Preliminary Evaluation towards Task Priority Control in HPX

TOHOKU UNIVERSITY
Cyberscience Center

Suhang Jiang, Mulya Agung, Ryusuke Egawa and Hiroyuki Takizawa

Background: Task-based and loop-based

Recently, **task-based** execution has been attracting attention, because it can reduce the waiting time from synchronization.

➢ **Loop-based:**
- Exploit data parallelism, and execute the same computation on different data.
- Use barrier between jobs to make sure the loop execution is already finished.

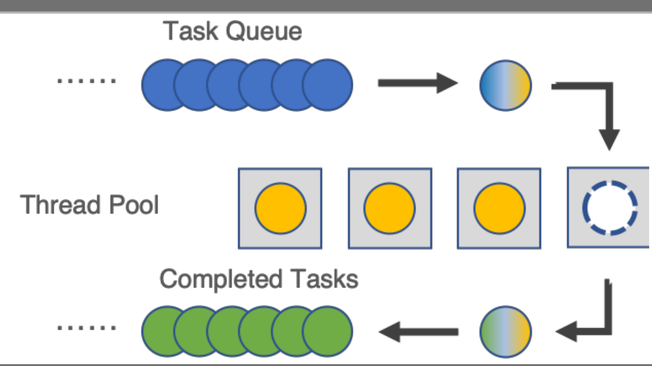
➢ **Task-based:**
- Exploit task parallelism, and execute different tasks in parallel.
- Use tasks and their dependencies to control the order of task execution. A directed acyclic graph (DAG) is used to express the dependency.

High Performance ParalleX (HPX)

- A runtime system for parallel computing based on the partitioned global address space (PGAS) model
- Provides a C++ class library to describe tasks and their dependencies

Task Priority Control

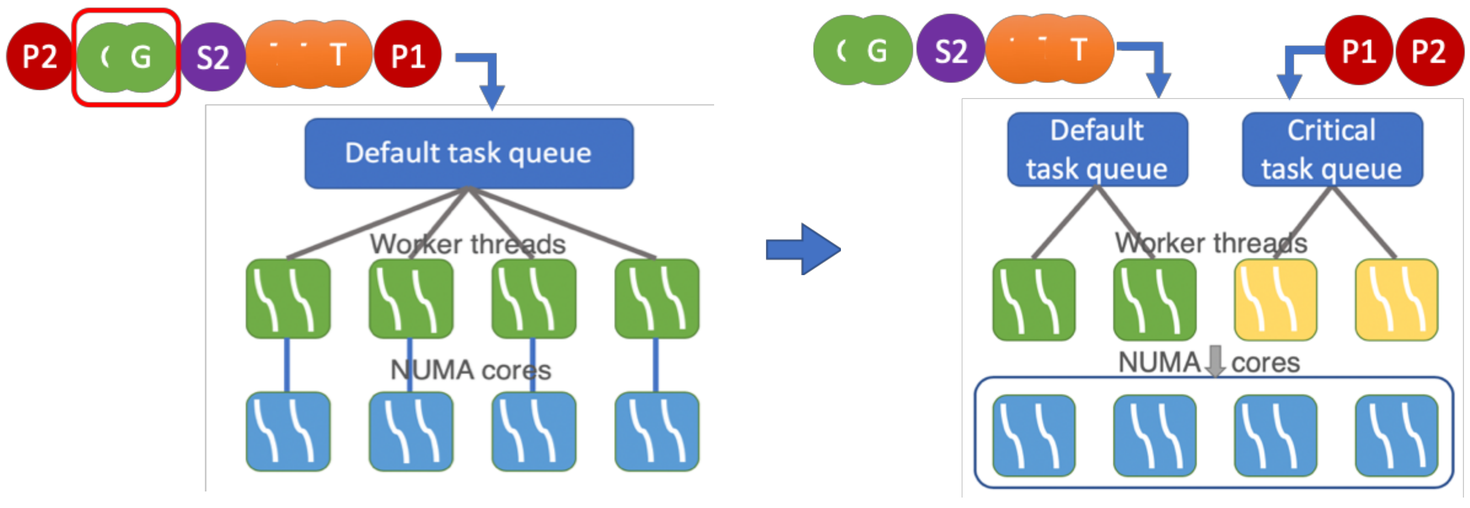
- All tasks are managed using a single default task queue.
- Tasks will be assigned to HPX threads in a thread pool in FIFO, every task has the same execution priority.



Problem & Motivation

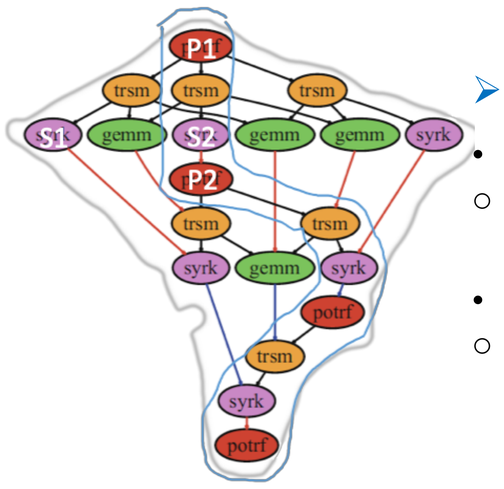
- The newest version of OpenMP supports task priority, resulting in higher performance.
- For multi-node parallel processing systems such as HPX runtime system, there is no task priority control until now.

Proposed Approach



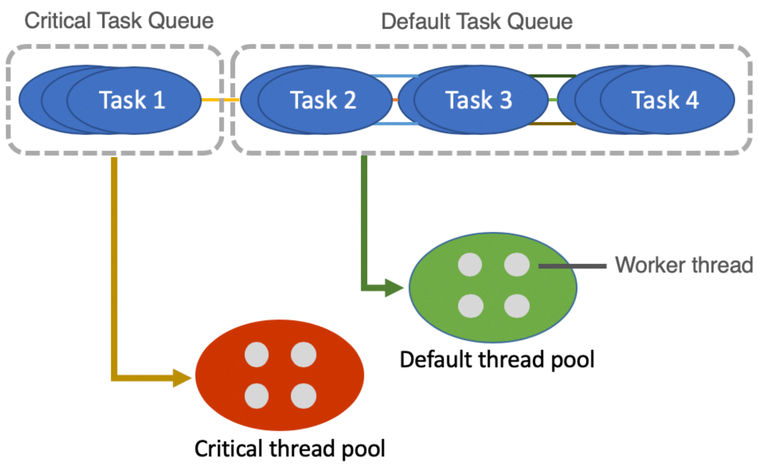
➢ Only one default task queue:
- Tasks have the same priority, and wait in the task queue in a FIFO fashion.
- Tasks will be executed as long as the dependencies among tasks are not violated.
- The critical task can be blocked by other non-critical tasks.

➢ Two decoupled task queues:
- Choose the critical task.
- Set decoupled task queues and thread pools.
- Map the threads in cores.

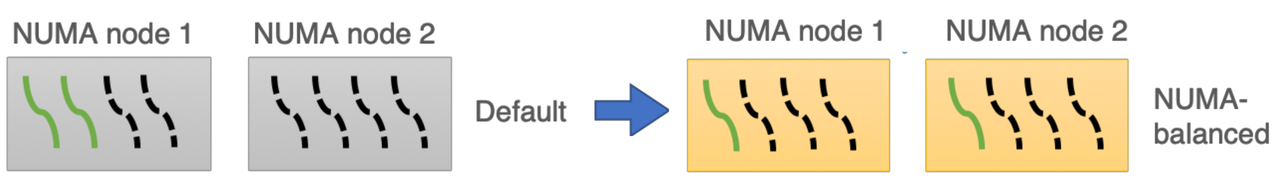


➢ STEP 1: Choose the critical task.

- Find the critical path in the DAG.
  - The longest path in the DAG, which has more tasks than the other paths.
- Find the critical task in the critical path
  - The task(s) that have the highest number of occurrences in the critical path.



➢ STEP 2: Decoupled task queues and thread pools.
- A higher priority is given to critical tasks in the critical task queue.
- Tasks from the critical task queue go to the critical thread pool, and the other tasks go to the default thread pool.
- Whenever a worker thread becomes idle, an HPX thread is retrieved from the thread pool, and assigned to the worker thread.



➢ STEP 3: Using an NUMA-balanced method for mapping tasks.
- Different from the default thread mapping method, the NUMA-balanced method will assign threads evenly to cores.
- More efficient by making better use of the resources.

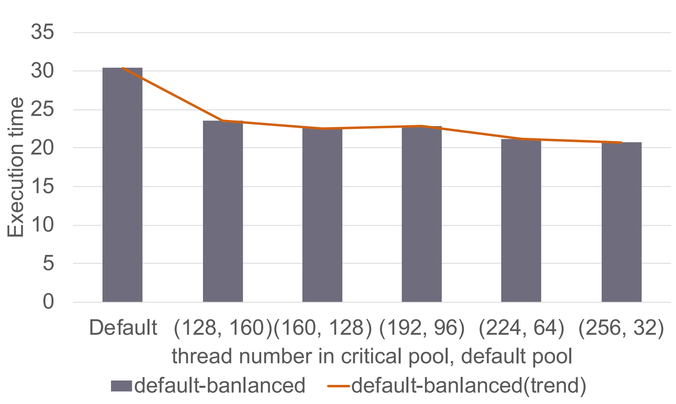
Evaluation Results

➢ Environment: Intel Xeon Phi Knight Landing

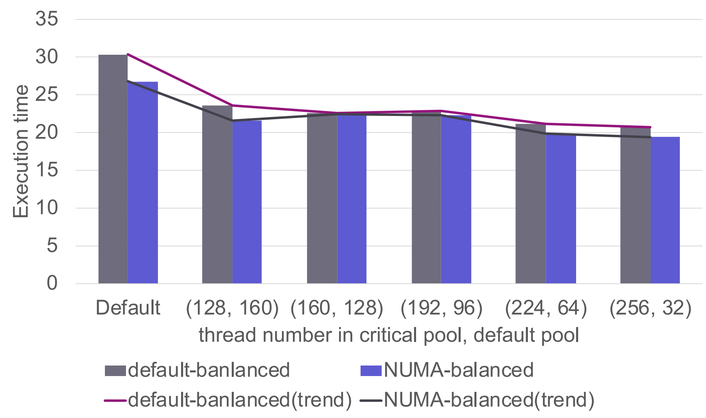
➢ Benchmark program: Cholesky factorization

- A decomposition of a Hermitian positive-definite matrix A is a decomposition into the form of $A=LL^{\mathsf{T}}$

  - By increasing the number of threads in the critical pool manually, the proposed implementation can increase the performance by 31.76% in terms of execution time.
  - Based on decoupled task queues, using the NUMA-balanced thread mapping method can further increase the performance by 4.8%.



The execution time using decoupled task queues in Intel Xeon Phi KNL, matrix size = 512 * 512



The execution time using decoupled task queues and NUMA-balanced thread mapping in Intel Xeon Phi KNL, matrix size = 512 * 512

Conclusions and Future work

➢ This work proposed an approach to more efficient task-based execution based on prioritizing critical tasks.

1. An algorithm is proposed to choose the critical task from the directed acyclic graph(DAG).
2. Decoupled task queues and thread pools can increase the performance.
3. NUMA-balanced method can further increase the performance.

➢ Future work:
- A different task-based application may have more critical tasks. Thus, different applications with more critical tasks will be evaluated, so that **using three or more tasks queues could potentially help** achieving higher efficiency.
- **Different thread mapping methods** will also be evaluated.

References and Acknowledgement

[1] Kaiser, Hartmut, et al. "HPX: A task based programming model in a global address space." Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models. ACM, 2014.
[2] Grubel, Patricia, et al. "The performance implication of task size for applications on the HPX runtime system." 2015 IEEE International Conference on Cluster Computing. IEEE, 2015.
[3] Cayrols, Sébastien, Iain Duff, and Florent Lopez. "Parallelization of the solve phase in a task-based Cholesky solver using a sequential task flow model." NLAFET Working Note (2018).
[4] Dorris, Joseph, et al. "Task-based Cholesky decomposition on knights corner using OpenMP." International Conference on High Performance Computing. Springer, Cham, 2016.

This work is partially supported by MEXT Next Generation High-Performance Computing Infrastructures and Applications R&D Program "R&D of A Quantum-Annealing-Assisted Next Generation HPC Infrastructure and its Applications," and Grant-in-Aid for Scientific Research(B) #17H01706.