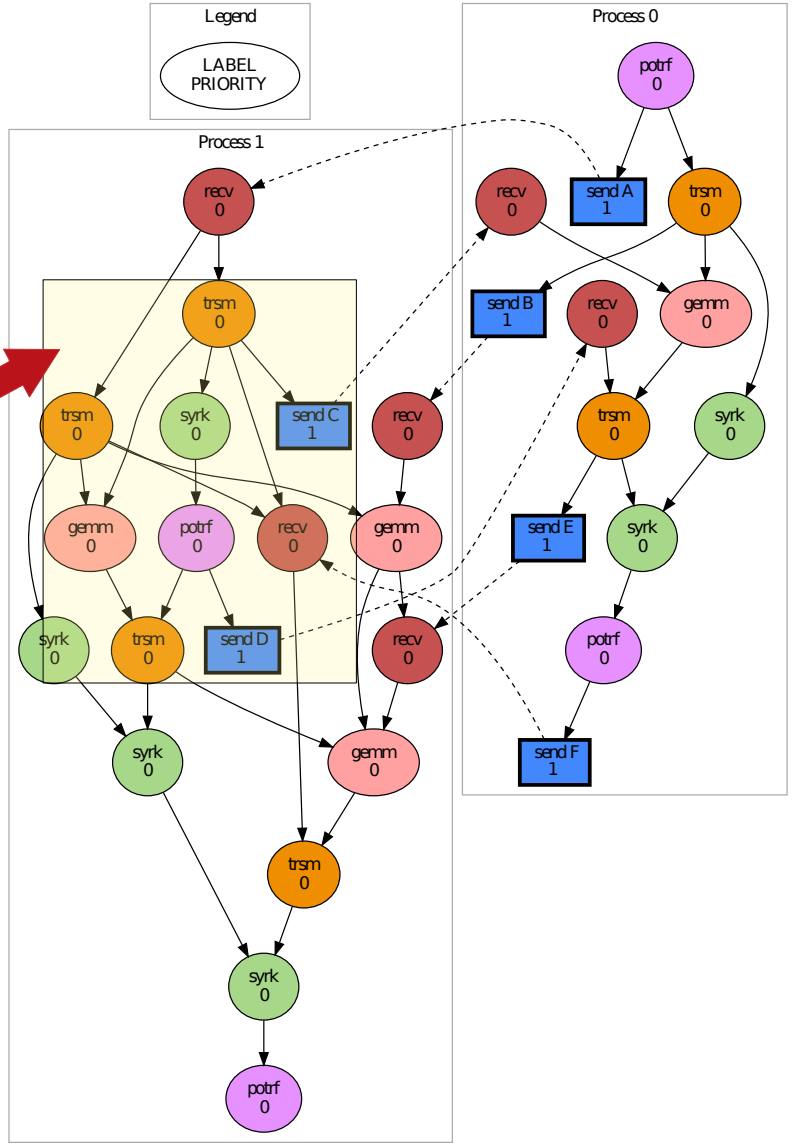
→ Once **trsm** is completed, **send C** is elected

Drawbacks

- No informations on send-tasks' predecessors

After **send C**: **syrk**, **gemm** or **recv**?





Data-dependency graph and MA1 priorities

Communication-Aware Task Re-ordering Strategy (1/5)

Task Ordering

- Ready tasks with the highest priority first
- OpenMP **priority hint** clause

Manual Approach (MA1)

- Hint to **1** on **send-tasks**
- No hint on other tasks (0 by default)

Advantages

- Whenever a **send** is ready, it is scheduled

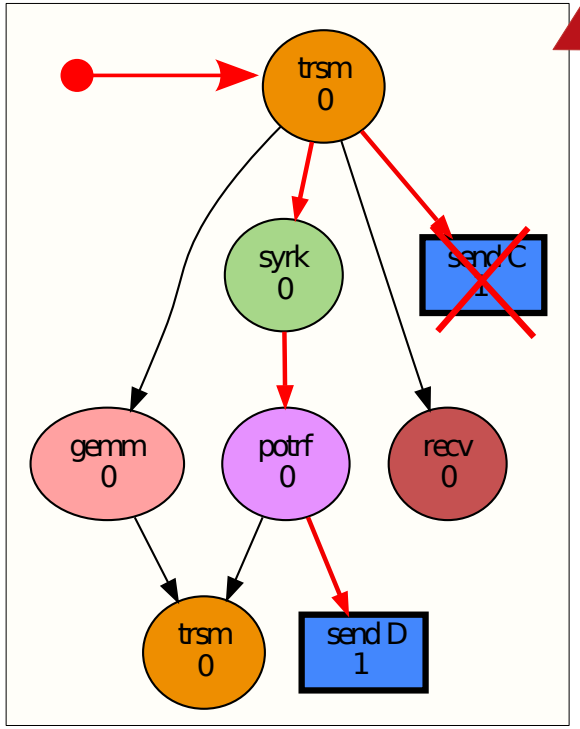
→ Once **trsm** is completed, **send C** is elected

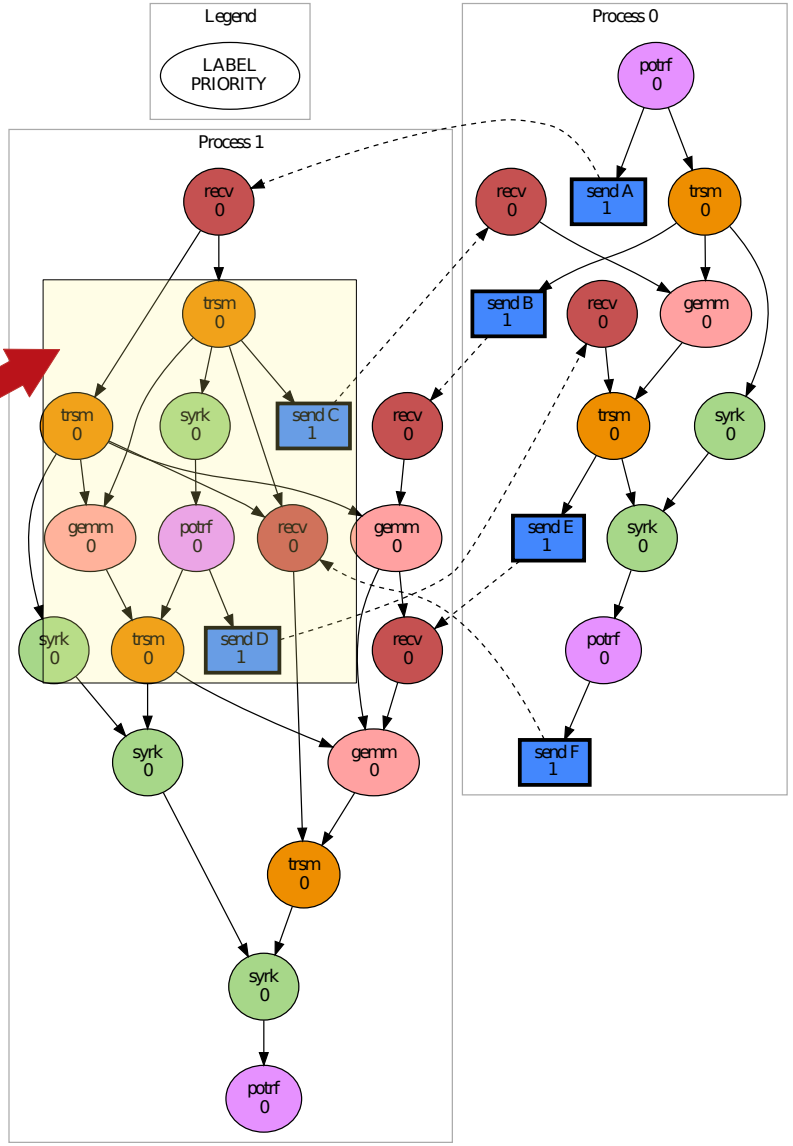
Drawbacks

- No informations on send-tasks' predecessors

After **send C**: **syrk**, **gemm** or **recv**?

→ **syrk** to unlock **potrf** and **send D**





Data-dependency graph and MA1 priorities

Communication-Aware Task Re-ordering Strategy (1/5)

Task Ordering

- Ready tasks with the highest priority first
- OpenMP **priority hint** clause

Manual Approach (MA1)

- Hint to **1** on **send-tasks**
- No hint on other tasks (0 by default)

Advantages

- Whenever a **send** is ready, it is scheduled

→ Once **trsm** is completed, **send C** is elected

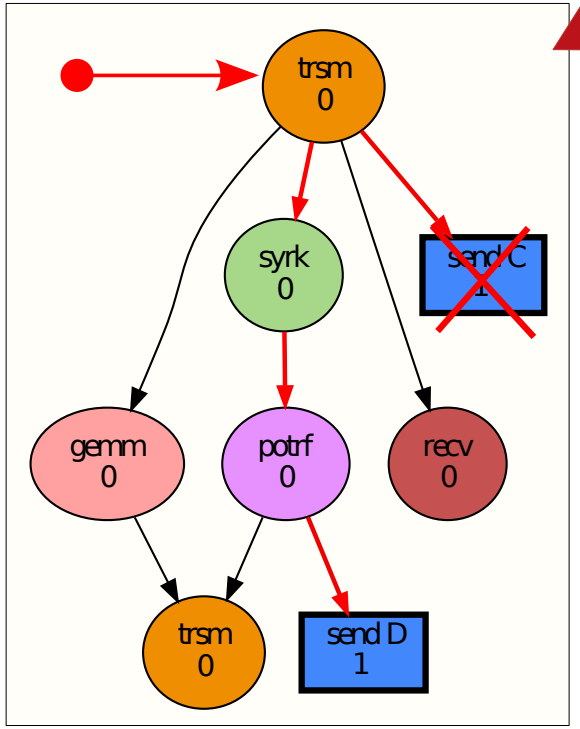
Drawbacks

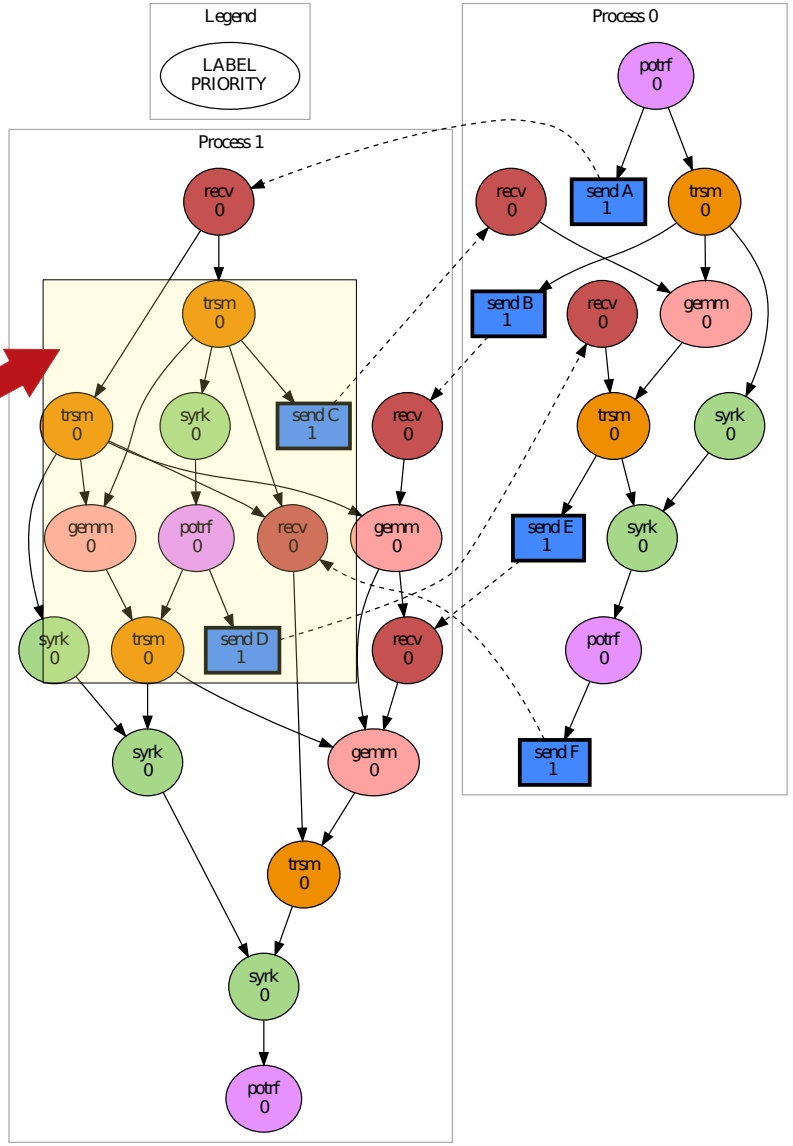
- No informations on send-tasks' predecessors

After **send C**: **syrk**, **gemm** or **recv**?

→ **syrk** to unlock **potrf** and **send D**

➡ Schedule send-task **predecessors**

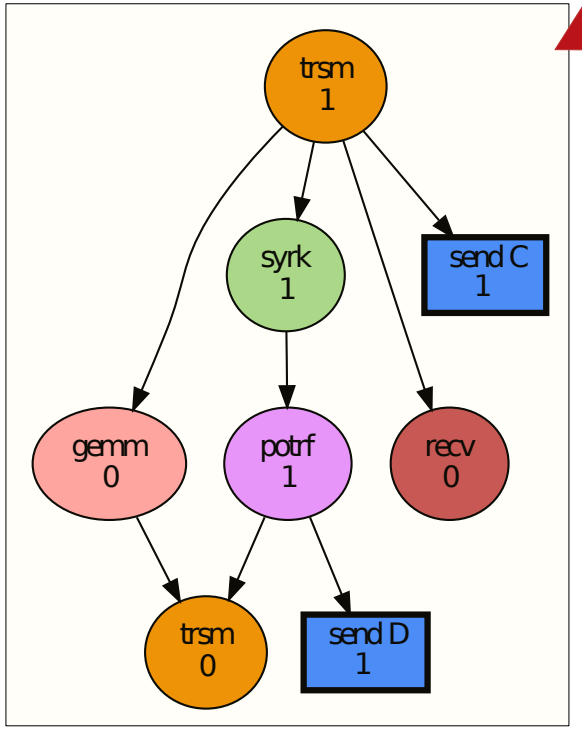


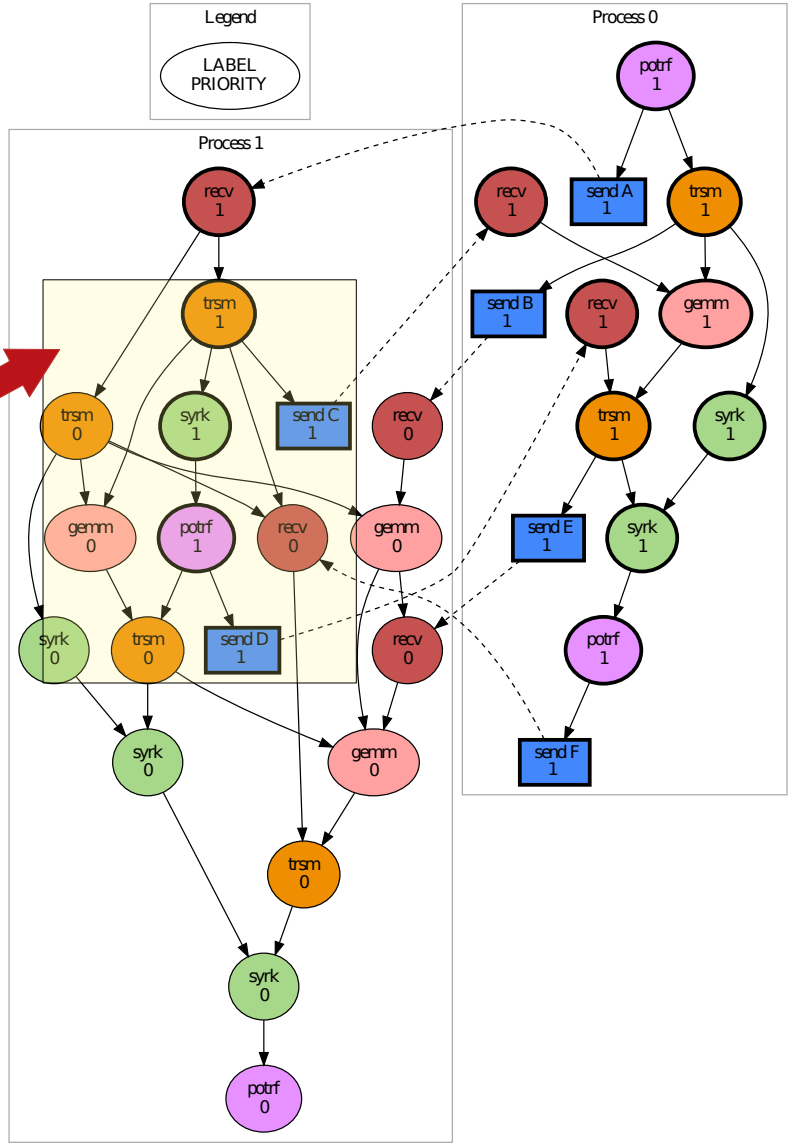


Data-dependency graph and MA1 priorities

Manual Approach (MA2)

- Hint to **1** on **send-tasks** and on their **predecessors** recursively
- No hint on other tasks (0 by default)





Data-dependency graph and MA2 priorities

Communication-Aware Task Re-ordering Strategy (2/5)

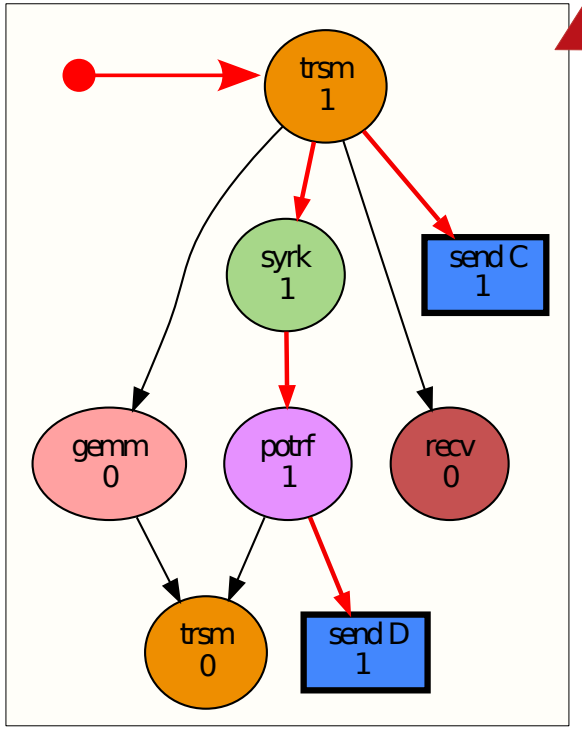
Manual Approach (MA2)

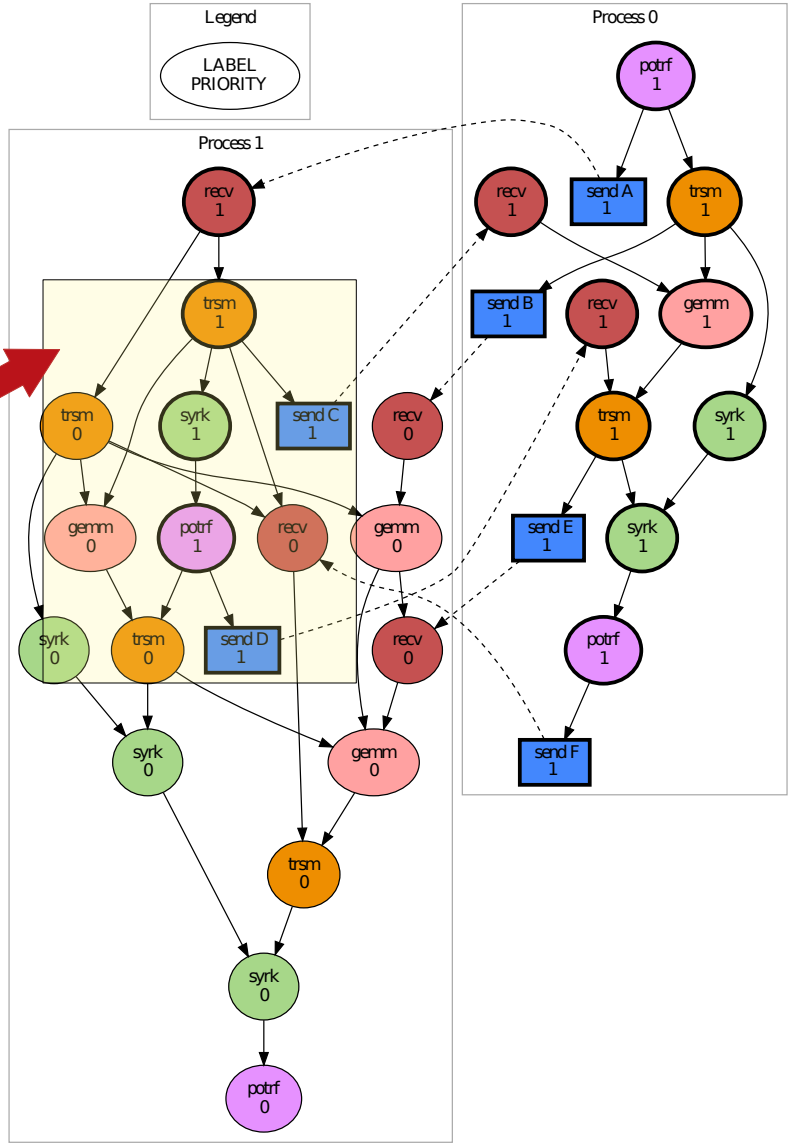
- Hint to **1** on **send-tasks** and on their **predecessors** recursively
- No hint on other tasks (0 by default)

Advantages

- Predecessors are marked

→ **syrk** - **potrf** over **gemm** and **recv**





Data-dependency graph and MA2 priorities

Communication-Aware Task Re-ordering Strategy (2/5)

Manual Approach (MA2)

- Hint to **1** on **send-tasks** and on their **predecessors** recursively
- No hint on other tasks (0 by default)

Advantages

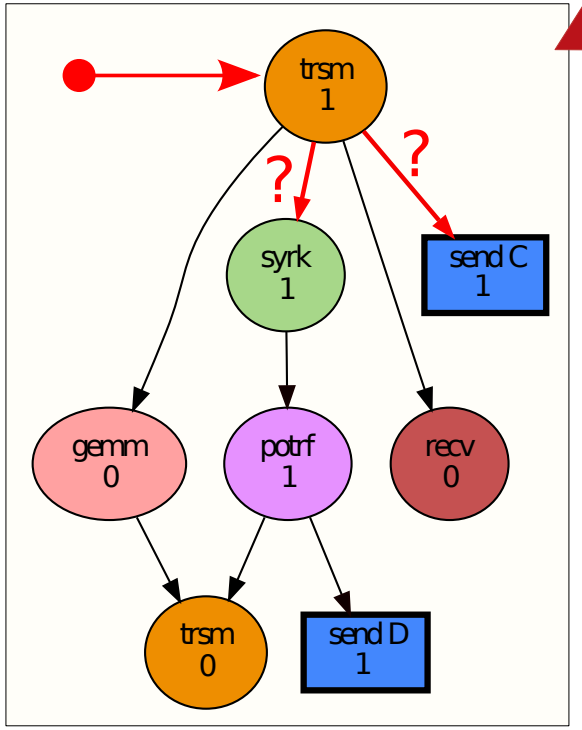
- Predecessors are marked

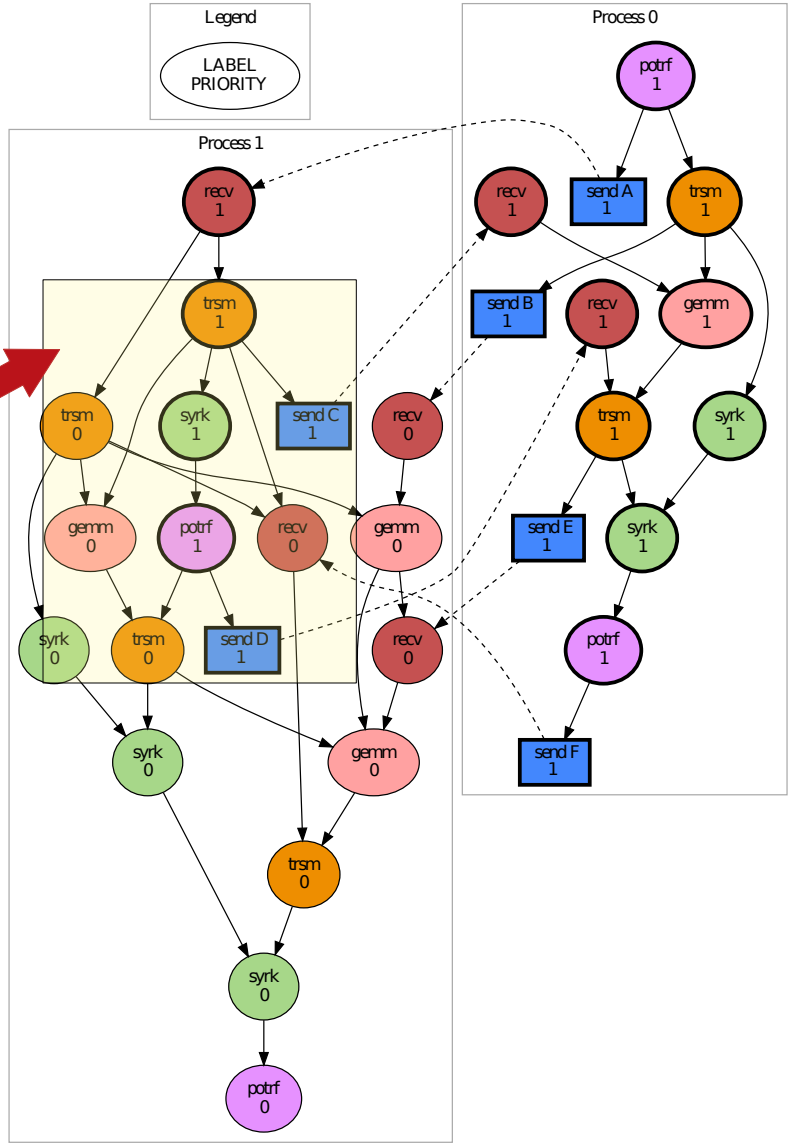
→ **syrk** - **potrf** over **gemm** and **recv**

Drawbacks

- Ready **send-tasks** are no longer top priority

→ **syrk** or **send C**?





Data-dependency graph and MA2 priorities

Communication-Aware Task Re-ordering Strategy (2/5)

Manual Approach (MA2)

- Hint to **1** on **send-tasks** and on their **predecessors** recursively
- No hint on other tasks (0 by default)

Advantages

- Predecessors are marked

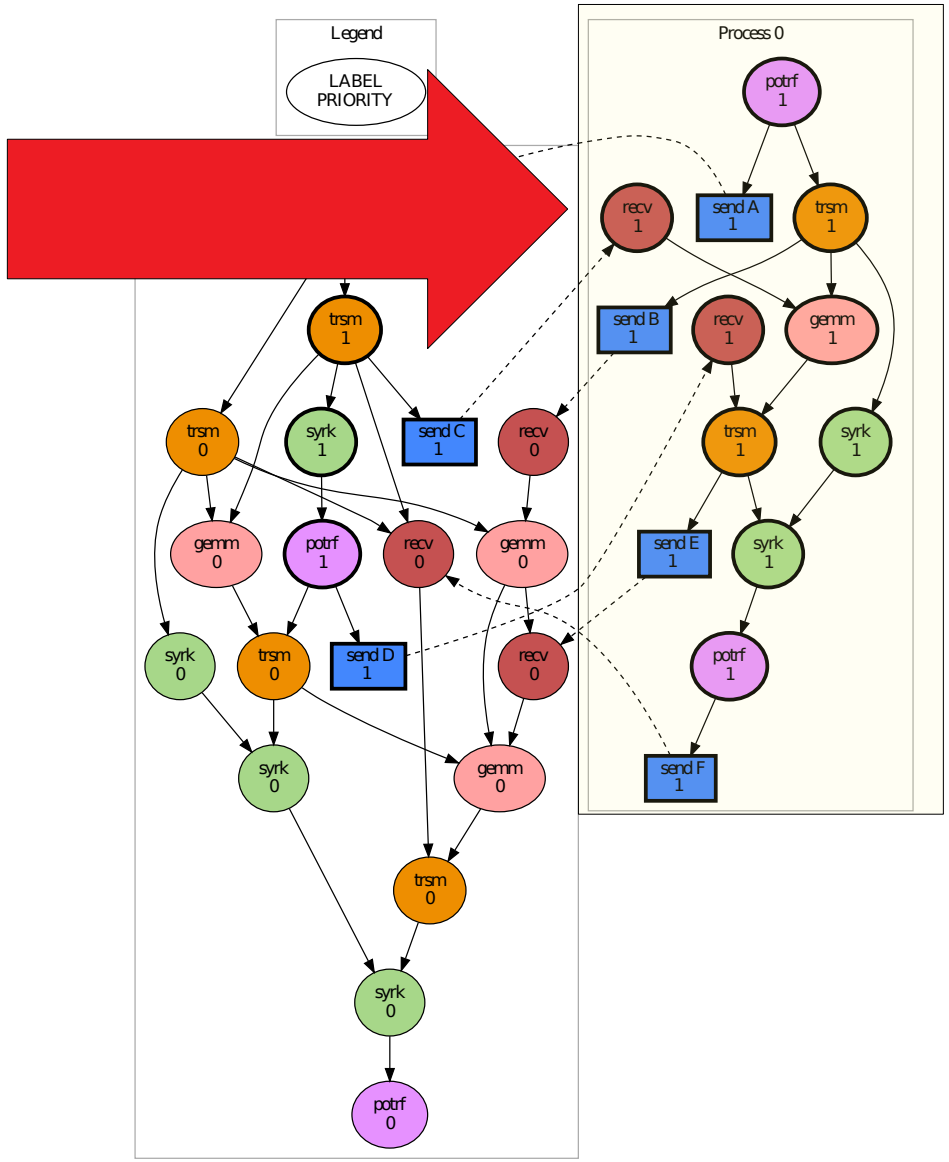
→ **syrk** - **potrf** over **gemm** and **recv**

Drawbacks

- Ready **send-tasks** are no longer top priority

→ **syrk** or **send C**?

- Process 0 priorities **gives no clue**



Data-dependency graph and MA2 priorities

Communication-Aware Task Re-ordering Strategy (2/5)

Manual Approach (MA2)

- Hint to **1** on **send-tasks** and on their **predecessors** recursively
- No hint on other tasks (0 by default)

Advantages

- Predecessors are marked

→ **syrk** - **potrf** over **gemm** and **recv**

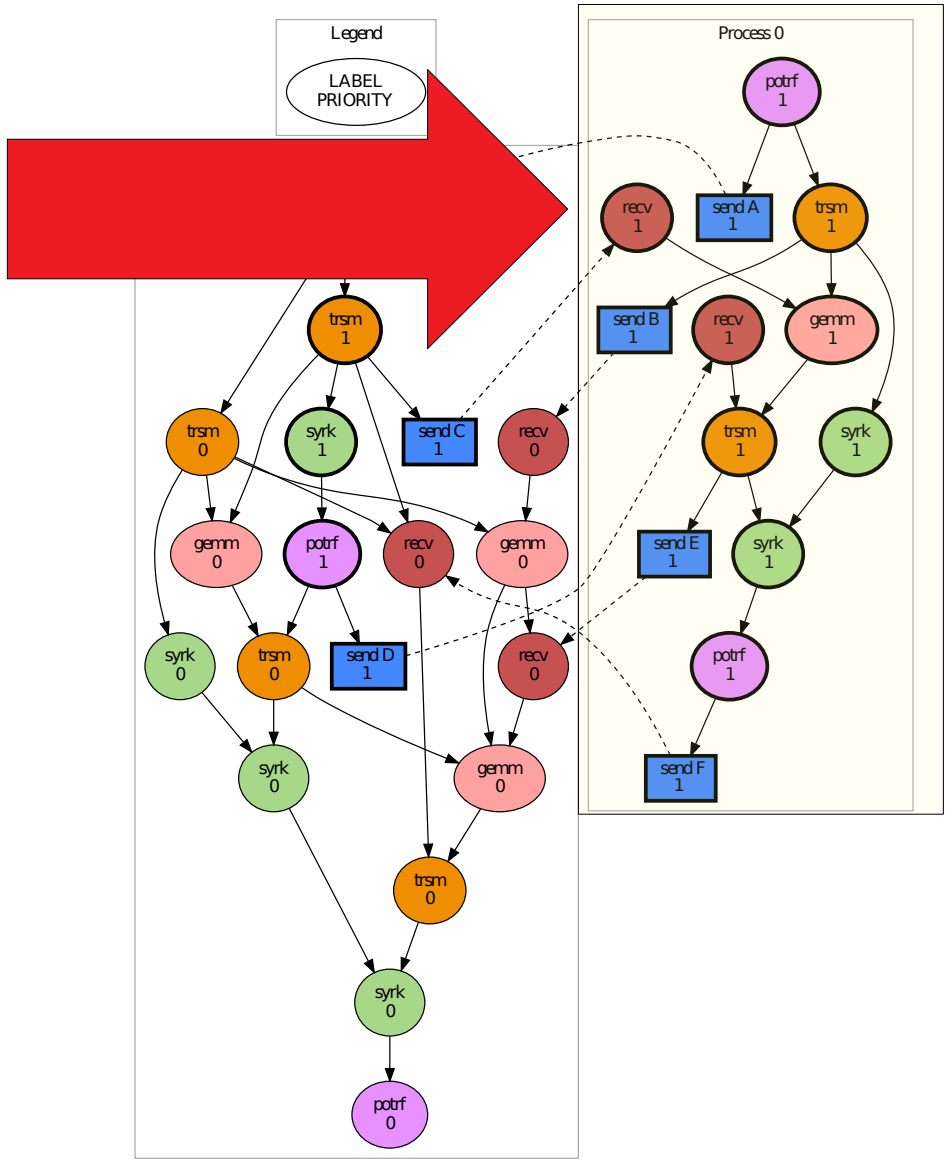
Drawbacks

- Ready **send-tasks** are no longer top priority

→ **syrk** or **send C**?

- Process 0 priorities **gives no clue**

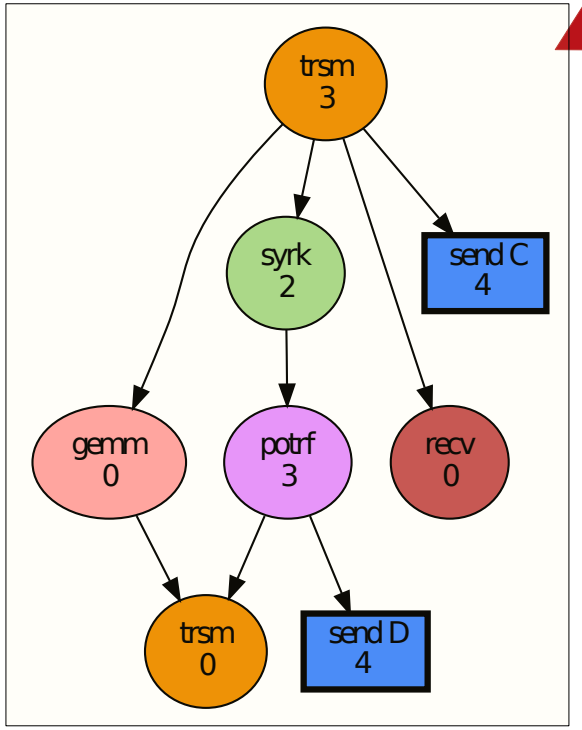
➡ Distinguish **send-tasks**, their **predecessors**, and other tasks

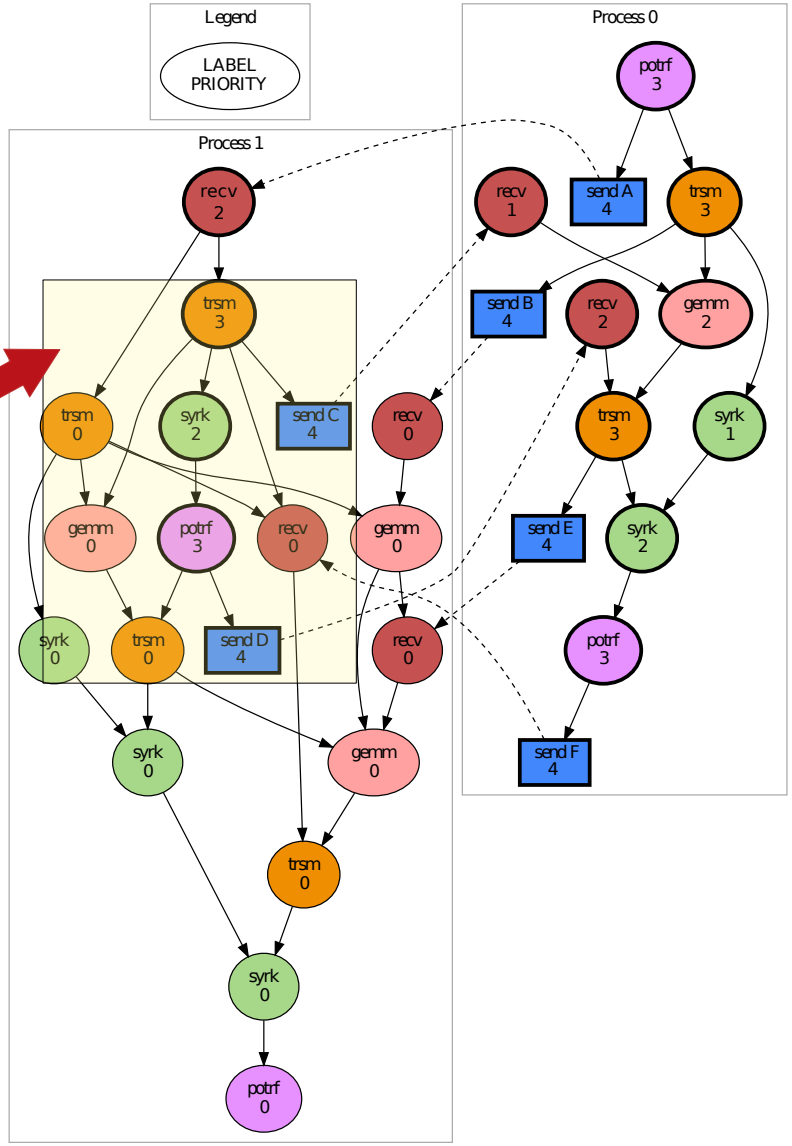


Data-dependency graph and MA2 priorities

Manual Approach (MA3)

- Hint to **omp_get_max_priority() - closest distance** to a **send-task**
- No hint on other tasks (0 by default)

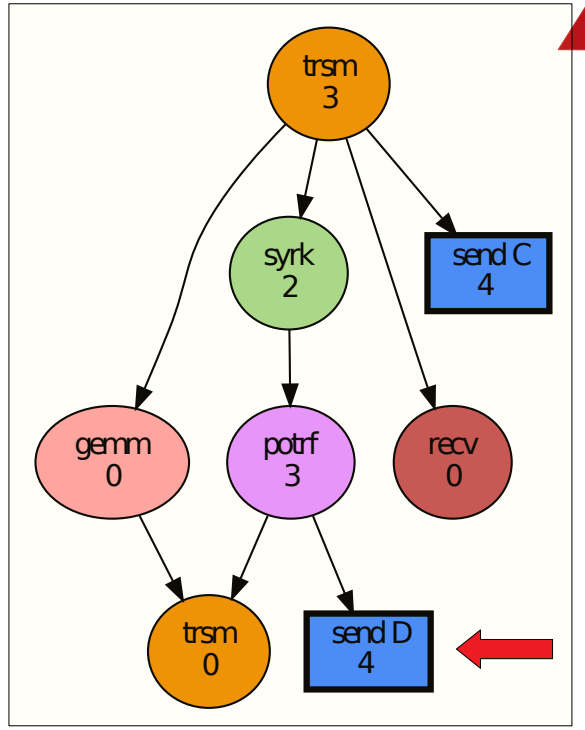


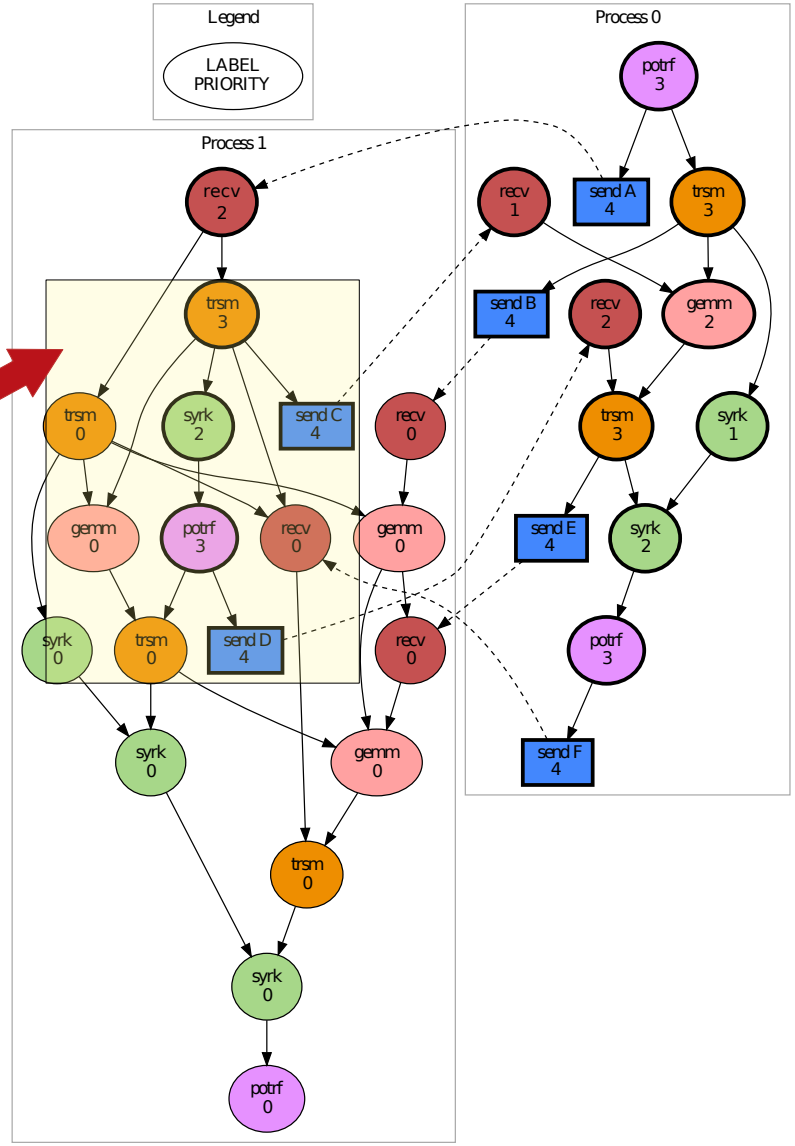


Data-dependency graph and MA3 priorities

Manual Approach (MA3)

- Hint to **omp_get_max_priority() - closest distance** to a **send-task**
- No hint on other tasks (0 by default)

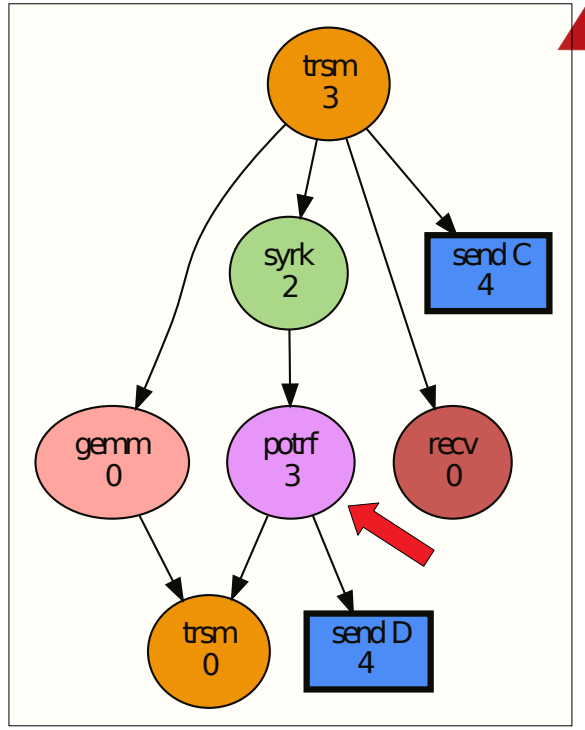


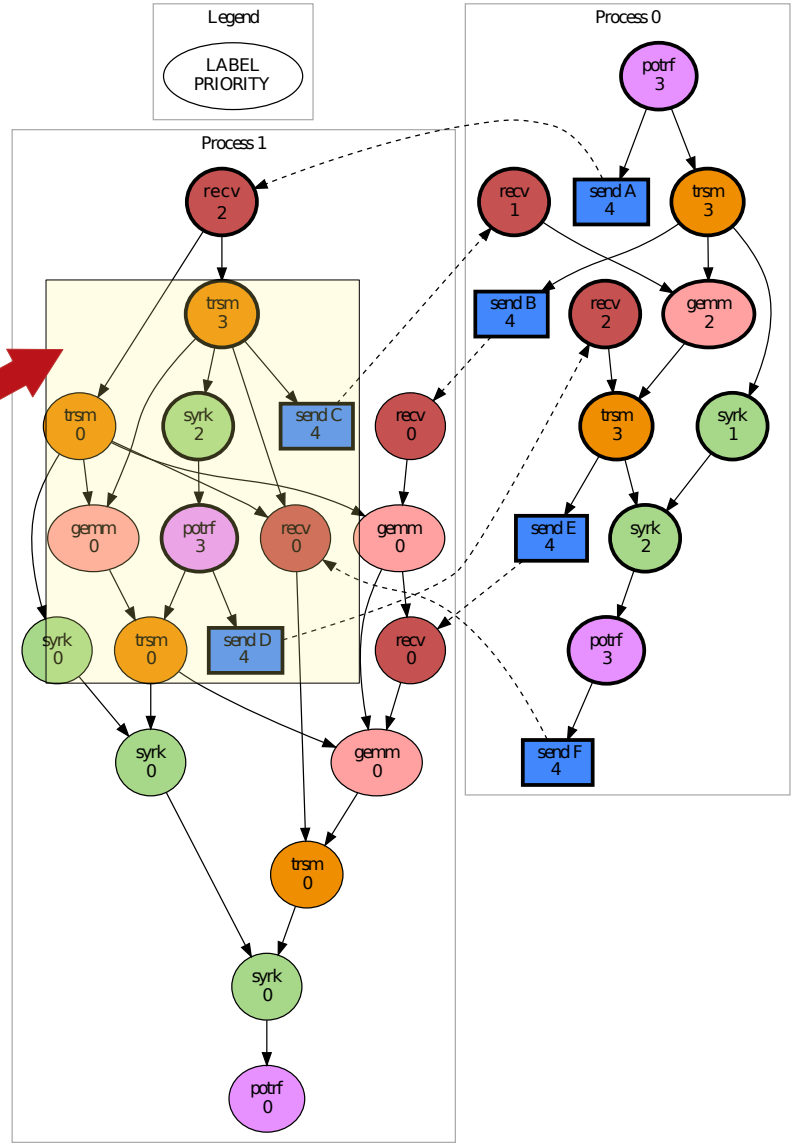


Data-dependency graph and MA3 priorities

Manual Approach (MA3)

- Hint to **omp_get_max_priority() - closest distance** to a **send-task**
- No hint on other tasks (0 by default)

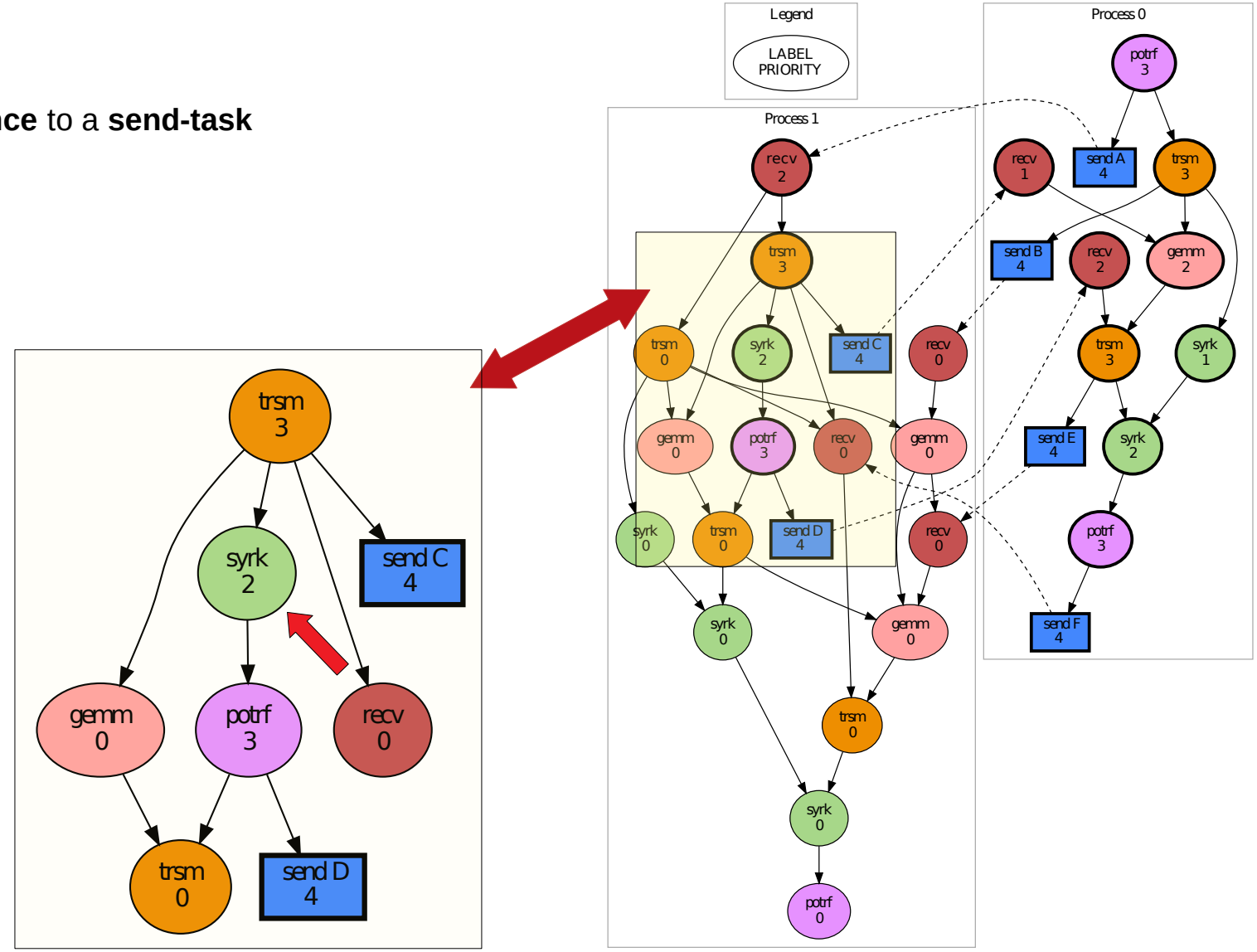




Data-dependency graph and MA3 priorities

Manual Approach (MA3)

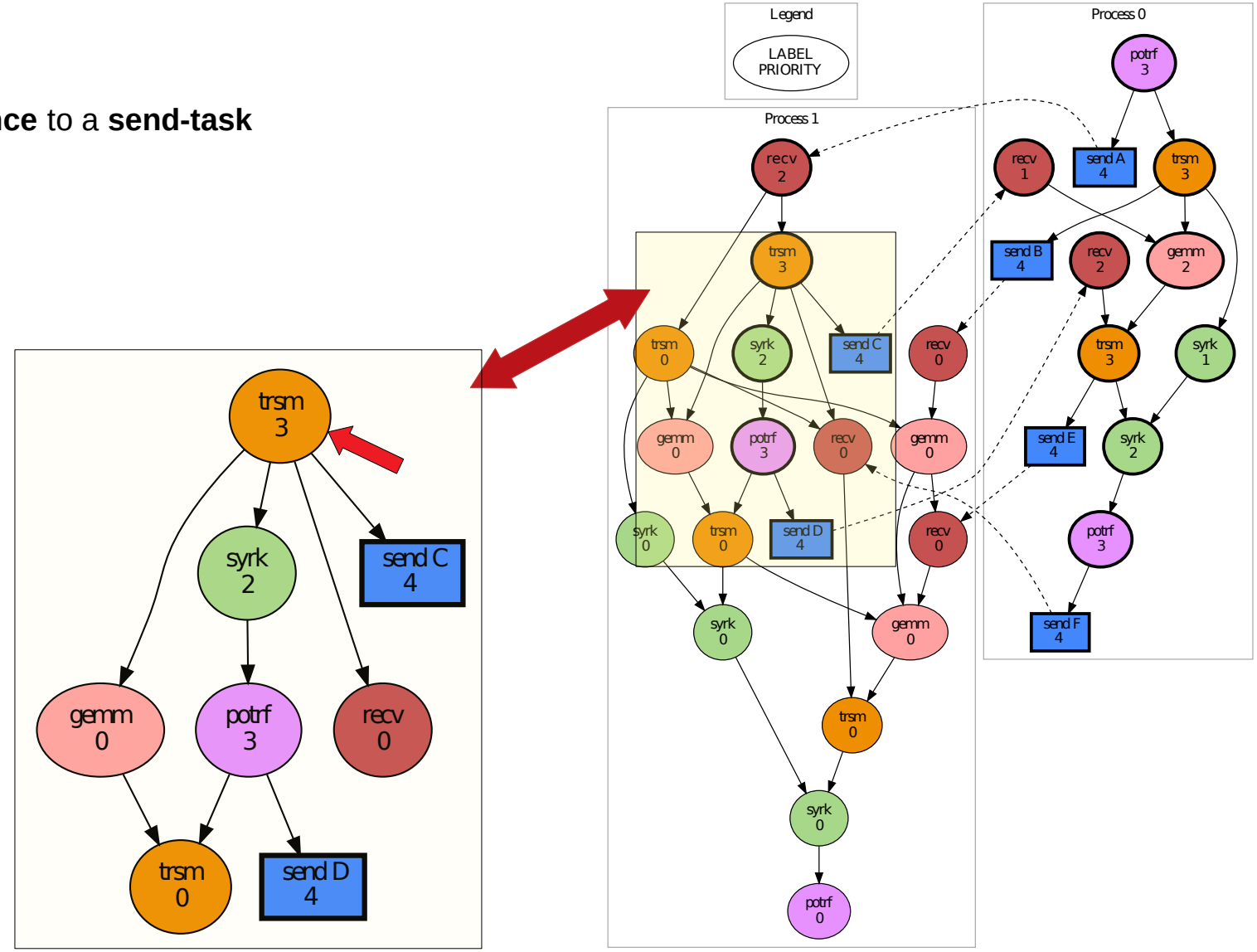
- Hint to **omp_get_max_priority() - closest distance** to a **send-task**
- No hint on other tasks (0 by default)



Data-dependency graph and MA3 priorities

Manual Approach (MA3)

- Hint to **omp_get_max_priority() - closest distance** to a **send-task**
- No hint on other tasks (0 by default)



Data-dependency graph and MA3 priorities

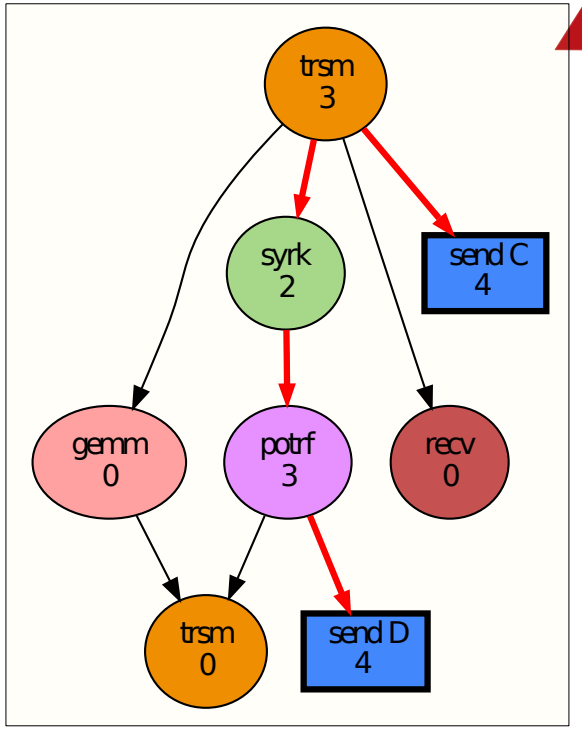
Manual Approach (MA3)

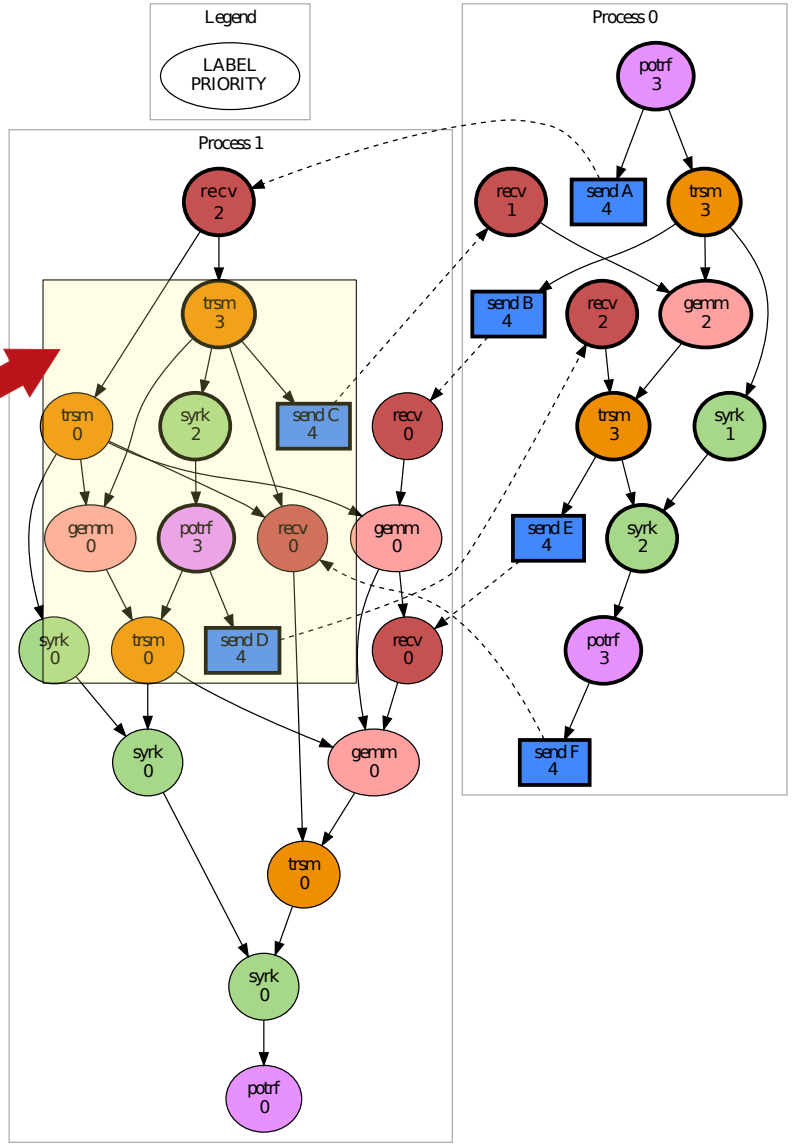
- Hint to **omp_get_max_priority() - closest distance** to a **send-task**
- No hint on other tasks (0 by default)

Advantages

- Whenever a send is ready, it is scheduled
  → Once **trsm** is completed, **send C** is elected

- Predecessors are marked
  → **syrk** - **potrf** over **gemm** and **recv**





Data-dependency graph and MA3 priorities

Communication-Aware Task Re-ordering Strategy (3/5)

Manual Approach (MA3)

- Hint to **omp_get_max_priority() - closest distance** to a **send-task**
- No hint on other tasks (0 by default)

Advantages

- Whenever a send is ready, it is scheduled

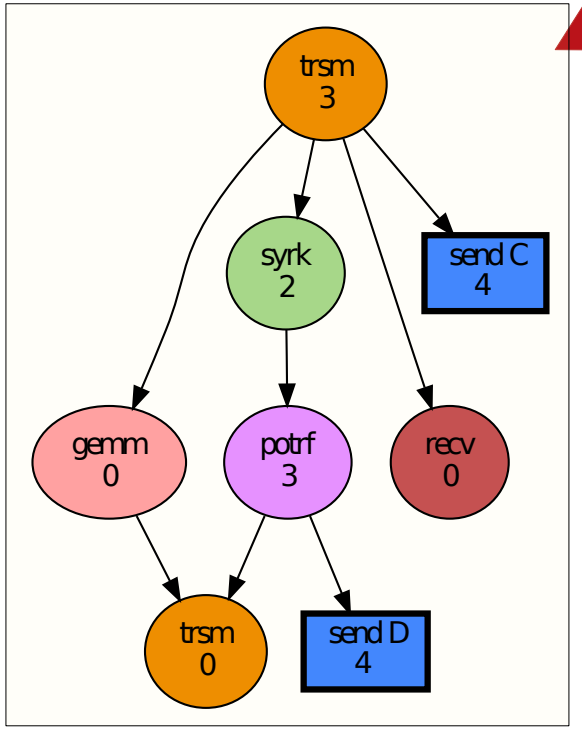
→ Once **trsm** is completed, **send C** is elected

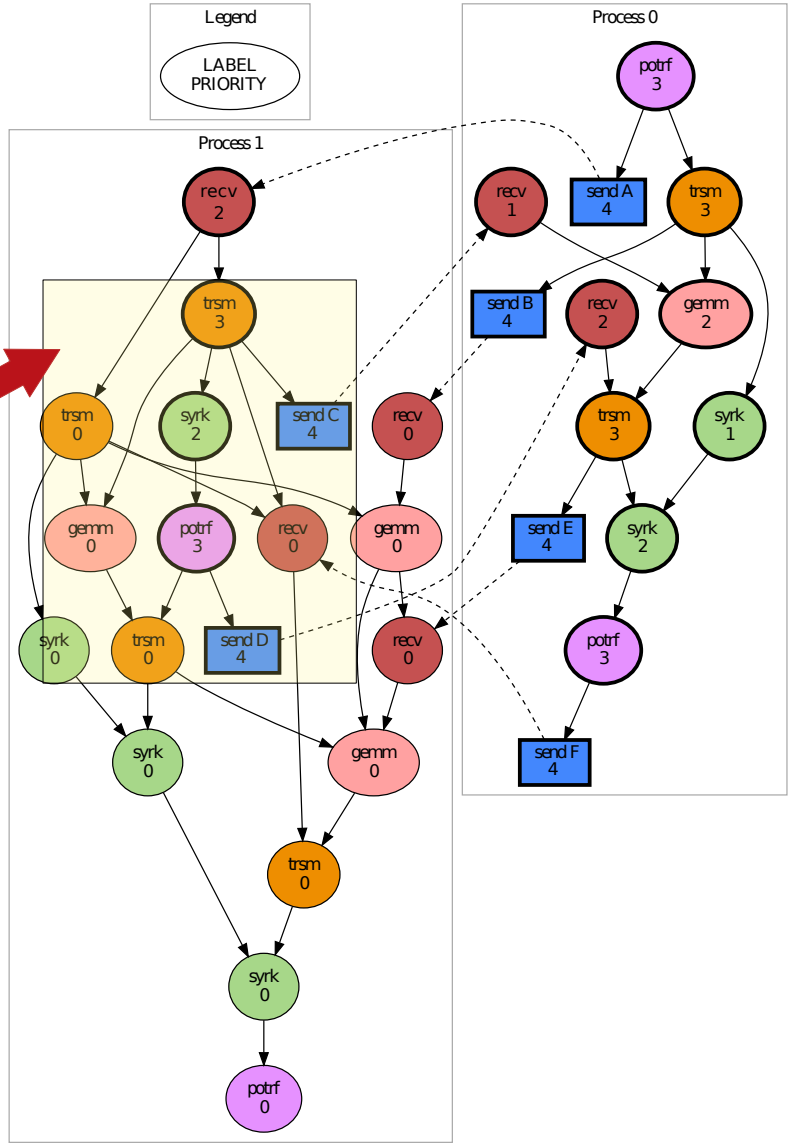
- Predecessors are marked

→ **syrk** - **potrf** over **gemm** and **recv**

Drawbacks

- Tedious to implement as a user





Data-dependency graph and MA3 priorities

Communication-Aware Task Re-ordering Strategy (3/5)

Manual Approach (MA3)

- Hint to **omp_get_max_priority() - closest distance** to a **send-task**
- No hint on other tasks (0 by default)

Advantages

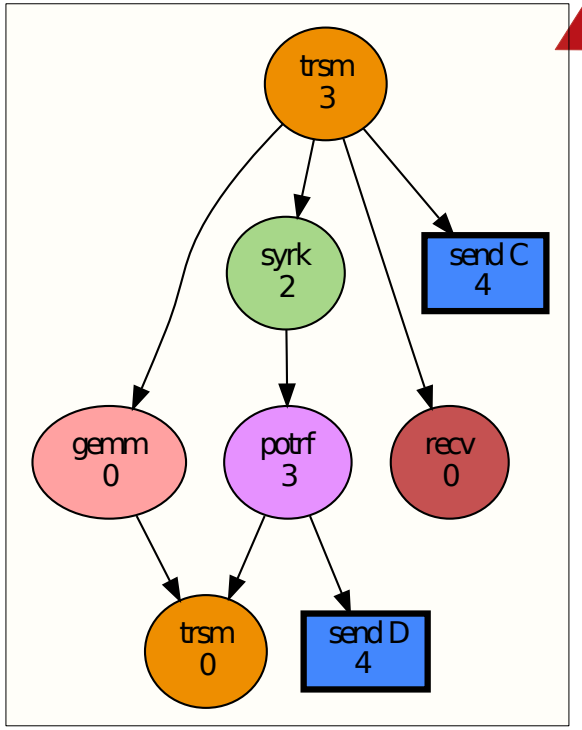
- Whenever a send is ready, it is scheduled
  → Once **trsm** is completed, **send C** is elected

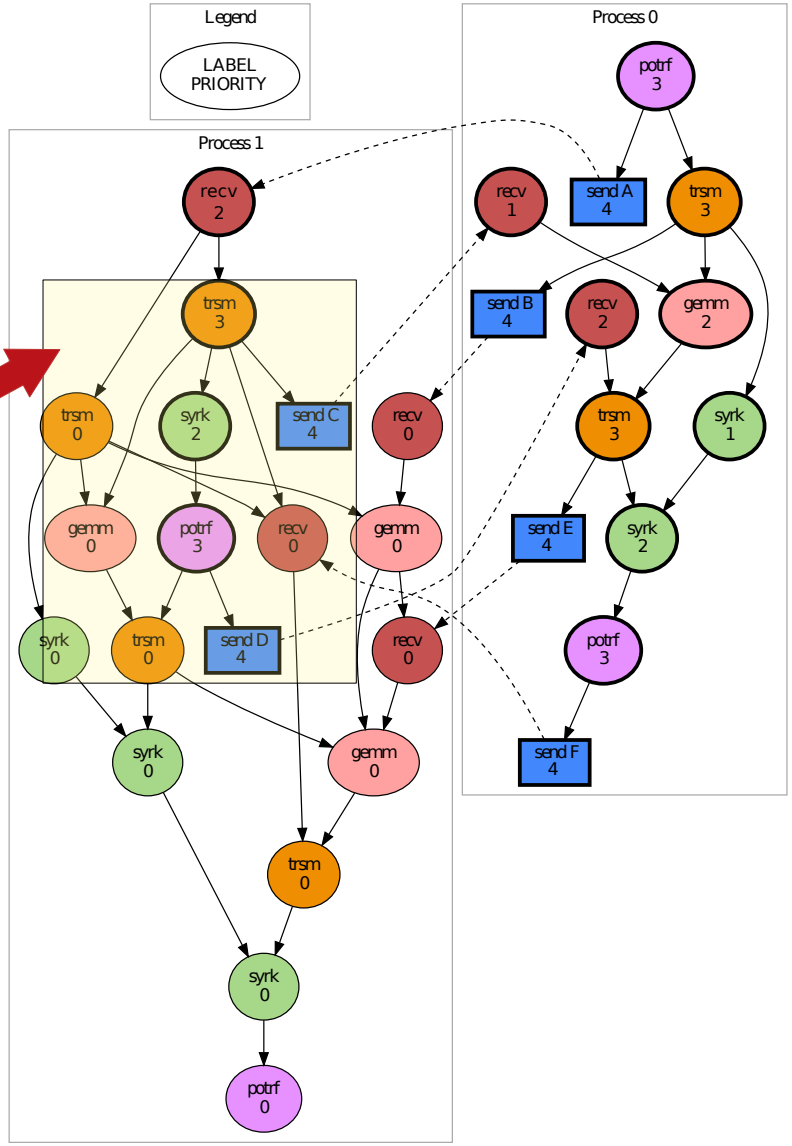
- Predecessors are marked
  → **syrk** - **potrf** over **gemm** and **recv**

Drawbacks

- Tedious to implement as a user

→ Towards runtime **automation**





Data-dependency graph and MA3 priorities

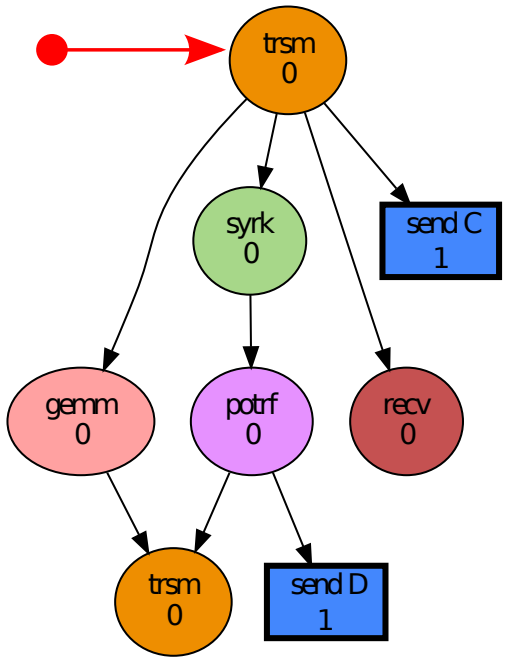
Semi-Automatic (SA)

Manually

- Specific priority hint **send-tasks** *(e.g, 1)*
- No priority on other tasks

```
1    # pragma omp task depends(in: data) priority(1)
2    {
3        […]
4        MPI_Send(data, …) ;
5        […]
6     }
```

Hybrid MPI+OpenMP(tasks) minimal code example



Graph given by the user

Semi-Automatic (SA)

Manually

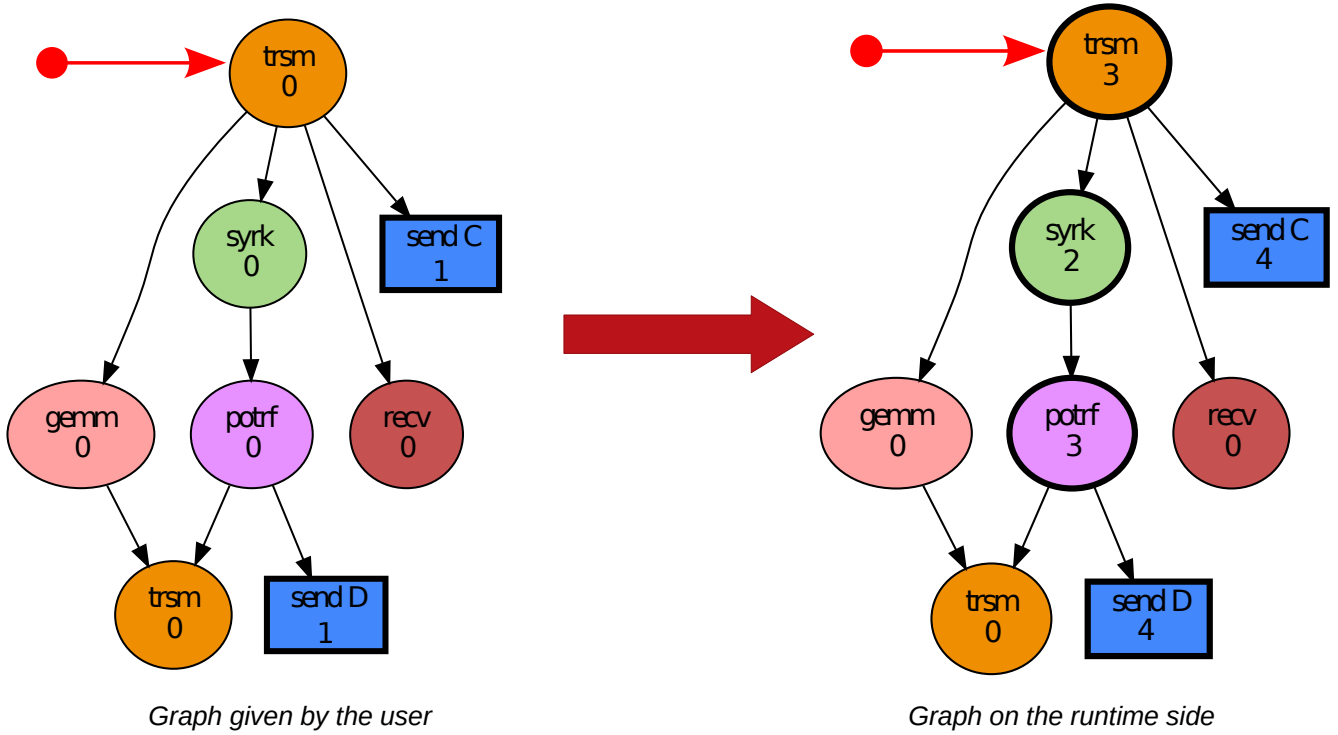
- Specific priority hint **send-tasks** *(e.g, 1)*
- No priority on other tasks

```
1    # pragma omp task depends(in: data) priority(1)
2    {
3        […]
4        MPI_Send(data, …) ;
5        […]
6     }
```

Hybrid MPI+OpenMP(tasks) minimal code example

Automatically

- Runtime converts priorities to **MA3**
  - Back propagation algorithm



Graph given by the user *Graph on the runtime side*

Semi-Automatic (SA)

Manually

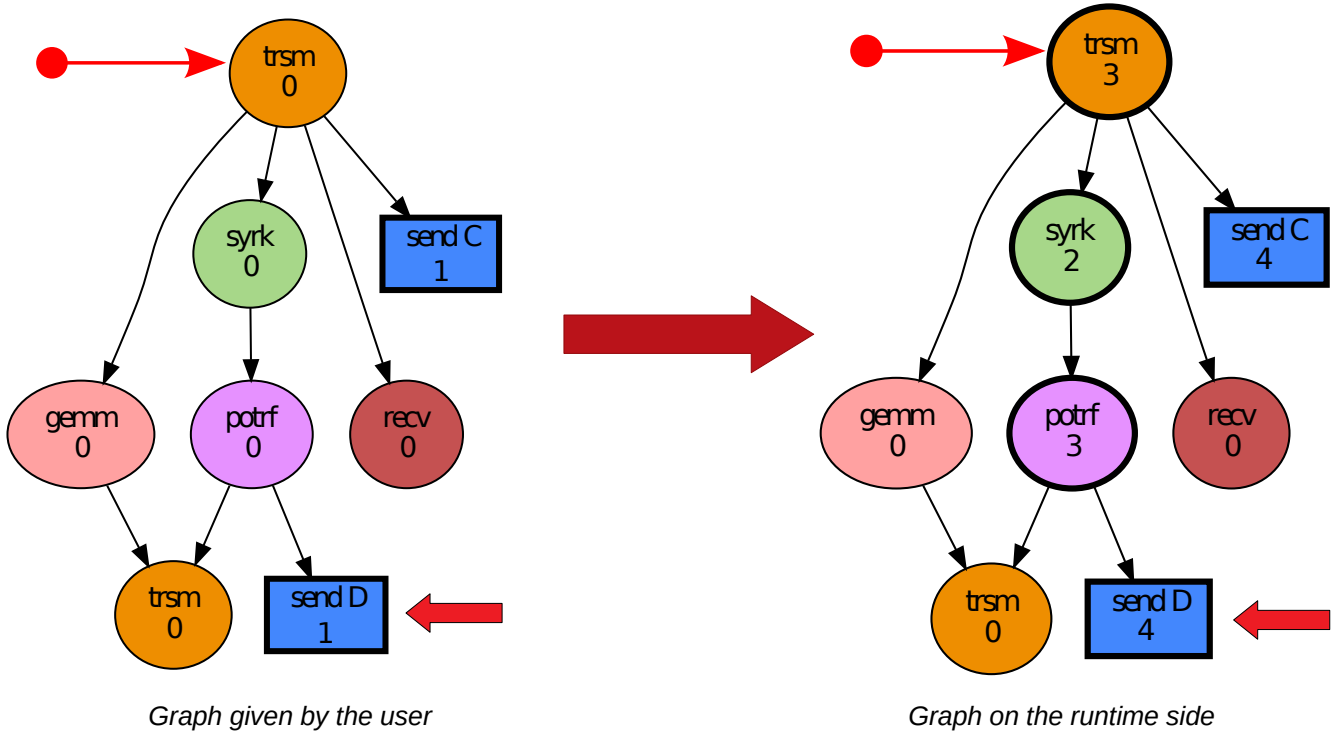
- Specific priority hint **send-tasks** *(e.g, 1)*
- No priority on other tasks

```
1    # pragma omp task depends(in: data) priority(1)
2    {
3        […]
4        MPI_Send(data, …) ;
5        […]
6     }
```

Hybrid MPI+OpenMP(tasks) minimal code example

Automatically

- Runtime converts priorities to **MA3**
  - Back propagation algorithm



Graph given by the user *Graph on the runtime side*

Semi-Automatic (SA)

Manually

- Specific priority hint **send-tasks** *(e.g, 1)*
- No priority on other tasks

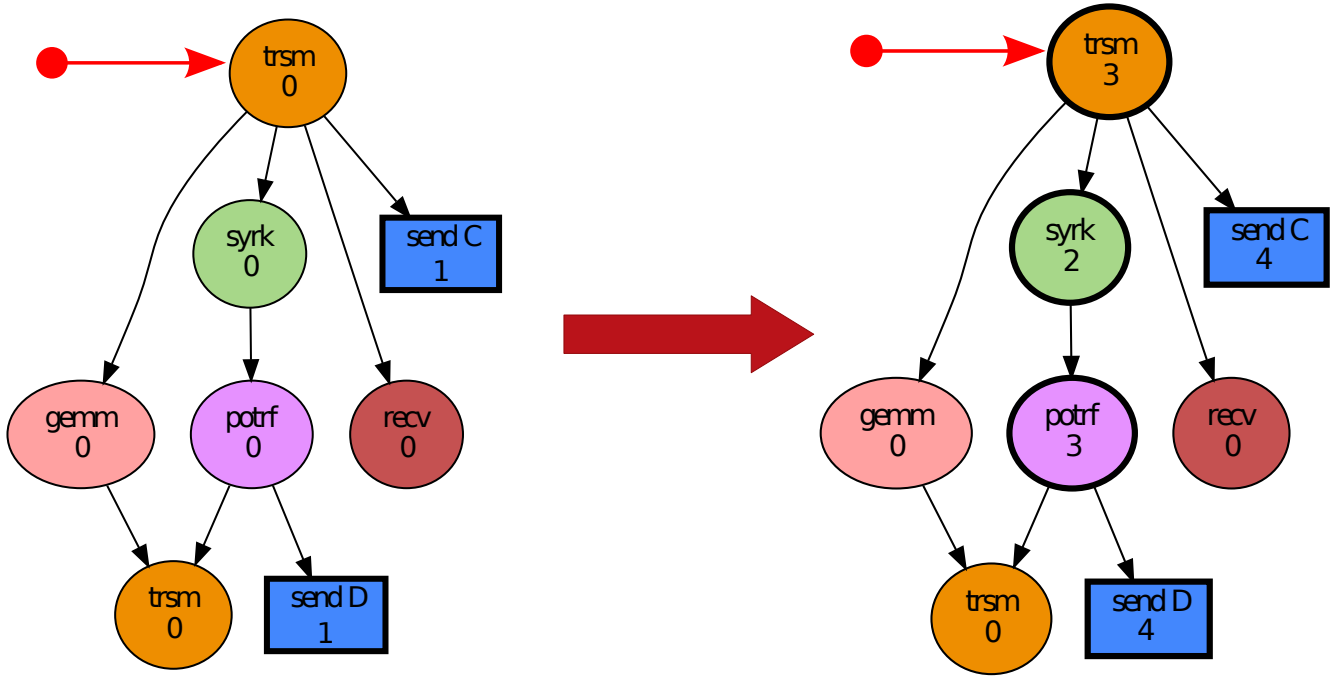
```
1    # pragma omp task depends(in: data) priority(1)
2    {
3        […]
4        MPI_Send(data, …) ;
5        […]
6     }
```

Hybrid MPI+OpenMP(tasks) minimal code example

Automatically

- Runtime converts priorities to **MA3**
  - Back propagation algorithm

➡ Towards **send-task detection**



Graph given by the user *Graph on the runtime side*

Communication-Aware Task Re-ordering Strategy (5/5)

Fully-automatic (FA)

- The user sets no priority on any tasks
- The scheduler **detects** send-tasks, and sets priorities

Communication-Aware Task Re-ordering Strategy (5/5)

Fully-automatic (FA)

- The user sets no priority on any tasks
- The scheduler **detects** send-tasks, and sets priorities

Send-tasks detection by the runtime

- No hint on task creation – too late on task execution

Fully-automatic (FA)

- The user sets no priority on any tasks
- The scheduler **detects** send-tasks, and sets priorities

Send-tasks detection by the runtime

⚠ No hint on task creation – too late on task execution

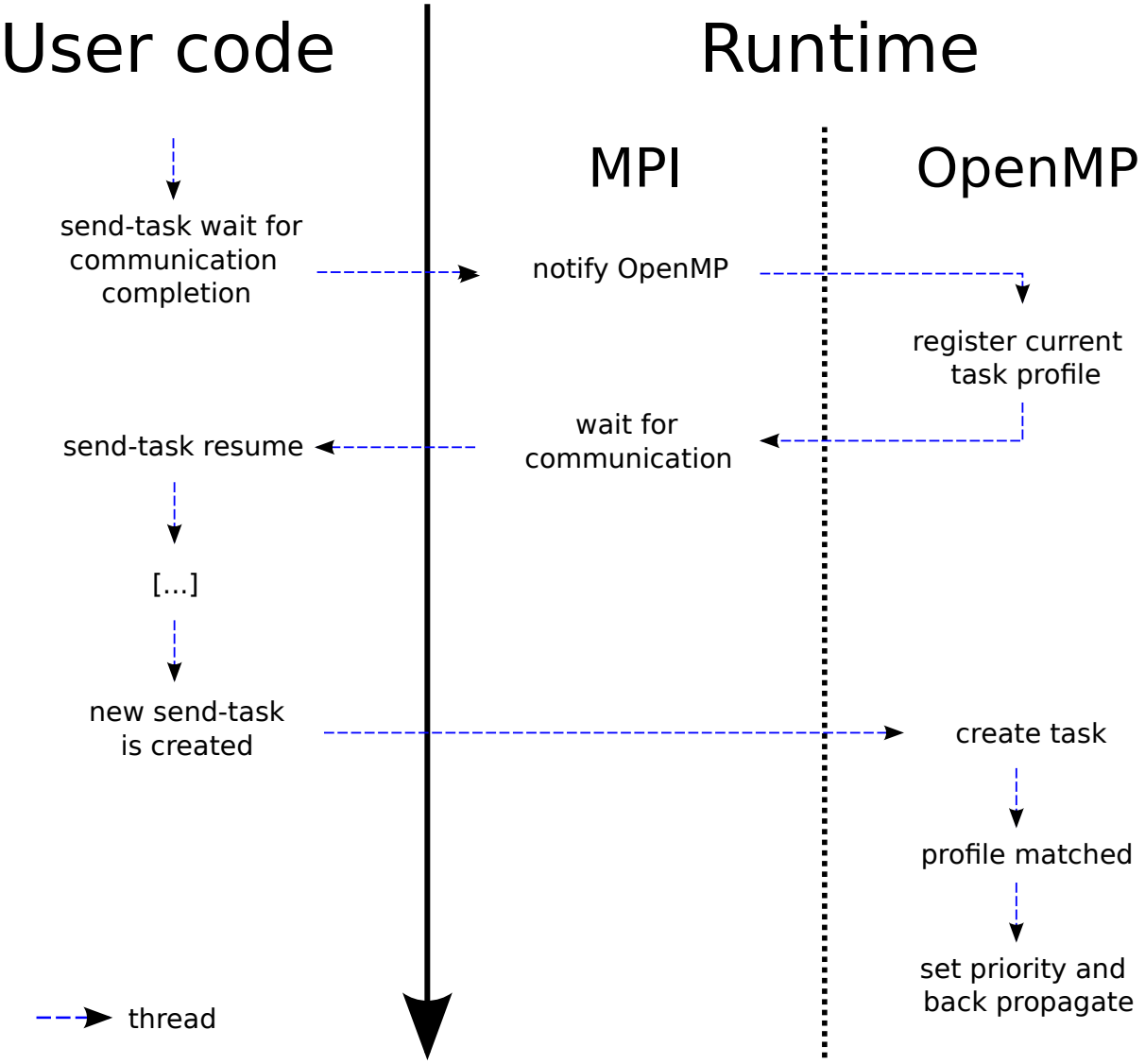
On **send-tasks execution**

- MPI runtime notifies OpenMP runtime

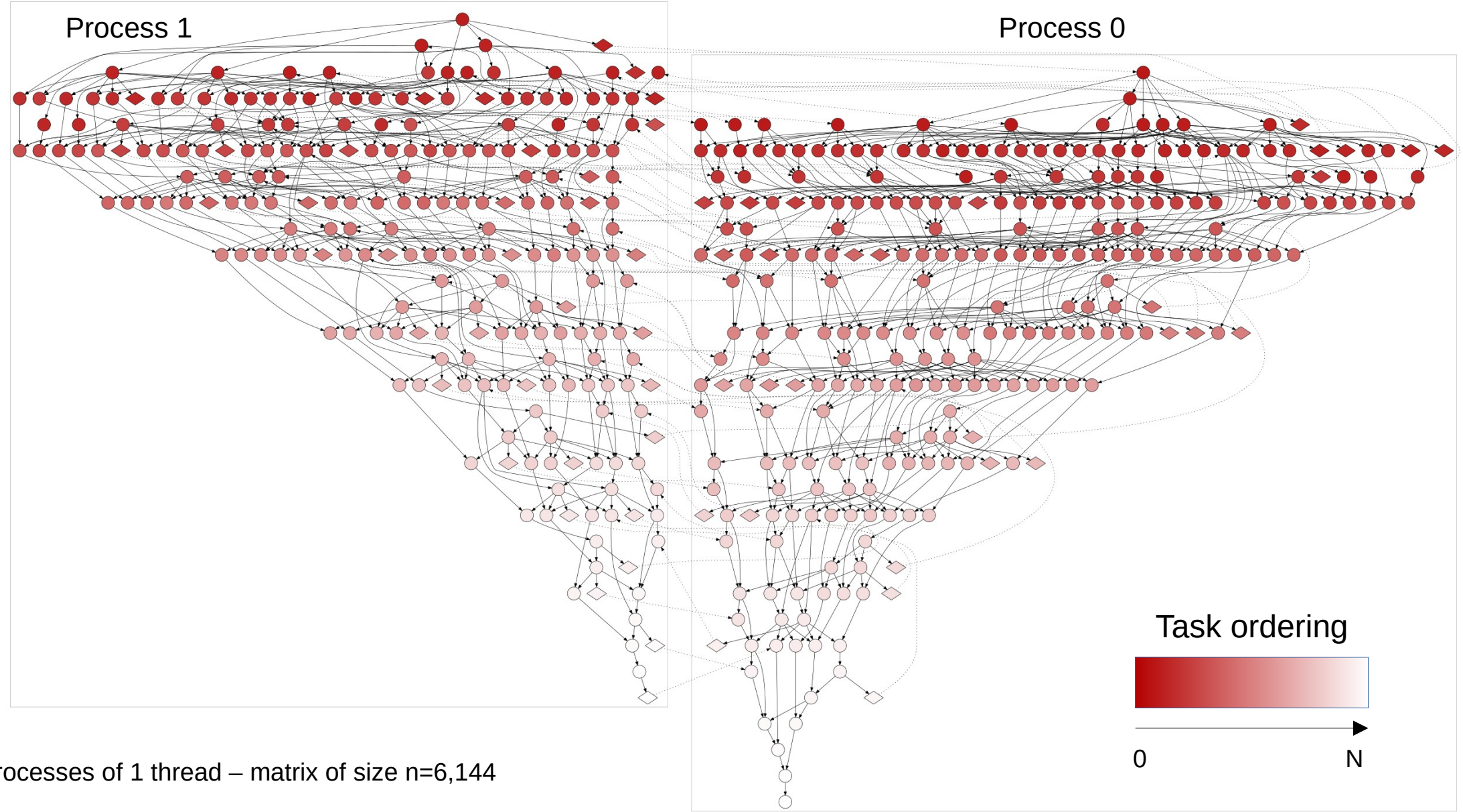
- OpenMP runtime stores current task profile
  - Size (shared variables)
  - Properties (tiedness, final…)
  - Parent task identifier
  - Number of predecessors (fully-known at run-time)
  - Number of successors (may be incomplete)

- On future tasks creations, and on idle periods
  - Match profiles to detect send-tasks
  - Propagate priorities

Target **iterative-based** applications

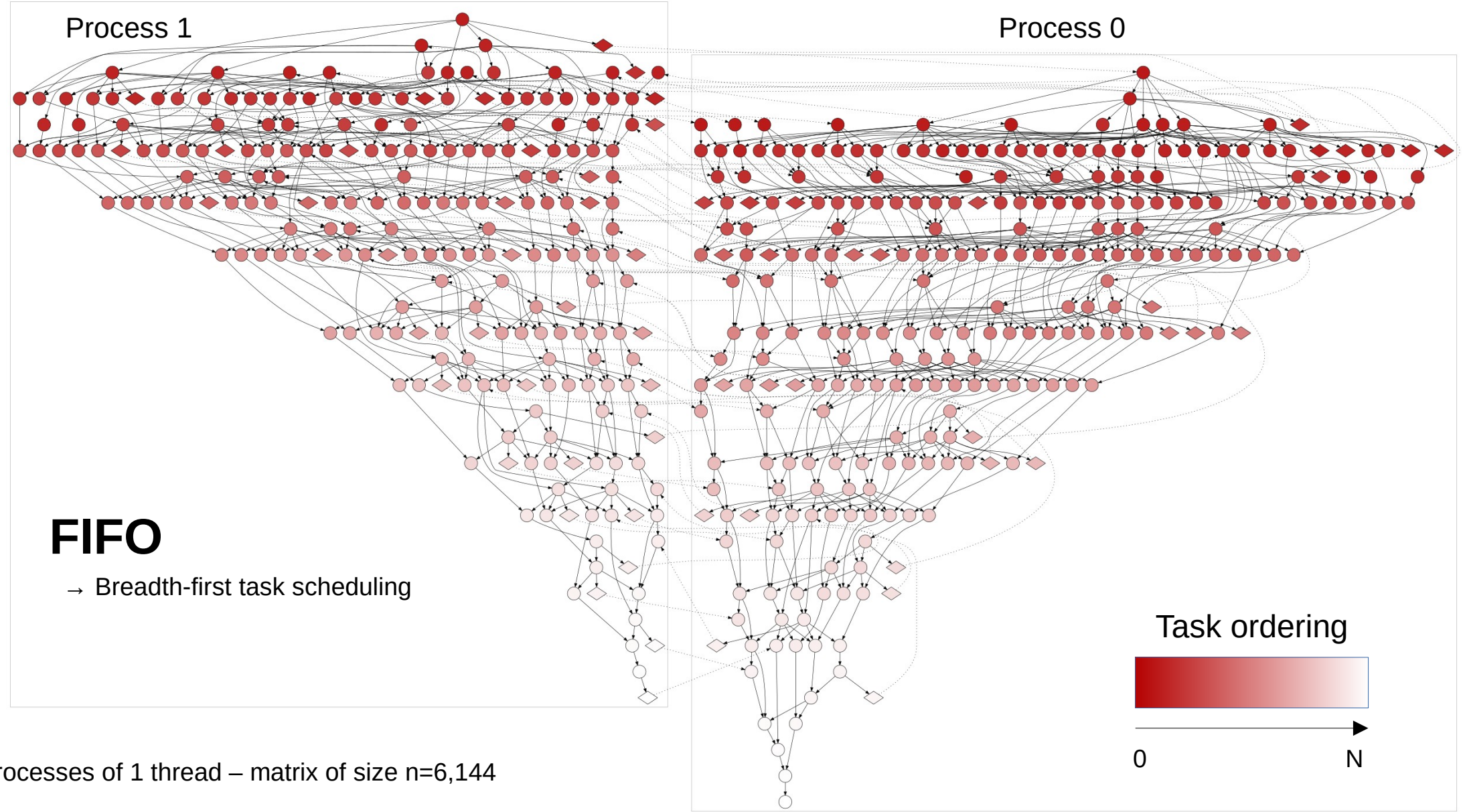


Communication-Aware Task Re-ordering Strategy - Summary



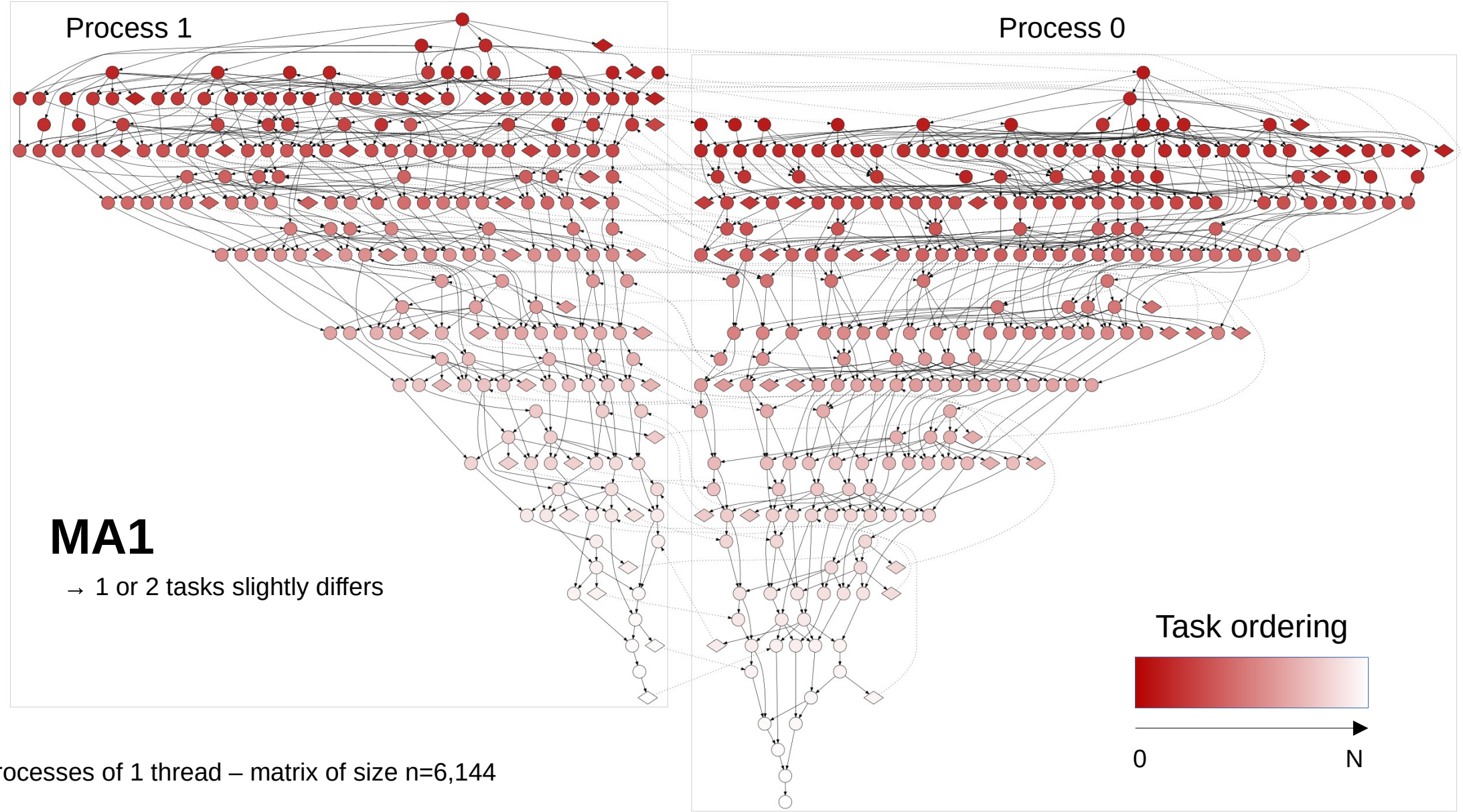
2 processes of 1 thread – matrix of size n=6,144

Communication-Aware Task Re-ordering Strategy - Summary



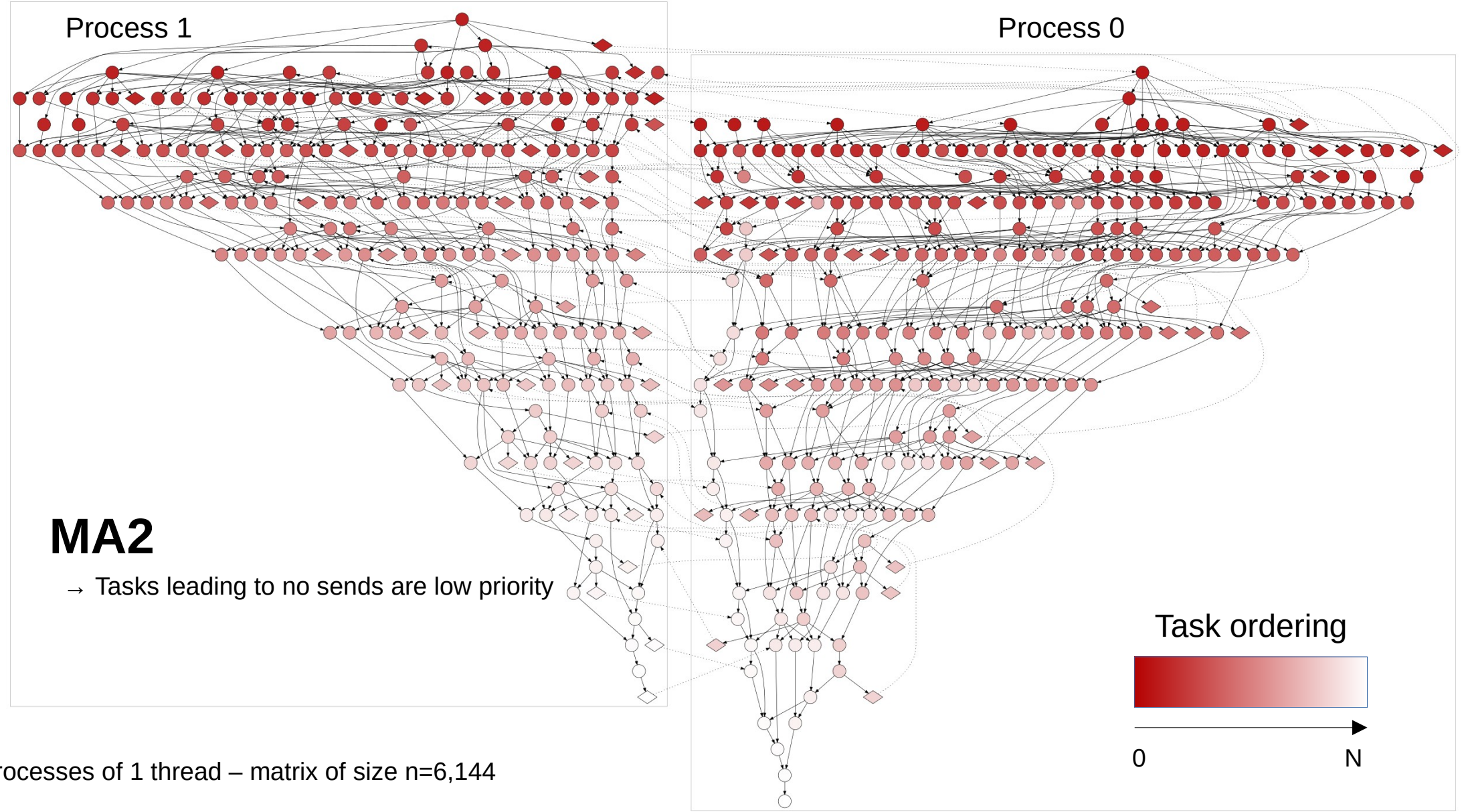
2 processes of 1 thread – matrix of size n=6,144

Communication-Aware Task Re-ordering Strategy - Summary



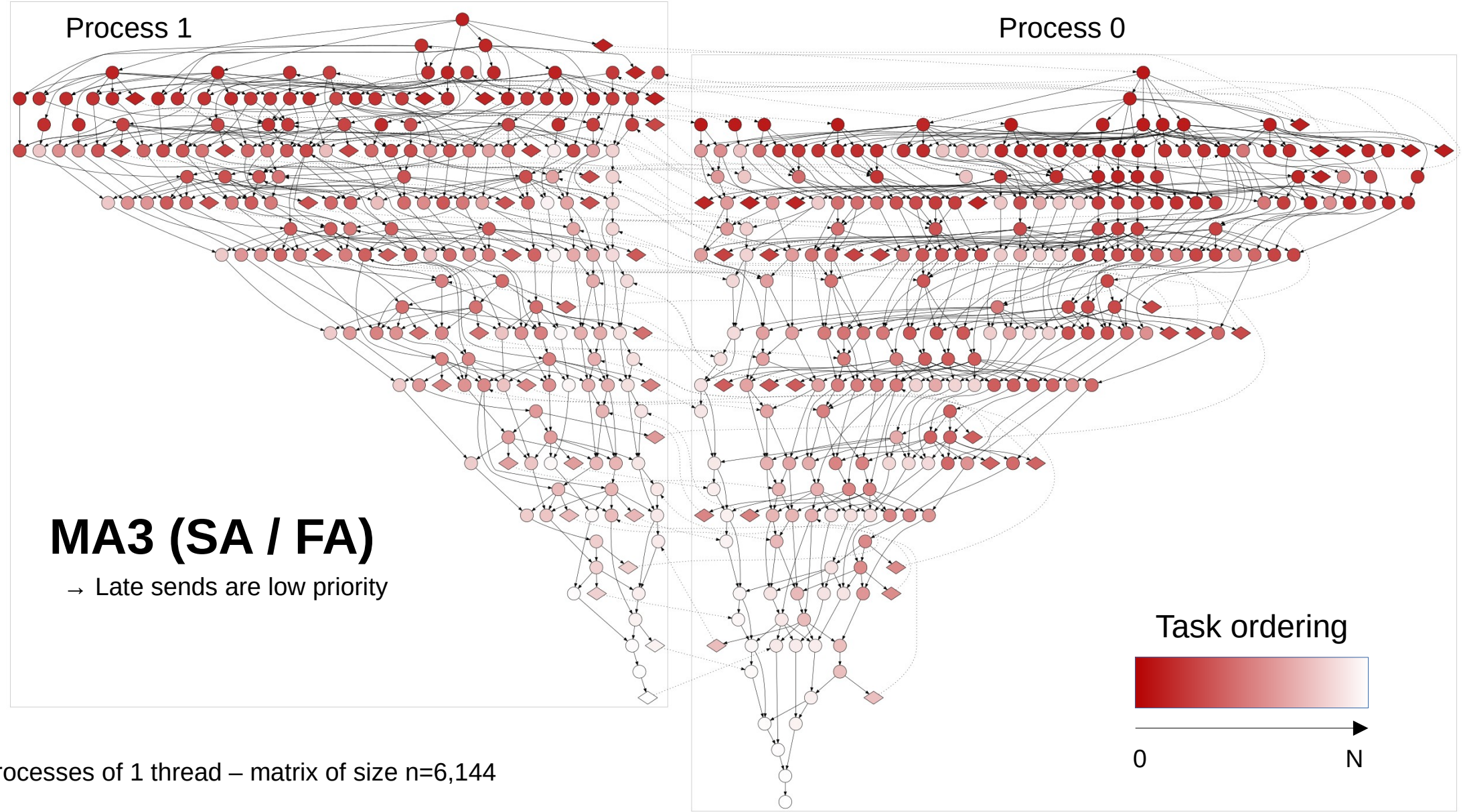
2 processes of 1 thread – matrix of size n=6,144

Communication-Aware Task Re-ordering Strategy - Summary



2 processes of 1 thread – matrix of size n=6,144

Communication-Aware Task Re-ordering Strategy - Summary



2 processes of 1 thread – matrix of size n=6,144

Communication-aware task re-ordering strategy

- Priority policies
- **Implementation**

Implementation (1/2)

Multi-Processor Computing (MPC) framework

- Unified implementation of OpenMP and MPI relying on user threads **[MPC-08, MPC-10]**
- Open source implementation at https://mpc.hpcframework.com/

[MPC-08] – M. Pérache, H. Jourdren, and R. Namyst, MPC: a unified parallel runtime for clusters of NUMA machines, in Proceedings of the 14th international EURO-PAR conference (EURO-PAR 2008).
[MPC-10] – P. Carribault, M. Pérache, and H. Jourdren, Enabling low-overhead hybrid MPI/OpenMP parallelism with MPC, in International workshop on OpenMP (IWOMP'10), 2010

Implementation (1/2)

Multi-Processor Computing (MPC) framework

- Unified implementation of OpenMP and MPI relying on user threads **[MPC-08, MPC-10]**
- Open source implementation at https://mpc.hpcframework.com/

OpenMP task scheduling

- User context
- Red-black trees as priority queues
- Topological work-stealing

[MPC-08] – M. Pérache, H. Jourdren, and R. Namyst, MPC: a unified parallel runtime for clusters of NUMA machines, in Proceedings of the 14th international EURO-PAR conference (EURO-PAR 2008).
[MPC-10] – P. Carribault, M. Pérache, and H. Jourdren, Enabling low-overhead hybrid MPI/OpenMP parallelism with MPC, in International workshop on OpenMP (IWOMP'10), 2010

Implementation (1/2)

Multi-Processor Computing (MPC) framework

- Unified implementation of OpenMP and MPI relying on user threads **[MPC-08, MPC-10]**
- Open source implementation at https://mpc.hpcframework.com/

OpenMP task scheduling

- User context
- Red-black trees as priority queues
- Topological work-stealing

Priorities computation

- MA & SA → synchronously on task creation by the producer thread
- FA → asynchronously by any threads on **idle** periods

[MPC-08] – M. Pérache, H. Jourdren, and R. Namyst, MPC: a unified parallel runtime for clusters of NUMA machines, in Proceedings of the 14th international EURO-PAR conference (EURO-PAR 2008).
[MPC-10] – P. Carribault, M. Pérache, and H. Jourdren, Enabling low-overhead hybrid MPI/OpenMP parallelism with MPC, in International workshop on OpenMP (IWOMP'10), 2010

Implementation (2/2)

MPI+OpenMP interoperability

```
1   # pragma omp task depends(in: data)
2   {
3       [...]
4       MPI_Request request;
5       MPI_Isend(data, …, &request) ;
6       [...]
7       MPI_Wait(&request, …) ;
8       [...]
9    }
```

User-side hybrid MPI+OpenMP(tasks) code

Implementation (2/2)

MPI+OpenMP interoperability

- The loss of thread issue → task switch

```
1   # pragma omp task depends(in: data)
2   {
3       [...]
4       MPI_Request request;
5       MPI_Isend(data, …, &request) ;
6       [...]
7       MPI_Wait(&request, …) ;
8       [...]
9    }
```

User-side hybrid MPI+OpenMP(tasks) code

Implementation (2/2)

MPI+OpenMP interoperability

- The loss of thread issue → task switch

```
1   # pragma omp task depends(in: data)
2   {
3       [...]
4       MPI_Request request;
5       MPI_Isend(data, …, &request) ;
6       [...]
7       MPI_Wait(&request, …) ;
8       [...]
9    }
```

User-side hybrid MPI+OpenMP(tasks) code

```
11  /* Waits for an MPI requests to complete */
12  void MPI_Wait(MPI_Request * request, MPI_Status * status)
13  {
14      if (mpc_omp_in_explicit_task())
15      {
16
17
18
19
20
21
22
23
24
25          /* Block current task until its associated event is fulfilled */
26          mpc_omp_task_block(request->omp_event);
27
28       } else {
29           /* Standard MPI_Wait implementation [...] */
30       }
31   }
```

MPC MPI+OpenMP runtimes interoperation

Implementation (2/2)

MPI+OpenMP interoperability

- The loss of thread issue → task switch

```
1   # pragma omp task depends(in: data)
2   {
3       [...]
4       MPI_Request request;
5       MPI_Isend(data, …, &request) ;
6       [...]
7       MPI_Wait(&request, …) ;
8       [...]
9    }
```

User-side hybrid MPI+OpenMP(tasks) code

```
11  /* Waits for an MPI requests to complete */
12  void MPI_Wait(MPI_Request * request, MPI_Status * status)
13  {
14      if (mpc_omp_in_explicit_task())
15      {
16
17
18
19
20
21
22
23
24
25          /* Block current task until its associated event is fulfilled */
26          mpc_omp_task_block(request->omp_event);
27
28       } else {
29           /* Standard MPI_Wait implementation [...] */
30       }
31   }
```

MPC MPI+OpenMP runtimes interoperation

Implementation (2/2)

MPI+OpenMP interoperability

- The loss of thread issue → task switch
- Task blocking **is not** yielding – using `omp_event_handle_t`

```
1   # pragma omp task depends(in: data)
2   {
3       [...]
4       MPI_Request request;
5       MPI_Isend(data, …, &request) ;
6       [...]
7       MPI_Wait(&request, …) ;
8       [...]
9    }
```

User-side hybrid MPI+OpenMP(tasks) code

```
11  /* Waits for an MPI requests to complete */
12  void MPI_Wait(MPI_Request * request, MPI_Status * status)
13  {
14      if (mpc_omp_in_explicit_task())
15      {
16
17
18
19
20
21
22
23
24
25          /* Block current task until its associated event is fulfilled */
26          mpc_omp_task_block(request->omp_event);
27
28       } else {
29           /* Standard MPI_Wait implementation [...] */
30       }
31   }
```

MPC MPI+OpenMP runtimes interoperation

Implementation (2/2)

MPI+OpenMP interoperability

- The loss of thread issue → task switch
- Task blocking **is not** yielding – using `omp_event_handle_t`
- Communication progression **[Buettner et al.]**

```
1   # pragma omp task depends(in: data)
2   {
3       [...]
4       MPI_Request request;
5       MPI_Isend(data, …, &request) ;
6       [...]
7       MPI_Wait(&request, …) ;
8       [...]
9    }
```

User-side hybrid MPI+OpenMP(tasks) code

```
11  /* Waits for an MPI requests to complete */
12  void MPI_Wait(MPI_Request * request, MPI_Status * status)
13  {
14      if (mpc_omp_in_explicit_task())
15      {
16
17
18
19
20
21
22
23
24
25          /* Block current task until its associated event is fulfilled */
26          mpc_omp_task_block(request->omp_event);
27
28       } else {
29            /* Standard MPI_Wait implementation [...] */
30       }
31   }
```

MPC MPI+OpenMP runtimes interoperation

[Buettner et al.] - Buettner, David & Acquaviva, Jean-Thomas & Weidendorfer, Josef. (2013). Real Asynchronous MPI Communication in Hybrid Codes through OpenMP Communication Tasks.

Implementation (2/2)

MPI+OpenMP interoperability

- The loss of thread issue → task switch
- Task blocking **is not** yielding – using `omp_event_handle_t`
- Communication progression **[Buettner et al.]**

```
50  int MPIX_Progress(MPI_Request * request)
51  {
52      int completed ;
53      MPI_Test(request, &completed);
54      if (completed)
55      {
56          omp_fulfill_event(request->omp_event);
57          return 1 ;
58      }
59      return 0 ;
60  }
```

MPC MPI+OpenMP communication progression

```
1   # pragma omp task depends(in: data)
2   {
3       [...]
4       MPI_Request request;
5       MPI_Isend(data, …, &request) ;
6       [...]
7       MPI_Wait(&request, …) ;
8       [...]
9    }
```

User-side hybrid MPI+OpenMP(tasks) code

```
11  /* Waits for an MPI requests to complete */
12  void MPI_Wait(MPI_Request * request, MPI_Status * status)
13  {
14      if (mpc_omp_in_explicit_task())
15      {
16
17
18
19
20
21
22          /* Register a callback in OpenMP runtime */
23          mpc_omp_callback(MPIX_Progress, request, MPC_OMP_TASK_SCHEDULE);
24
25          /* Block current task until its associated event is fulfilled */
26          mpc_omp_task_block(request->omp_event);
27
28      } else {
29          /* Standard MPI_Wait implementation [...] */
30      }
31  }
```

MPC MPI+OpenMP runtimes interoperation

[Buettner et al.] - Buettner, David & Acquaviva, Jean-Thomas & Weidendorfer, Josef. (2013). Real Asynchronous MPI Communication in Hybrid Codes through OpenMP Communication Tasks.

Implementation (2/2)

MPI+OpenMP interoperability

- The loss of thread issue → task switch
- Task blocking **is not** yielding – using `omp_event_handle_t`
- Communication progression **[Buettner et al.]**

```
50  int MPIX_Progress(MPI_Request * request)
51  {
52      int completed ;
53      MPI_Test(request, &completed);
54      if (completed)
55      {
56          omp_fulfill_event(request->omp_event);
57          return 1 ;
58      }
59      return 0 ;
60  }
```

MPC MPI+OpenMP communication progression

[Buettner et al.] - Buettner, David & Acquaviva, Jean-Thomas & Weidendorfer, Josef. (2013). Real Asynchronous MPI Communication in Hybrid Codes through OpenMP Communication Tasks.

```
1   # pragma omp task depends(in: data)
2   {
3       [...]
4       MPI_Request request;
5       MPI_Isend(data, …, &request) ;
6       [...]
7       MPI_Wait(&request, …) ;
8       [...]
9    }
```

User-side hybrid MPI+OpenMP(tasks) code

```
11  /* Waits for an MPI requests to complete */
12  void MPI_Wait(MPI_Request * request, MPI_Status * status)
13  {
14      if (mpc_omp_in_explicit_task())
15      {
16
17
18
19
20
21
22          /* Register a callback in OpenMP runtime */
23          mpc_omp_callback(MPIX_Progress, request, MPC_OMP_TASK_SCHEDULE);
24
25          /* Block current task until its associated event is fulfilled */
26          mpc_omp_task_block(request->omp_event);
27
28       } else {
29           /* Standard MPI_Wait implementation [...] */
30       }
31   }
```

MPC MPI+OpenMP runtimes interoperation

Implementation (2/2)

MPI+OpenMP interoperability

- The loss of thread issue → task switch
- Task blocking **is not** yielding – using `omp_event_handle_t`
- Communication progression **[Buettner et al.]**

```
50  int MPIX_Progress(MPI_Request * request)
51  {
52      int completed ;
53      MPI_Test(request, &completed);
54      if (completed)
55      {
56          omp_fulfill_event(request->omp_event);
57          return 1 ;
58      }
59      return 0 ;
60  }
```

MPC MPI+OpenMP communication progression

```
1   # pragma omp task depends(in: data)
2   {
3       [...]
4       MPI_Request request;
5       MPI_Isend(data, …, &request) ;
6       [...]
7       MPI_Wait(&request, …) ;
8       [...]
9    }
```

User-side hybrid MPI+OpenMP(tasks) code

```
11  /* Waits for an MPI requests to complete */
12  void MPI_Wait(MPI_Request * request, MPI_Status * status)
13  {
14      if (mpc_omp_in_explicit_task())
15      {
16
17
18
19
20
21
22          /* Register a callback in OpenMP runtime */
23          mpc_omp_callback(MPIX_Progress, request, MPC_OMP_TASK_SCHEDULE);
24
25          /* Block current task until its associated event is fulfilled */
26          mpc_omp_task_block(request->omp_event);
27
28       } else {
29           /* Standard MPI_Wait implementation [...] */
30       }
31   }
```

MPC MPI+OpenMP runtimes interoperation

[Buettner et al.] - Buettner, David & Acquaviva, Jean-Thomas & Weidendorfer, Josef. (2013). Real Asynchronous MPI Communication in Hybrid Codes through OpenMP Communication Tasks.

Implementation (2/2)

MPI+OpenMP interoperability

- The loss of thread issue → task switch
- Task blocking **is not** yielding – using `omp_event_handle_t`
- Communication progression [Buettner et al.]

```
50  int MPIX_Progress(MPI_Request * request)
51  {
52      int completed ;
53      MPI_Test(request, &completed);
54      if (completed)
55      {
56          omp_fulfill_event(request->omp_event);
57          return 1 ;
58      }
59      return 0 ;
60  }
```

MPC MPI+OpenMP communication progression

- FA send-tasks detection

```
1   # pragma omp task depends(in: data)
2   {
3       [...]
4       MPI_Request request;
5       MPI_Isend(data, …, &request) ;
6       [...]
7       MPI_Wait(&request, …) ;
8       [...]
9    }
```

User-side hybrid MPI+OpenMP(tasks) code

```
11  /* Waits for an MPI requests to complete */
12  void MPI_Wait(MPI_Request * request, MPI_Status * status)
13  {
14      if (mpc_omp_in_explicit_task())
15      {
16          /* Notify OpenMP that we are within a send task (FA) */
17          if (request->type == SEND)
18          {
19              mpc_omp_notify_send_operation();
20          }
21
22          /* Register a callback in OpenMP runtime */
23          mpc_omp_callback(MPIX_Progress, request, MPC_OMP_TASK_SCHEDULE);
24
25          /* Block current task until its associated event is fulfilled */
26          mpc_omp_task_block(request->omp_event);
27
28       } else {
29           /* Standard MPI_Wait implementation [...] */
30       }
31   }
```

MPC MPI+OpenMP runtimes interoperation

[Buettner et al.] - Buettner, David & Acquaviva, Jean-Thomas & Weidendorfer, Josef. (2013). Real Asynchronous MPI Communication in Hybrid Codes through OpenMP Communication Tasks.

Implementation (2/2)

MPI+OpenMP interoperability

- The loss of thread issue → task switch
- Task blocking **is not** yielding – using `omp_event_handle_t`
- Communication progression [Buettner et al.]

```
50  int MPIX_Progress(MPI_Request * request)
51  {
52      int completed ;
53      MPI_Test(request, &completed);
54      if (completed)
55      {
56          omp_fulfill_event(request->omp_event);
57          return 1 ;
58      }
59      return 0 ;
60  }
```

MPC MPI+OpenMP communication progression

- FA send-tasks detection

```
1   # pragma omp task depends(in: data)
2   {
3       [...]
4       MPI_Request request;
5       MPI_Isend(data, …, &request) ;
6       [...]
7       MPI_Wait(&request, …) ;
8       [...]
9    }
```

User-side hybrid MPI+OpenMP(tasks) code

```
11  /* Waits for an MPI requests to complete */
12  void MPI_Wait(MPI_Request * request, MPI_Status * status)
13  {
14      if (mpc_omp_in_explicit_task())
15      {
16          /* Notify OpenMP that we are within a send task (FA) */
17          if (request->type == SEND)
18          {
19              mpc_omp_notify_send_operation();
20          }
21
22          /* Register a callback in OpenMP runtime */
23          mpc_omp_callback(MPIX_Progress, request, MPC_OMP_TASK_SCHEDULE);
24
25          /* Block current task until its associated event is fulfilled */
26          mpc_omp_task_block(request->omp_event);
27
28       } else {
29           /* Standard MPI_Wait implementation [...] */
30       }
31   }
```

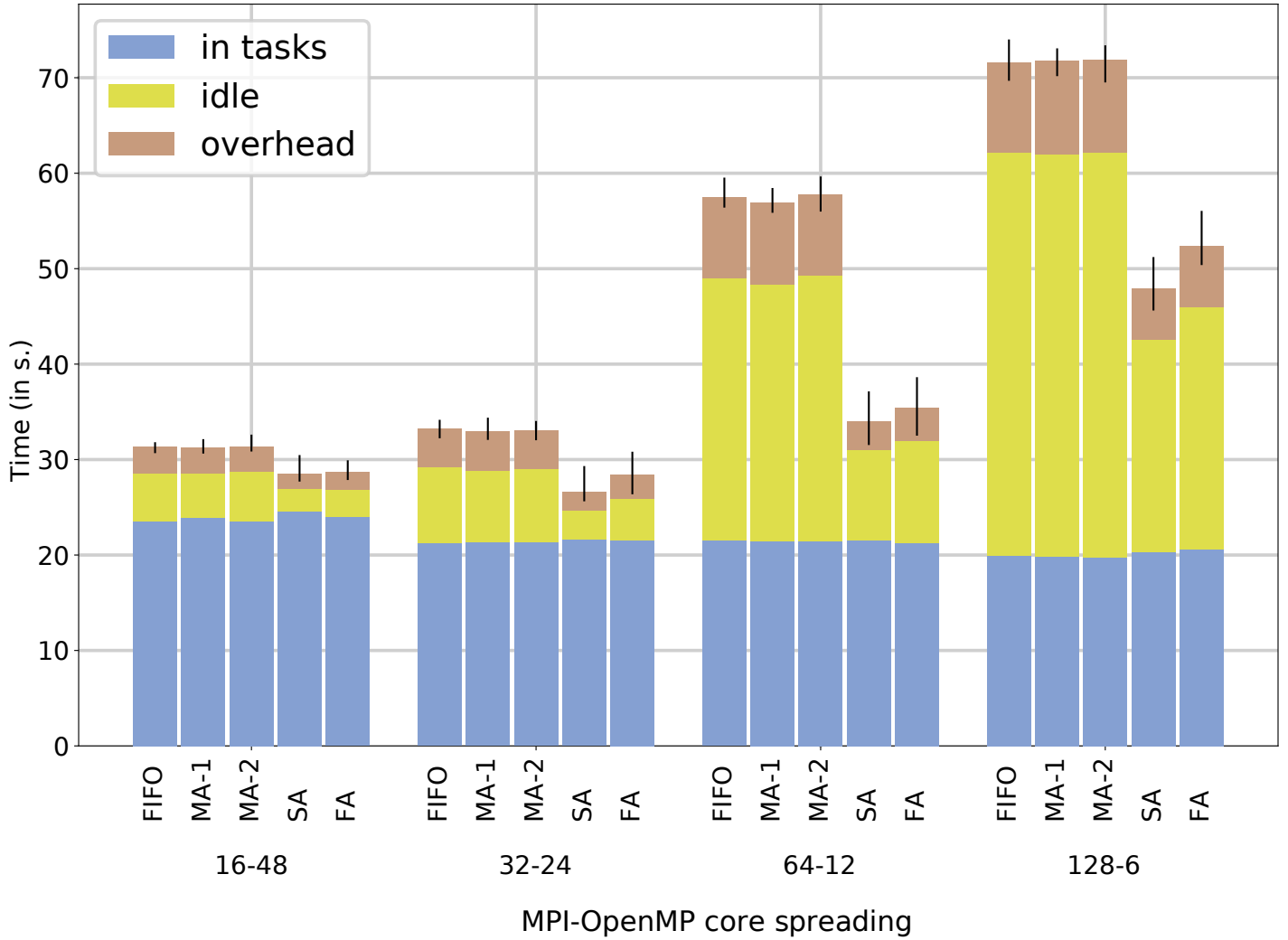
MPC MPI+OpenMP runtimes interoperation

[Buettner et al.] - Buettner, David & Acquaviva, Jean-Thomas & Weidendorfer, Josef. (2013). Real Asynchronous MPI Communication in Hybrid Codes through OpenMP Communication Tasks.

Evaluation

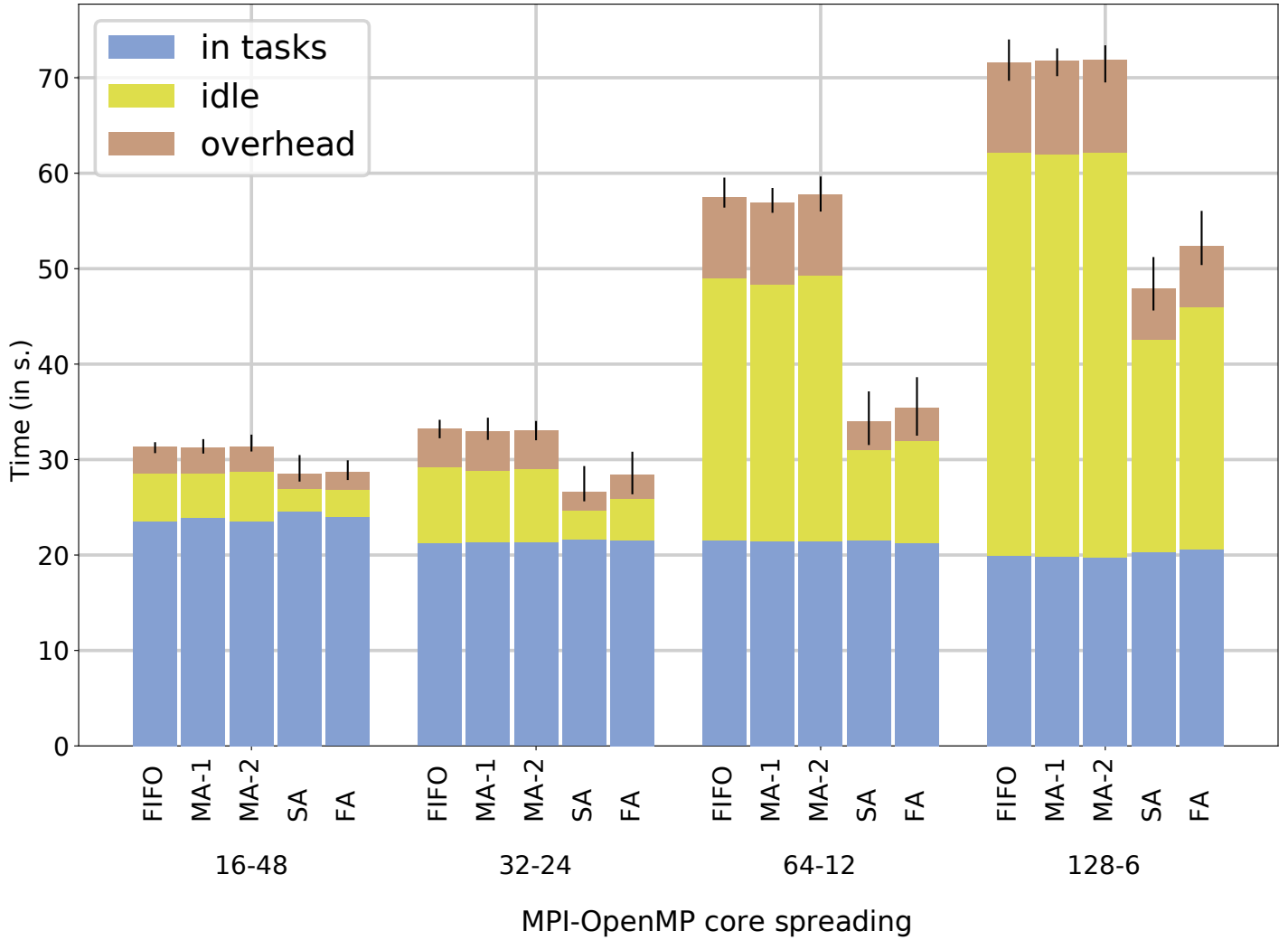
Core spreading over MPI+OpenMP

- Experimental Environnement
  - MPC-MPI + MPC-OpenMP
  - Skylake nodes – 16 nodes
    - Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
    - 2 sockets (NUMA) – 24 cores per socket
  - Cholesky matrix factorization – matrix of size n=131,072



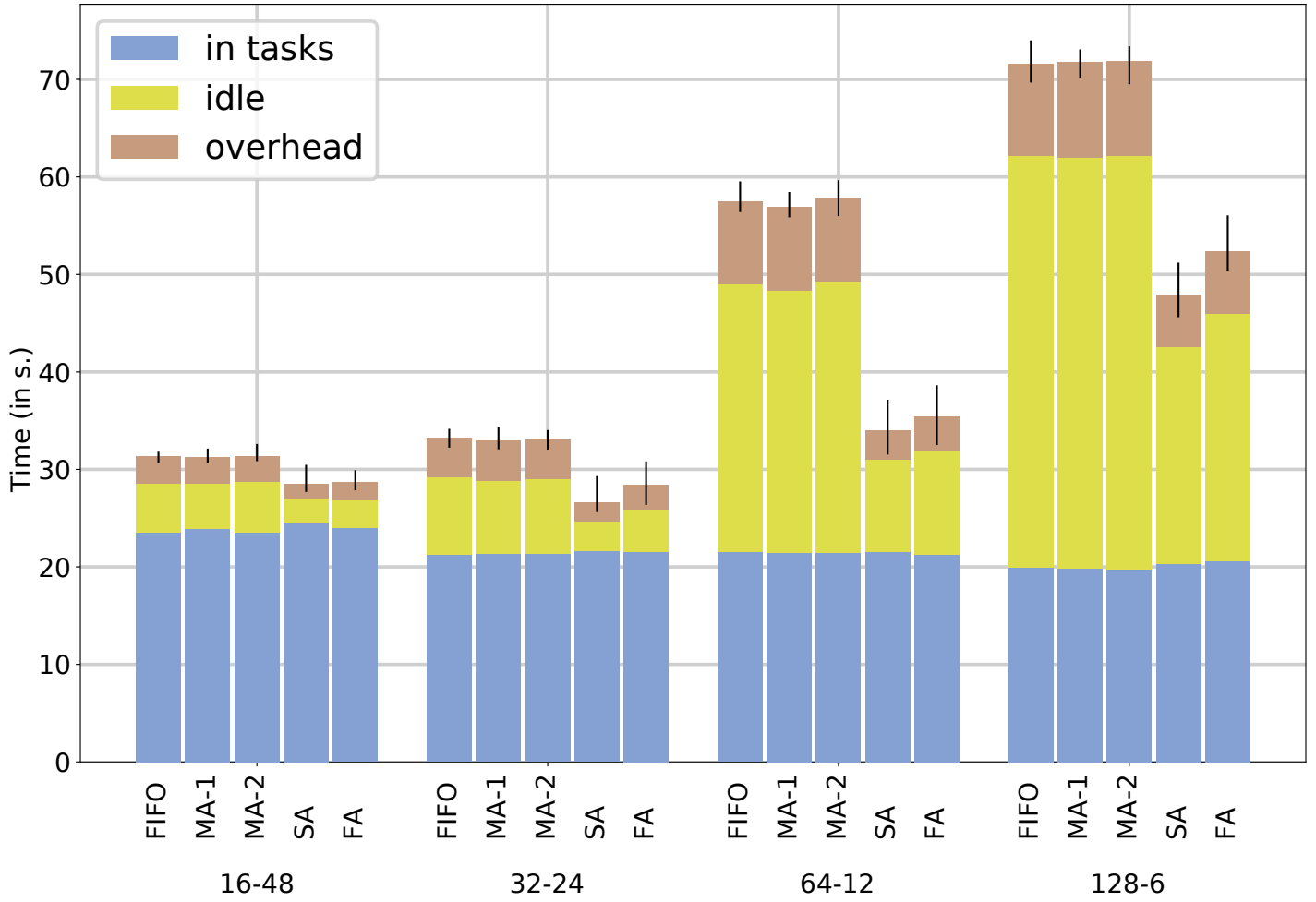
Core spreading over MPI+OpenMP

- Experimental Environnement
  - MPC-MPI + MPC-OpenMP
  - Skylake nodes – 16 nodes
    - Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
    - 2 sockets (NUMA) – 24 cores per socket
  - Cholesky matrix factorization – matrix of size n=131,072



Core spreading over MPI+OpenMP

- Experimental Environnement
  - MPC-MPI + MPC-OpenMP
  - Skylake nodes – 16 nodes
    - Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
    - 2 sockets (NUMA) – 24 cores per socket
  - Cholesky matrix factorization – matrix of size n=131,072



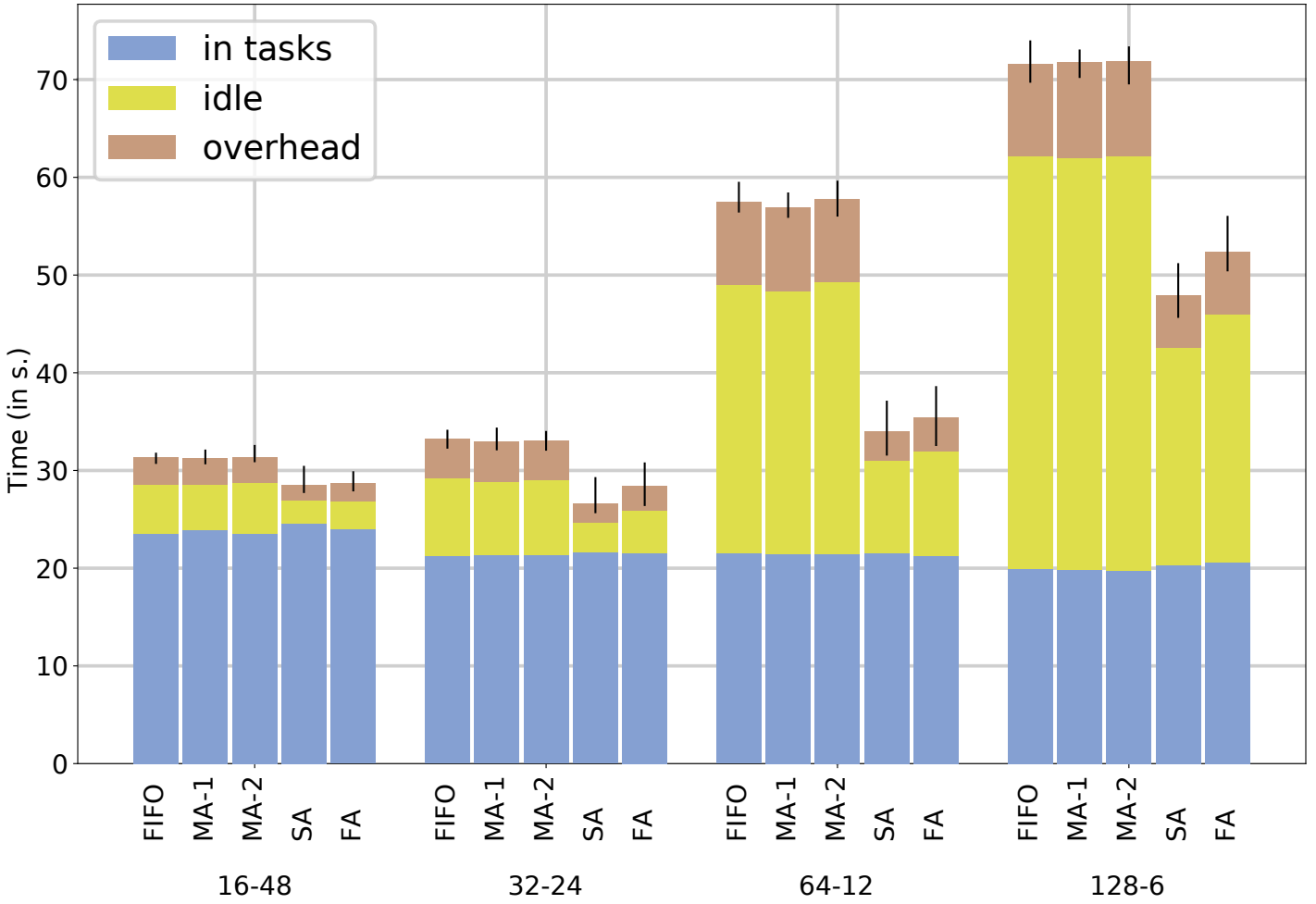
Core spreading (MPI - OpenMP)	16-48	32-24	64-12	128-6
P2P communication tasks (overall)	388,624	640,632	892,640	1,372,880

Number of point to point communication per spreading

Core spreading over MPI+OpenMP

- Experimental Environnement
  - MPC-MPI + MPC-OpenMP
  - Skylake nodes – 16 nodes
    - Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
    - 2 sockets (NUMA) – 24 cores per socket
  - Cholesky matrix factorization – matrix of size n=131,072

- Result
  - More communications + less threads →more idle time
  - Binary priorities are not sufficient (FIFO/MA1/MA2)
  - SA / FA re-ordering reduces idle time
  - Automation is costly: 3% to 8% performance loss



Core spreading (MPI - OpenMP)	16-48	32-24	64-12	128-6
P2P communication tasks (overall)	388,624	640,632	892,640	1,372,880

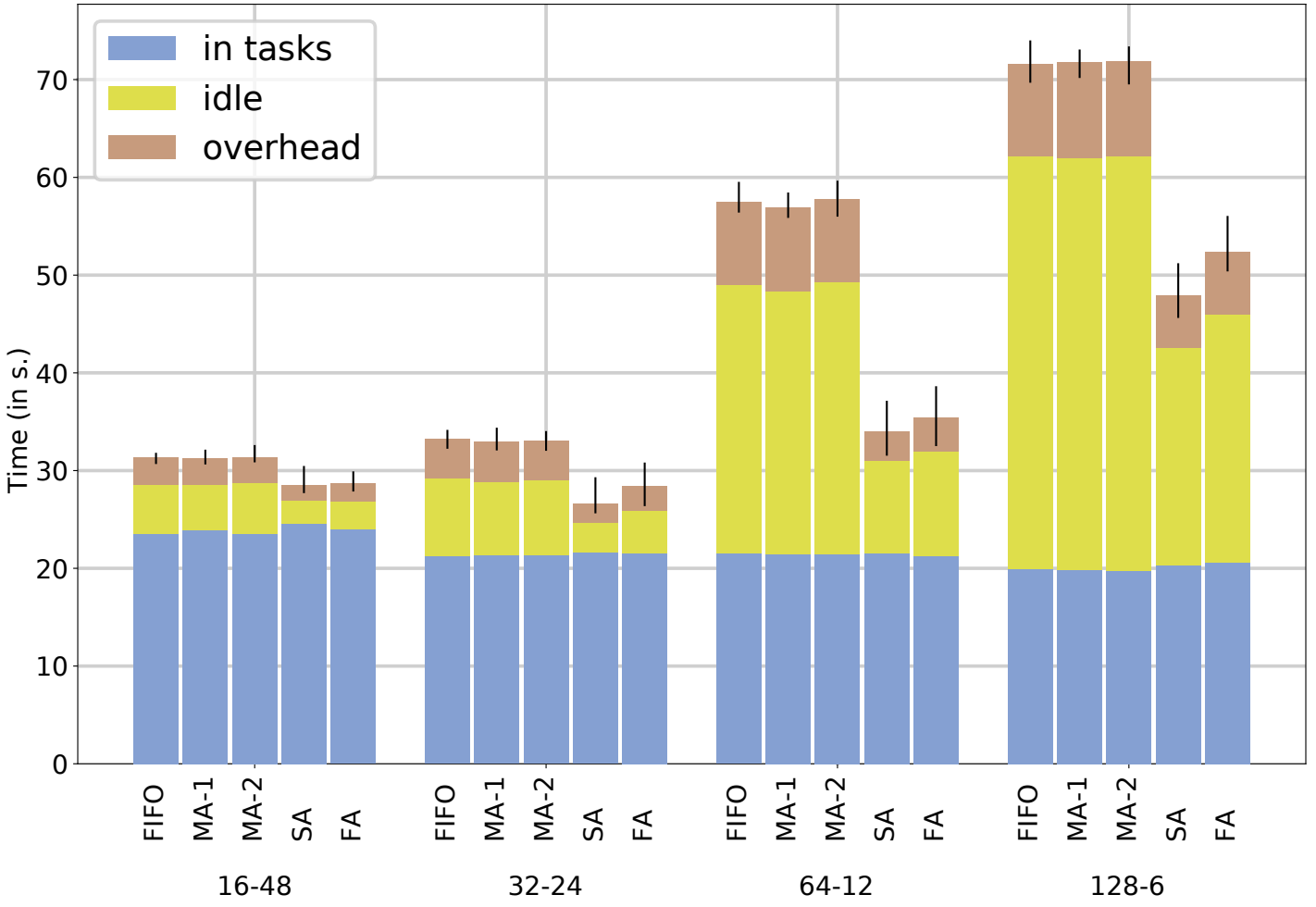
Number of point to point communication per spreading

Core spreading over MPI+OpenMP

- Experimental Environnement
  - MPC-MPI + MPC-OpenMP
  - Skylake nodes – 16 nodes
    - Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
    - 2 sockets (NUMA) – 24 cores per socket
  - Cholesky matrix factorization – matrix of size n=131,072

- Result
  - More communications + less threads →more idle time
  - Binary priorities are not sufficient (FIFO/MA1/MA2)
  - SA / FA re-ordering reduces idle time
  - Automation is costly: 3% to 8% performance loss

➡ Tasks re-ordering performance scaling

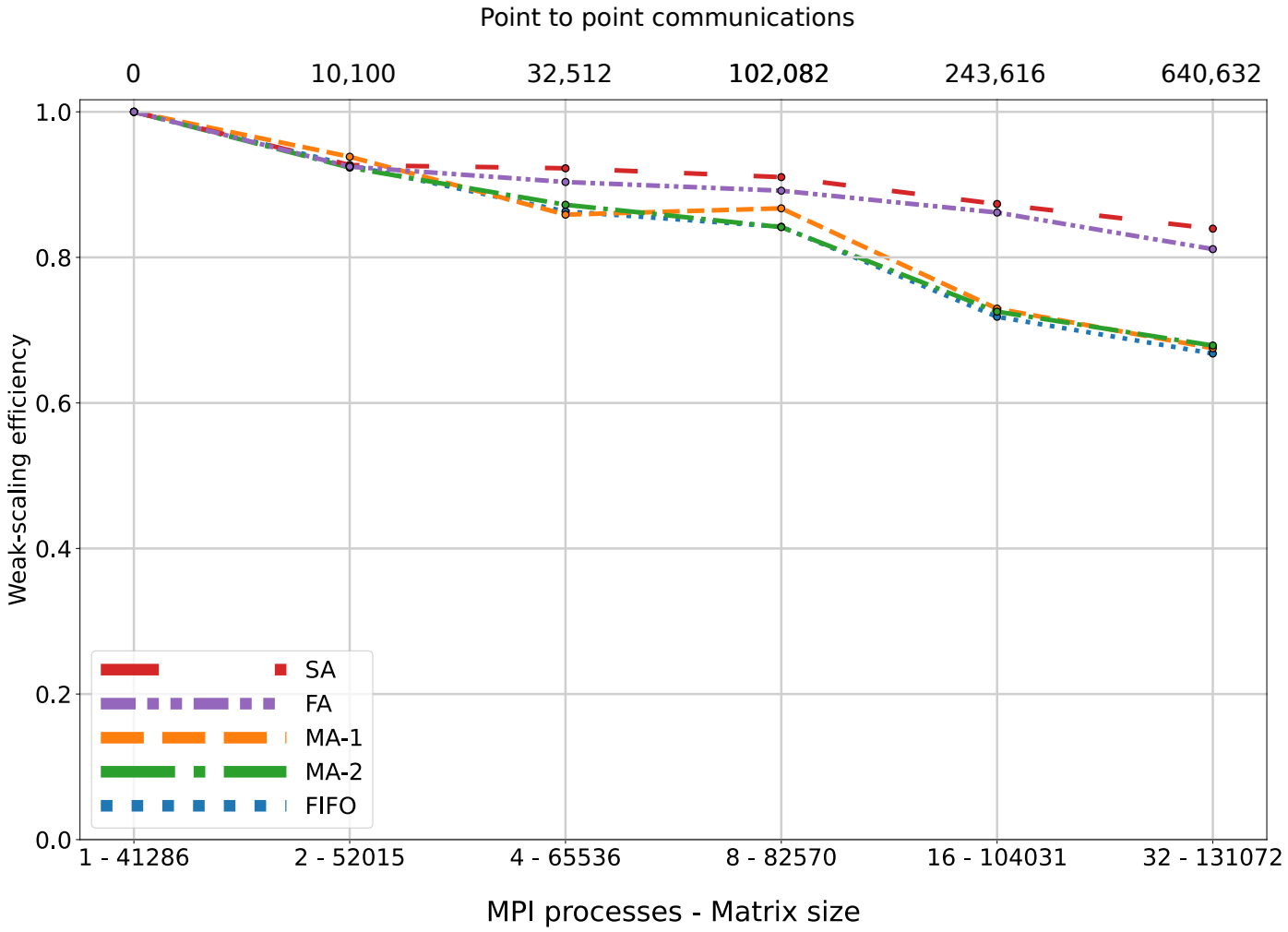


Core spreading (MPI - OpenMP)	16-48	32-24	64-12	128-6
P2P communication tasks (overall)	388,624	640,632	892,640	1,372,880

Number of point to point communication per spreading

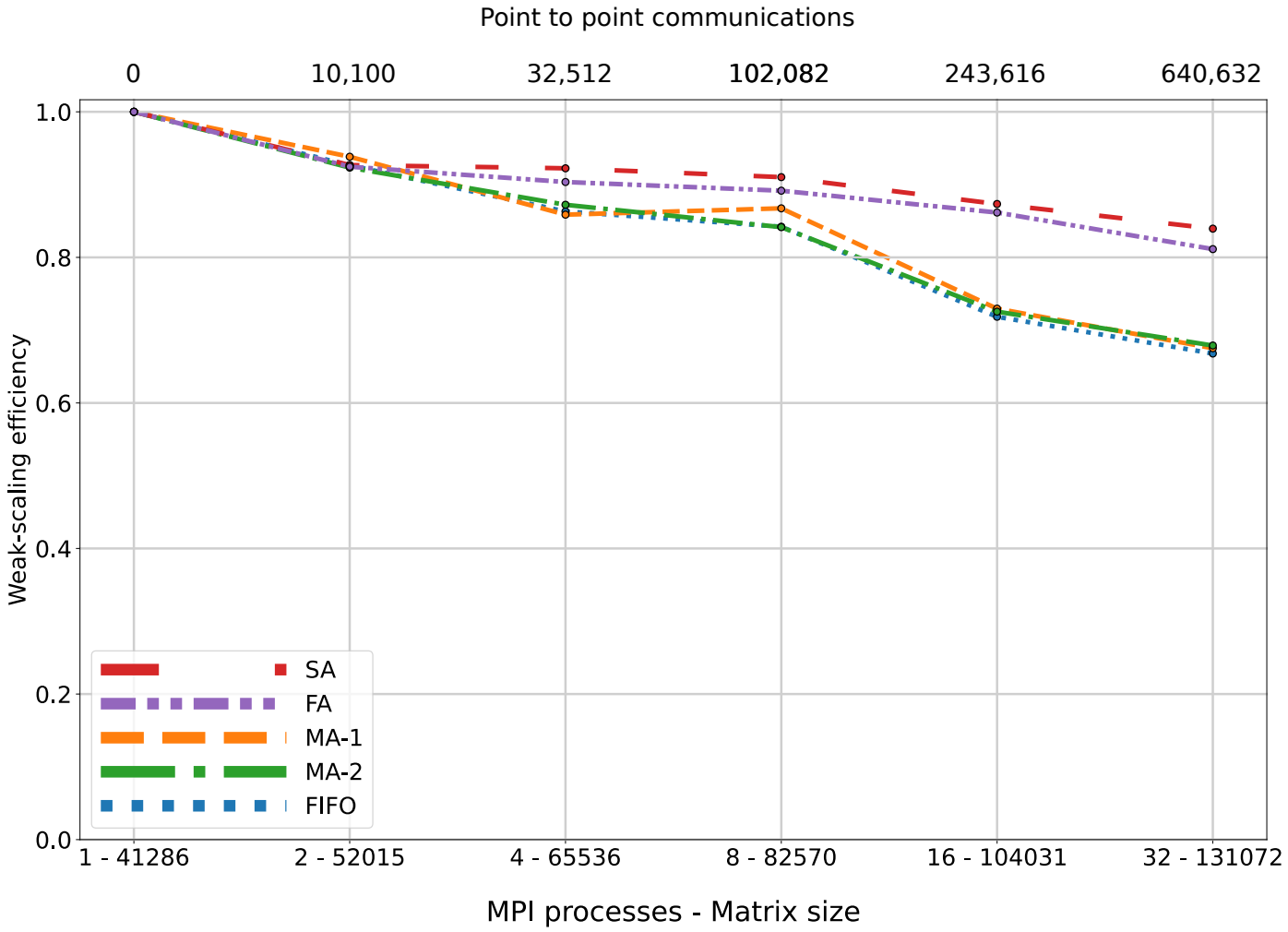
Weak-Scaling

- Experimental Environnement
  - MPC-MPI + MPC-OpenMP
  - Skylake nodes – 1 to 16 nodes
    - Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
    - 2 sockets (NUMA) – 24 cores per socket
  - Cholesky matrix factorization
    - Initial matrix of size n=41,286 for ~22.5 s.
  - 1 MPI process / processor (24 OMP threads)
  - Tasking
    - # computation task / process constant
    - # communication task increases



Weak-Scaling

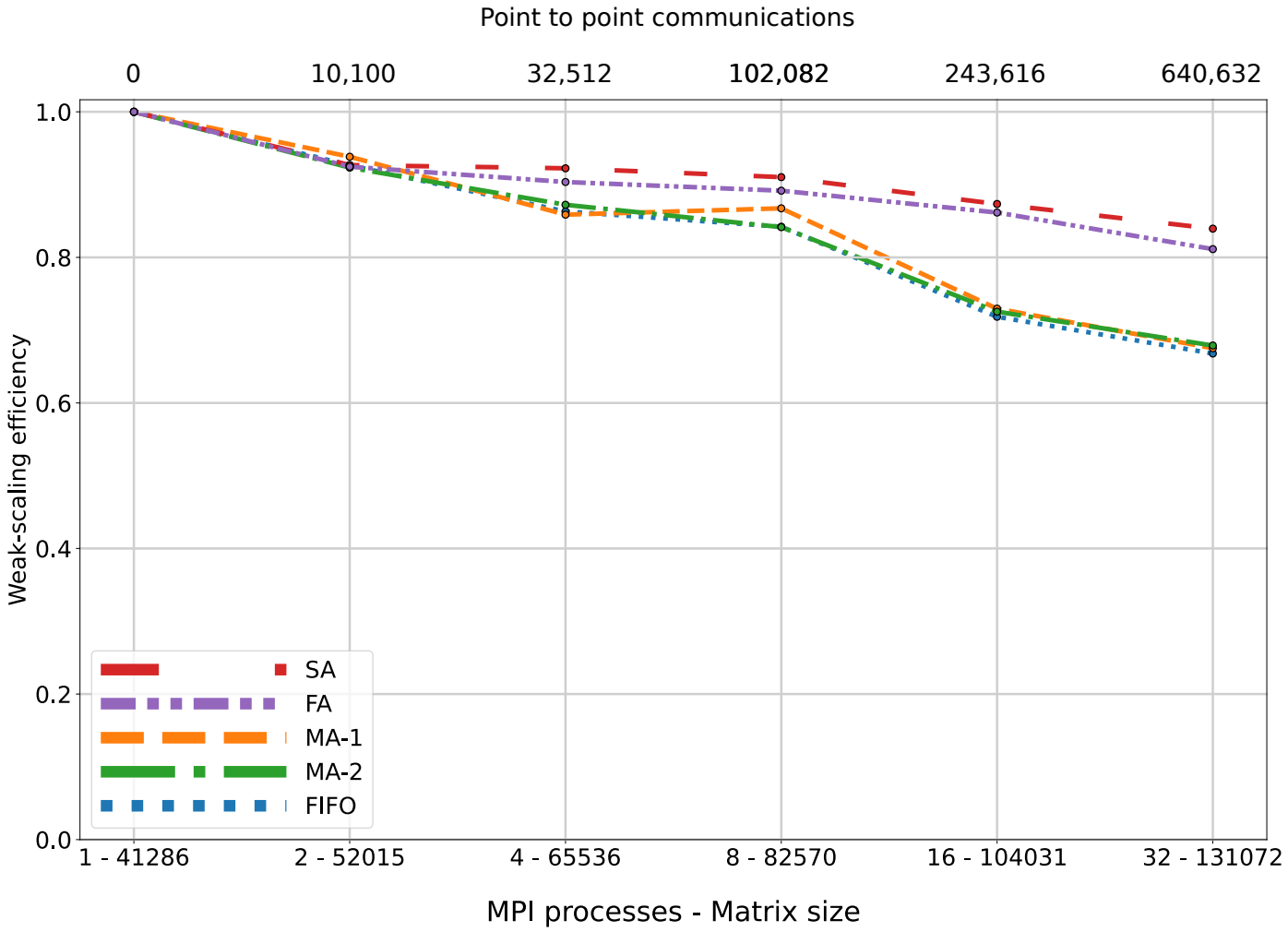
- Experimental Environnement
  - MPC-MPI + MPC-OpenMP
  - Skylake nodes – 1 to 16 nodes
    - Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
    - 2 sockets (NUMA) – 24 cores per socket
  - Cholesky matrix factorization
    - Initial matrix of size n=41,286 for ~22.5 s.
  - 1 MPI process / processor (24 OMP threads)
  - Tasking
    - # computation task / process constant
    - # communication task increases



Weak-Scaling

- Experimental Environnement
  - MPC-MPI + MPC-OpenMP
  - Skylake nodes – 1 to 16 nodes
    - Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
    - 2 sockets (NUMA) – 24 cores per socket
  - Cholesky matrix factorization
    - Initial matrix of size n=41,286 for ~22.5 s.
  - 1 MPI process / processor (24 OMP threads)
  - Tasking
    - # computation task / process constant
    - # communication task increases

- Result
  - SA/FA dampen efficiency loss
  - 32 processors efficiency: 84% (SA) – 66% (FIFO)



Conclusion

Conclusion

- MPI+OpenMP(tasks) programming
  - Interoperability issues
    - Loss of threads
    - Communication delay

→ Threads idling

Conclusion

- MPI+OpenMP(tasks) programming
  - Interoperability issues
    - Loss of threads
    - Communication delay

→ Threads idling

- Contribution: a task scheduling strategy for applications with MPI Communications
  - Favoring send-tasks
    - Different policies
    - Different approaches (user-code intrusivity / runtime automation)
  - Implementation within MPC

→ Reduces idle time, improve performance scaling

Conclusion

- MPI+OpenMP(tasks) programming
  - Interoperability issues
    - Loss of threads
    - Communication delay

→ Threads idling

- Contribution: a task scheduling strategy for applications with MPI Communications
  - Favoring send-tasks
    - Different policies
    - Different approaches (user-code intrusivity / runtime automation)
  - Implementation within MPC

→ Reduces idle time, improve performance scaling

- Future works
  – More applications, larger systems
  – Adjust policy for communication intensive applications
    - Bandwidth saturation
    - Post receives to limit buffer copies
  – Vision limited to local subgraph → Take into account remote processes task dependency graph

Thank you for your attention

Multi-Processor Computing (MPC) – Patched version of this work

- Repository – https://mpc.hpcframework.com/mpc/releases/mpcframework-4.0.0-iwomp.tar.gz
- Compile and prepare environnement – see the **README** file within the archive

Cholesky Factorization Application

- Repository – https://gitlab.inria.fr/ropereir/iwomp2021
- Commit – 88793c08
- Compile and prepare environnement – see the **README** file

For any help or issues, feel free to contact us.
We will gladly help you.

Romain PEREIRA – romain.pereira@cea.fr
Adrien ROUSSEL – adrien.roussel@cea.fr
Patrick CARRIBAULT – patrick.carribault@cea.fr
Thierry GAUTIER – thierry.gautier@inrialpes.fr