An Overview on Mixing MPI and OpenMP Dependent Tasking on Fugaku

IWAHPCE-2024 – January 25th, 2024

Romain PEREIRA [1], **Adrien ROUSSEL** [1,2], Miwako TSUJI [3], Patrick CARRIBAULT [2], Hitoshi MURAI [3], Mitsuhisa SATO [3], Thierry GAUTIER [4]

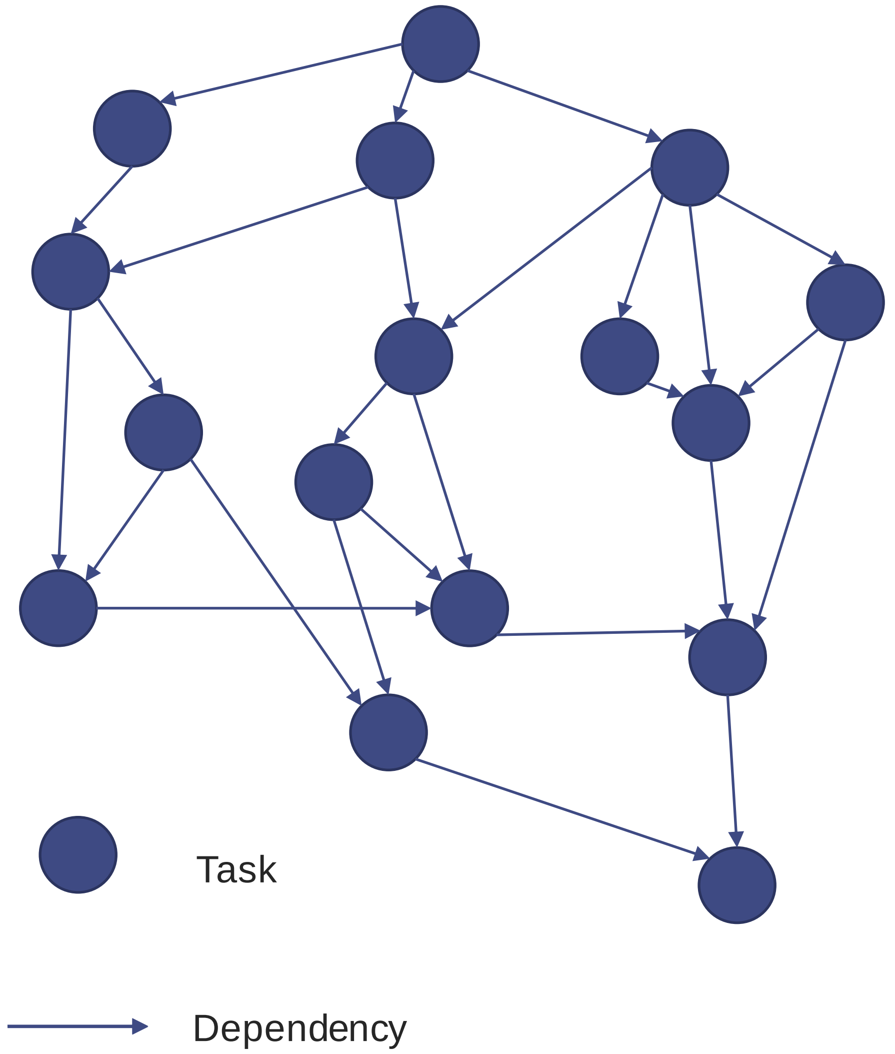
[1] Exascale Computing Research Laboratory, France
[2] CEA, Laboratoire en Informatique Haute Performance pour le Calcul et la Simulation, France
[3] RIKEN R-CCS, Japan
[4] Avalon, LIP, ENS, INRIA, France

1. Context

a. MPI+OpenMP Task-Based Programming
b. Software Stack on Fugaku
c. Problem Definition

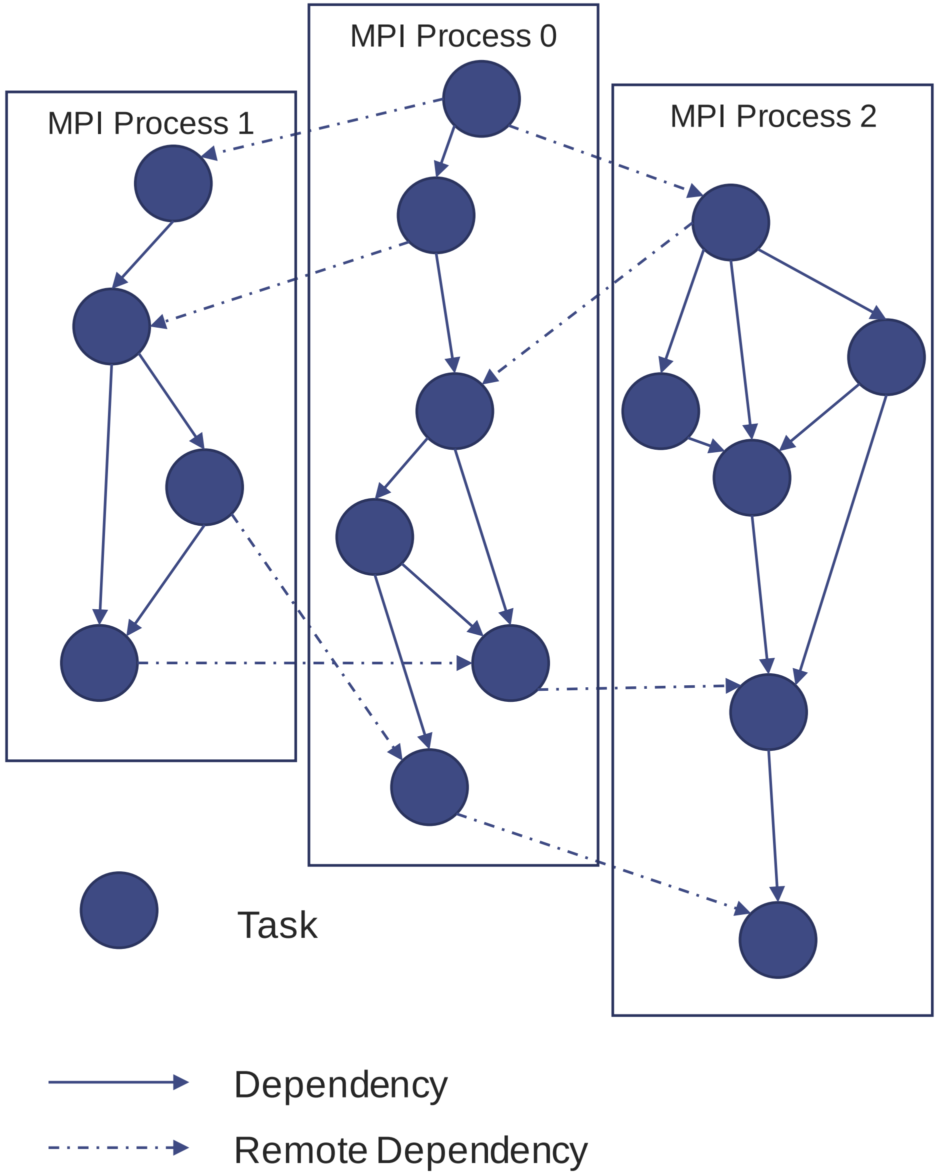
Task Programming Model

- **Task Programming Model (TPM)**
  - Candidate towards **Performance-Portable Applications**
  - Task Dependency Graph (TDG)
    - Node → set of instructions (tasks)
    - Edges → tasks precedence constraints (dependencies)



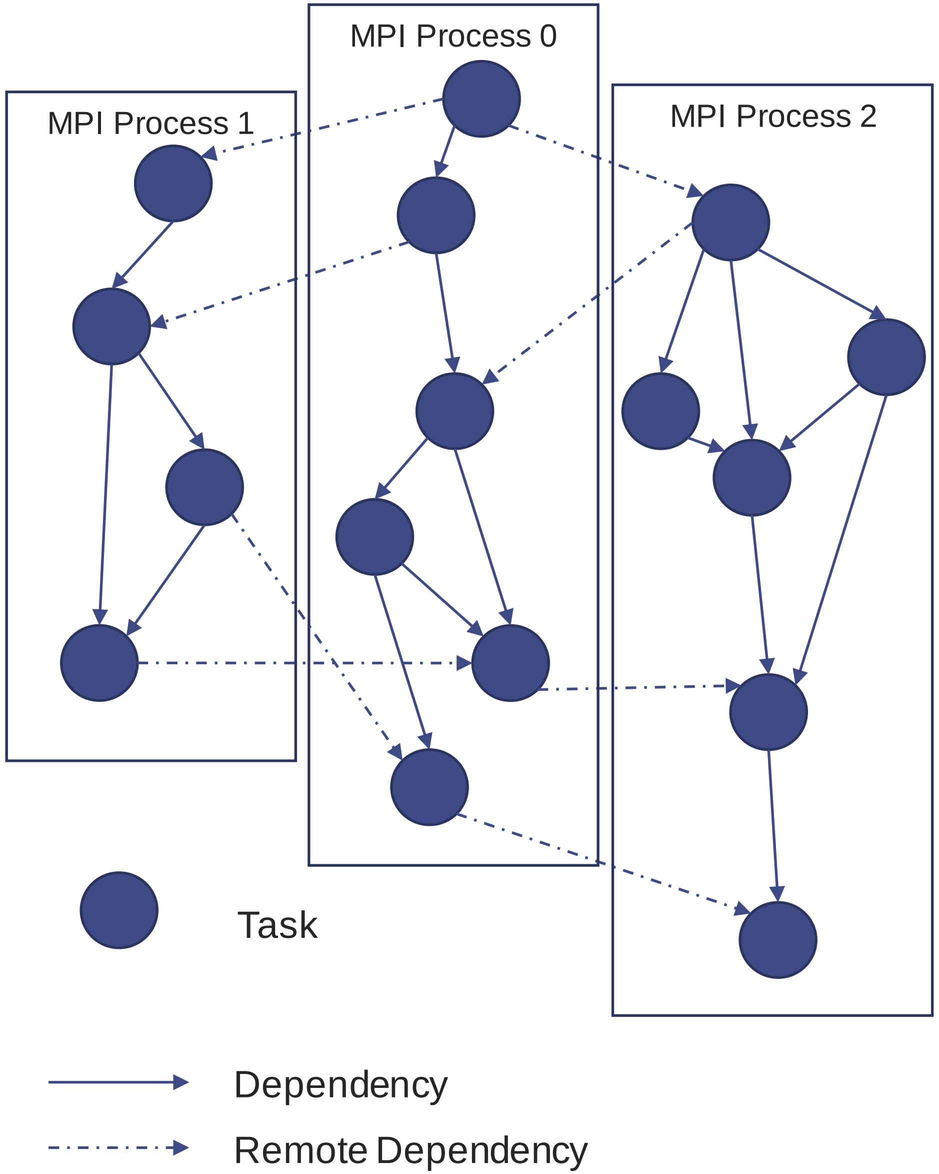
Task Programming Model

- **Task Programming Model (TPM)**
  - Candidate towards **Performance-Portable Applications**
  - Task Dependency Graph (TDG)
    - Node → set of instructions (tasks)
    - Edges → tasks precedence constraints (dependencies)
- **Hybrid and Heterogeneous MPI+OpenMP TPM**
  - User TDG partitionning
  - Local OpenMP **subgraphs**
  - Remote dependencies → MPI communications



Task Programming Model

- **Task Programming Model (TPM)**
  - Candidate towards **Performance-Portable Applications**
  - Task Dependency Graph (TDG)
    - Node → set of instructions (tasks)
    - Edges → tasks precedence constraints (dependencies)
- **Hybrid and Heterogeneous MPI+OpenMP TPM**
  - User TDG partitionning
  - Local OpenMP **subgraphs**
  - Remote dependencies → MPI communications
- Lessons learned from performance analysis
  - Highly sensitive to Supercomputer Architecture
  - Needs fine tuning on software stack
- ➢ **What about Fugaku?**



Software Stack on Fugaku

- Compilers
  - GCC
  - LLVM

- Runtimes
  - MPI
    - MPICH-TOFU
  - OpenMP
    - GOMP
    - LLVM-OpenMP
    - MPC-OpenMP

- Task-based Applications
  - Cholesky Factorization
  - HPCG
  - LULESH

Software Stack on Fugaku

- Compilers
  - GCC
  - LLVM

- Runtimes
  - MPI
    - MPICH-TOFU
  - OpenMP
    - GOMP
    - LLVM-OpenMP
    - MPC-OpenMP

- Task-based Applications
  - Cholesky Factorization
  - HPCG
  - LULESH

What's the impact on performances of each element?

2. Performance Study

a. Single Node

b. Multi-Nodes

Studied Software Stack Variations

- Summary:

Hardware	A64FX and TofuD Interconnect (Fugaku)
Compiler	LLVM/release/17.x (clang17), GCC/releases-13 (gcc)
OpenMP Runtime	LLVM/release/17.x (kmp), GCC/releases-13 (gomp), MPC (mpc)
MPI Runtime	MPICH-Tofu
MPI Request Progression Engine	MPI-Detach thread (mpi-detach), MPC-OMPT (mpc-ompt)
MPI Request Task Scheduling	Default (no-priority), Early-bird Posting (priority)
Applications	Cholesky, HPCCG, LULESH

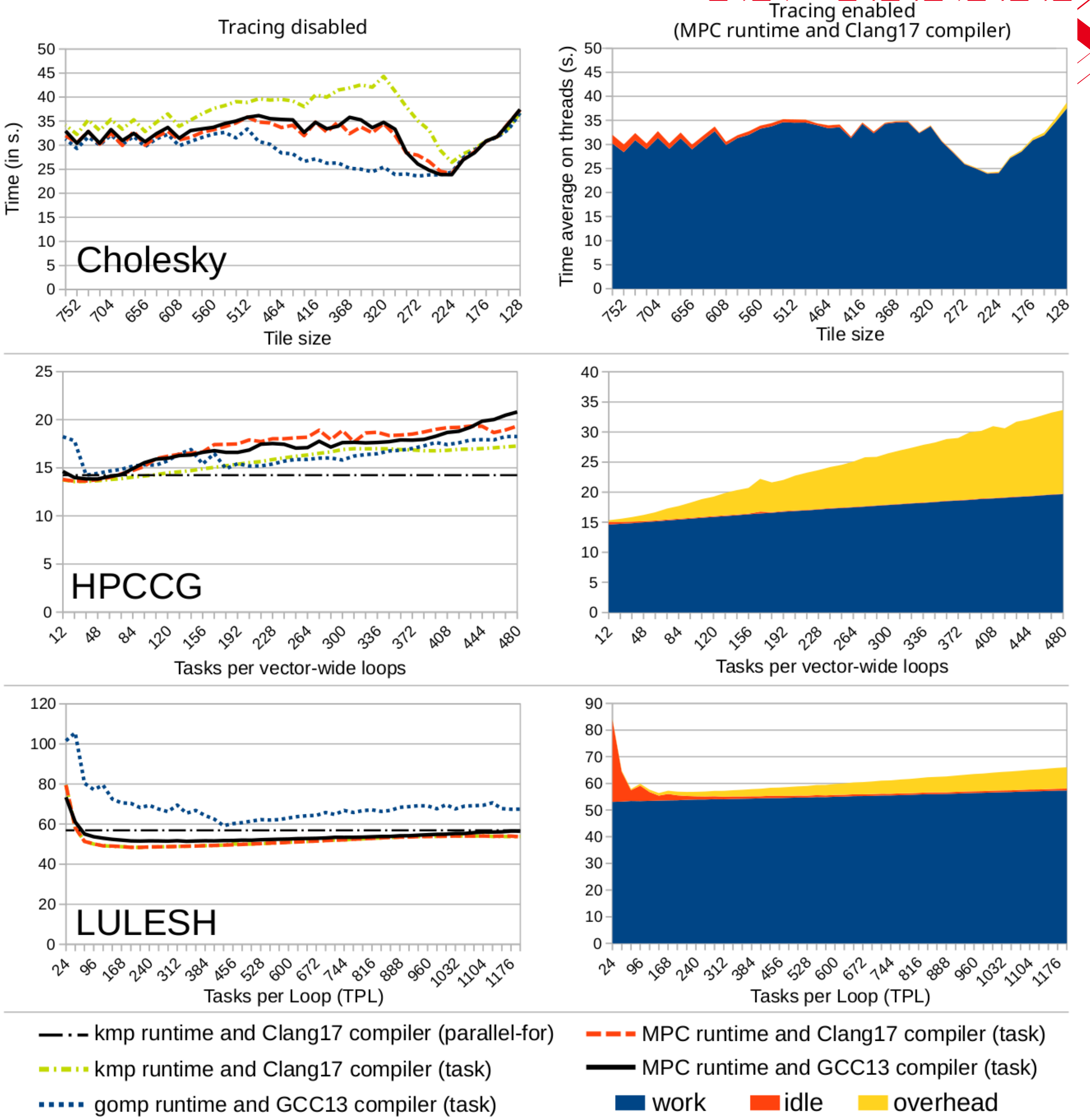
Single Node Analysis

- Task Grain Analysis
  - Problem size fixed
  - From left to right: increase in number of tasks

- Threads Placement and Configuration
  - 1 CMG – 12 cores

- Main Observations
  - Slight differences between runtimes, e.g.,
    - Cholesky perform better with GOMP, but
    - it performs worst than other on LULESH

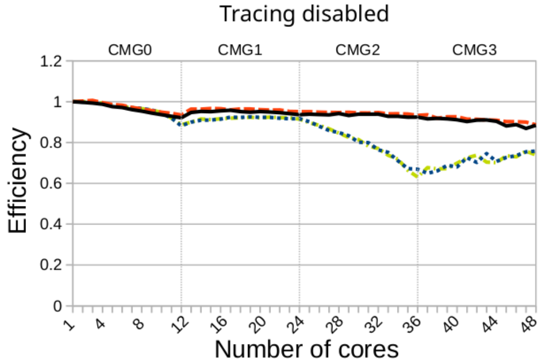
  - Overhead increases
    - Many small tasks are ready, but cannot be performed immediately

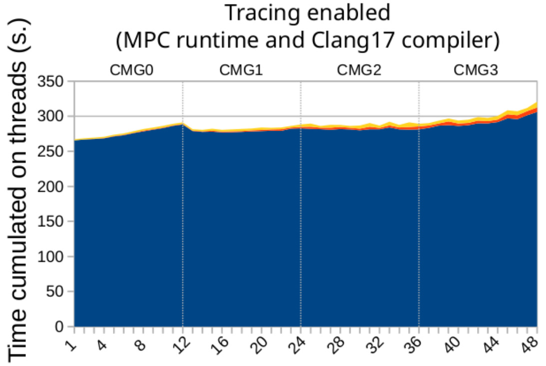


Single Node Efficiency

Cholesky

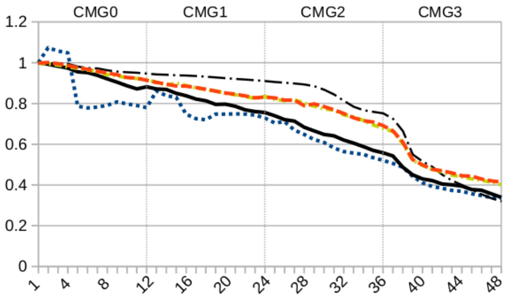
	Time (in s.)			
	GCC13		Clang17	
Cores	gomp(task)	mpc(task)	kmp(task)	mpc(task)
1	282.96	265.05	282.81	265.05
12	**26.73**	**23.88**	**26.58**	**23.62**
24	12.86	11.75	12.88	11.61
36	11.74	7.93	12.45	7.89
48	7.76	6.22	7.98	6.23

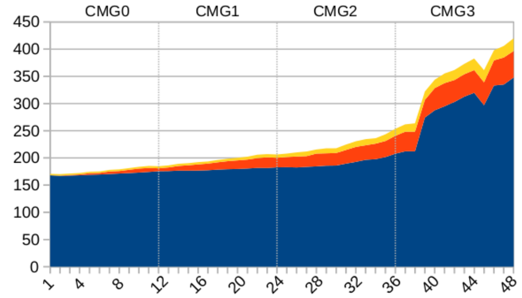




HPCCG

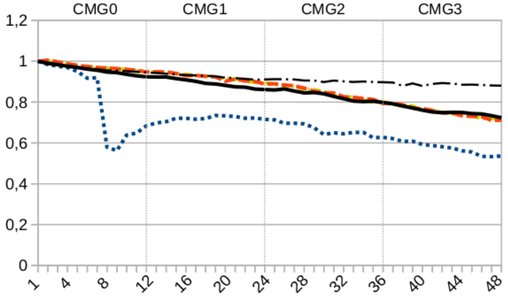
	Time (in s.)				
	GCC13		Clang17		
Cores	gomp(task)	mpc(task)	kmp(parallel-for)	kmp(task)	mpc(task)
1	170.72	154.60	166.99	164.81	164.90
12	**18.24**	**14.61**	**14.68**	**14.93**	**15.00**
24	9.77	8.52	7.63	8.21	8.24
36	9.10	7.69	6.16	6.73	6.60
48	10.61	9.49	10.84	8.51	16.24

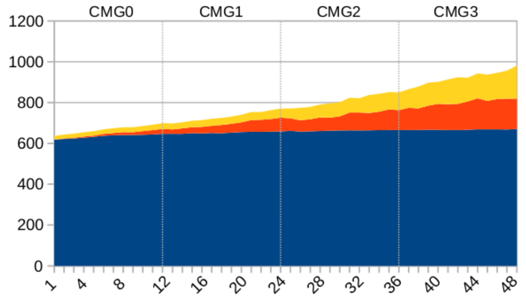




LULESH

	Time (in s.)				
	GCC13		Clang17		
Cores	gomp(task)	mpc(task)	kmp(parallel-for)	kmp(task)	mpc(task)
1	580.32	577.10	654.89	554.54	554.68
12	**70.66**	**52.06**	**57.74**	**48.66**	**48.78**
24	33.8	27.91	29.96	25.8	26.01
36	25.7	20.08	20.28	19.2	19.41
48	22.56	16.62	15.5	16.16	16.24





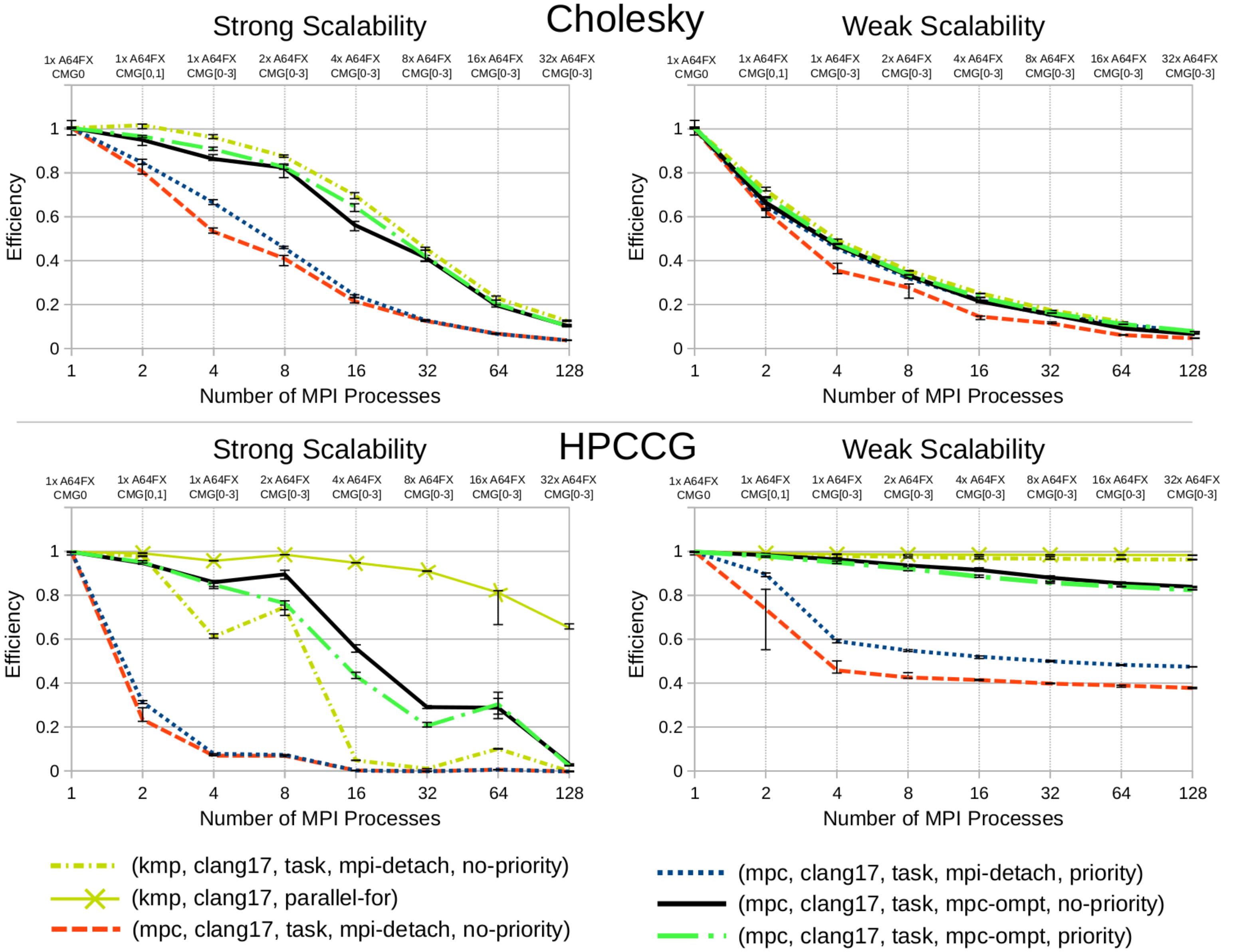
kmp runtime and Clang17 compiler (parallel-for)
kmp runtime and Clang17 compiler (task)
gomp runtime and GCC13 compiler (task)
MPC runtime and Clang17 compiler (task)
MPC runtime and GCC13 compiler (task)
work idle overhead

Focus: MPI Communication Progression

- 2 ≠ ways to progress communications
  - MPI Detach
    - MPI Progression is done by a **dedicated thread scheduled periodically**
  - MPC-OMPT
    - Progression engine **called at each OpenMP scheduling point**
    - **Opportunistic** method
    - Only works with MPC runtime

Multi-Nodes Performances

- Configuration
  - 1 MPI rank per CMG (i.e. 12 cores)
    - 4 MPI proc. per node
    - Up to 32 nodes
- Main Observations
  - Communication Progression Engine has a significant impact for MPC
    - Opportunistic way is better
  - Results fluctuations for strong scalability on HPCCG
    - Need further investigations



3. Conclusion

Conclusion

- Variation study of the software stack on Fugaku for task-based applications
  - Compilers
  - Runtimes
  - Applications

- Performance analysis obtained on up to 32 Fugaku nodes

- Main conclusions
  - Some noticeable differences between runtimes are observed
  - Communication progression engine significantly affects performances
  - Threads placement too

Future Work

- Lulesh with MPI
  - Fails whatever the OpenMP runtime is
  - But different kind of errors (race condition on GOMP, deadlock on LLVM)
  - ➢ Needs further investigations

Thank you for your attention

Questions?

Adrien Roussel, PhD
adrien.roussel@cea.fr
MPC Team