Asynchronous Task-Based Parallelization of Iterative Algebraic Solvers

Amani AlOnazi David Keyes

Extreme Computing Research Center, KAUST

IXPUG
INTEL EXTREME PERFORMANCE USERS GROUP
CONFERENCE

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

amani.alonazi@kaust.edu.sa

April 23, 2018

Overview

- Reducing synchronization and increasing concurrency are vital adaptations to hybrid architectures.
- We explore synchronization-reducing versions of classical solvers using a hybrid MPI+OmpSs/OpenMP.
- Main references:
  - Amani AlOnazi, George S. Markomanolis, and David Keyes. 2017. Asynchronous Task-Based Parallelization of Algebraic Multigrid. In ACM Proceedings of the Platform for Advanced Scientific Computing Conference (PASC ’17). DOI: https://doi.org/10.1145/3093172.3093230.
  - Amani Alonazi, Marcin Rogowski, Ahmed Al-Zawawi, and David Keyes. Performance Assessment of Hybrid Parallelism for Large-Scale Reservoir Simulation on Multi- and Many-core Architectures (submitted).
  - Amani AlOnazi, Lulu Liu, and David Keyes. Adoption of Less Synchronous Modes of Computation in Nonlinear Solvers (in progress).

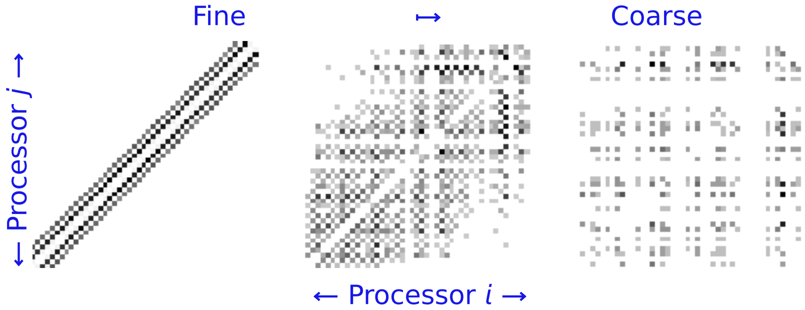
Hybridization and Taskification

- OpenMP is an API for shared memory parallel programming; "usually" uses a fork/join model.
- OmpSs uses data flow model to express the concurrency of the program to guarantee data race-free execution through synchronization mechanisms, i.e., dependences, taskwaits, atomics, ... etc.
- MPI has been used, since its appearance in the 90s, as one of the most favored parallel models for distributed memory environment.
- Tasks are the smallest unit of work that represent a specific instance of an executable kernel and its associated data.
- Dependences let the user express the data flow of the program, so that at runtime this information can be used to determine if the parallel execution of two tasks is concurrent or not.

Asynchronous Task-Based Iterative Algebraic Solvers

Motivations

- AMG is well-suited for large-scale scientific computing problems; operations scale as $O(N)$.
- Large parallel communication costs often limit parallel scalability:
  - Computation to communication ratio on coarser grids/levels.
  - Lack of scalability on emerging architectures.



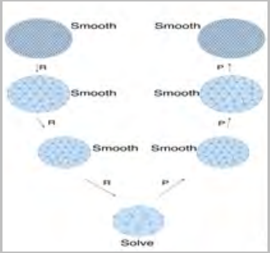
AMG

- AMG is an iterative method for solving $Ax = b$, very often used as a preconditioner.
- High-frequency error is reduced by relaxation, while low-frequency error is mapped to coarse-grids and reduced there.
- Consists of two phases: setup and solve.

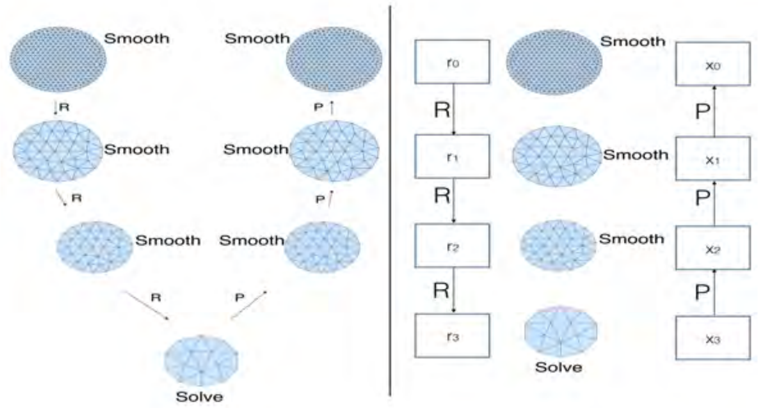
Setup Phase

- Select coarse grids/levels
- Define interpolation ${P_l}_{l=0}^{C-1}$
- Define restriction $R_l = {P_l}^T$
- Define coarse-grid $A_{l+1} = R_l A_l P_l$

Solve Phase



Additive AMG and Multiplicative AMG (1)



Additive AMG and Multiplicative AMG (2)

Algorithm 1 Multiplicative AMG V-cycle

Input: $\{x_l = 0\}_{l=0}^{C}, A_{l\,l=0}^{C}, R_{l\,l=0}^{C-1}, P_{l\,l=0}^{C-1}, S_{l\,l=0}^{C-1}$ in

$r_0 = b$
for $l = 0$ to $C-1$ **do**
    $x_l = S_l^{-1} r_l$
    $r_{l+1} = R_l(r_l - A_l x_l)$
end for
Solve $A_C x_C = r_C$
for $l = C-1$ to 0 **do**
    $x_l = x_l + P_l x_{l+1}$
    $x_l = x_l + S_l^{-T}(r_l - A_l x_l)$
end for

Algorithm 2 Additive VYAMG V-cycle

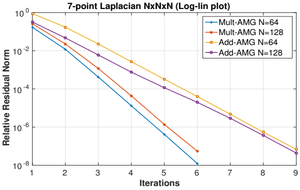
Input: $\{x_l = 0\}_{l=0}^{C}, A_{l\,l=0}^{C}, R_{l\,l=0}^{C-1}, \bar{P}_{l\,l=0}^{C-1}, S_{l\,l=0}^{C-1}$ in

$r_0 = b$
for $l = 0$ to $C-1$ **do**
    $r_{l+1} = R_l r_l$
end for
for $l = 0$ to $C-1$ **do**
    $x_l = S_l^{-1} r_l$
    $x_l = x_l + S_l^{-T}(r_l - A_l x_l)$
end for
Solve $A_C x_C = r_C$
for $l = C-1$ to 0 **do**
    $x_l = x_l + \bar{P}_l x_{l+1}$
end for

$\bar{P}_l$

$\bar{P}_l$ is a modified version of P_l to which one smoothing operation at the finer level has been applied, possibly truncated to a specified sparsity threshold. [1]

- One less SpMV
- Concurrent smoothing



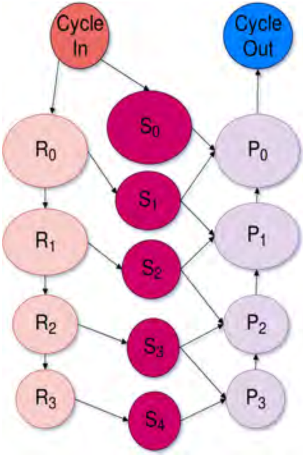
[1] Panayot Vassilevski and Ulrike Yang. Reducing communication in algebraic multigrid using additive variants. Numerical Lin. Alg. with Applic. (2014).

Task-based Parallelism

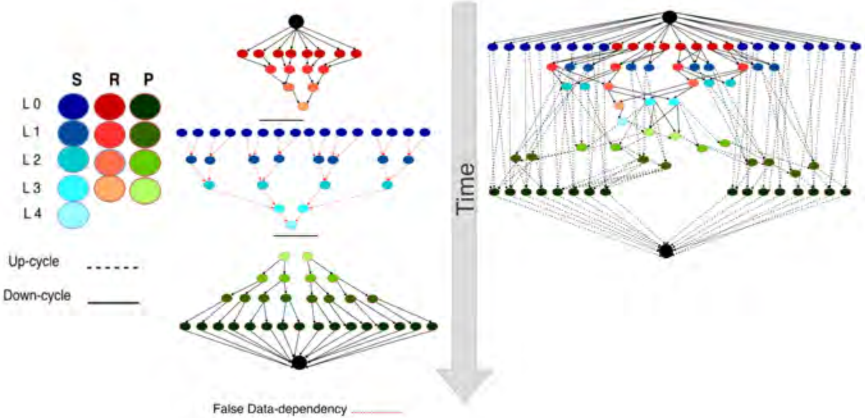
- Task-based parallel model expresses parallelism in terms of asynchronous tasks with data dependencies, i.e., input and output required for a unit of computation.
- Data dependencies between tasks are used to model the flow of data and by them schedule the execution dynamically.
- The runtime system schedules a pool of workers to conduct the execution of the DAG.
- Many task-based libraries, such as QUARK, StarPU, Legion, OmpSs ...

Taskification of The VYAMG Additive V-cycle

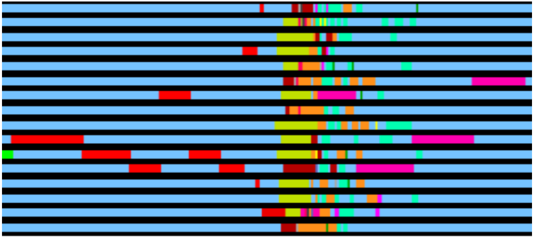
- Given the parallel loop of the smoothing levels in the grid hierarchy in the VYAMG.
- We can express the solve phase as a DAG, where vertices represent computational tasks and edges are the dependencies among them.
- This DAG will not lead to efficient performance due to the load imbalance between the tasks and the low level of concurrency.



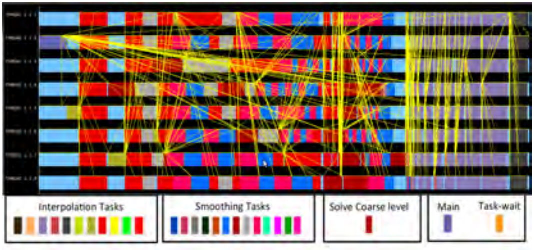
Tiling and Task Decomposition



Tiling and Task Decomposition Traces

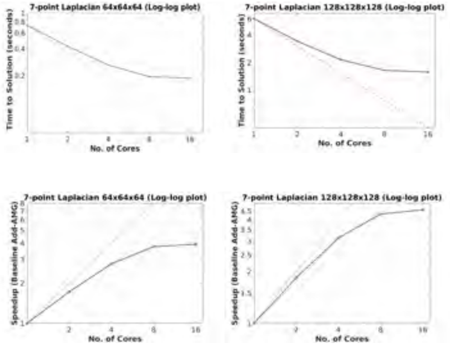


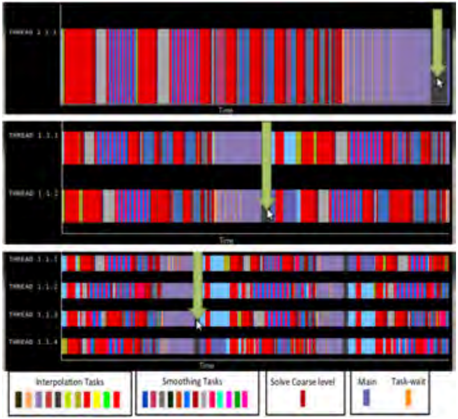
Time →



Intranode Concurrency Performance

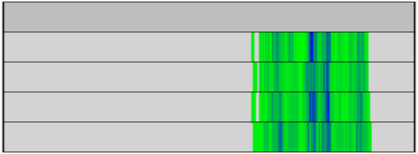
- Strong scalability data of only OmpSs VYAMG used as a preconditioner of CG solving the 3D Laplacian.





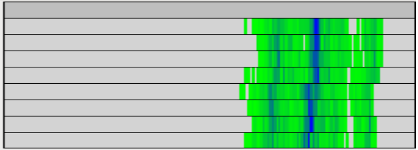
Intranode Concurrency Performance Analysis

0 → 22 L2 cache miss ratio

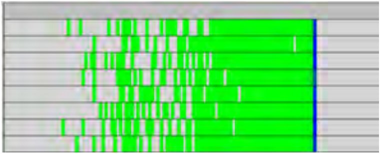


0 → 3.3 Ghz

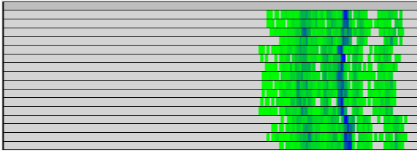
0 → 22 L2 cache miss ratio



0 → 3.3 Ghz



0 → 22 L2 cache miss ratio



0 → 3.3 Ghz

Hybrid MPI+OmpSs

- Hybrid distributed-shared memory parallelism.
- MPI exploits inter-node (or NUMA) parallelism while a task-based directed acyclic graph exploits node parallelism.
  - Asynchronous execution model
  - Hiding communication latency
  - Out-of-order execution of tasks
- Dynamic overlap of communications and computations through the inclusion of communication in the task graph (ongoing work).
- A separate task dependency graph for each MPI rank.
- `MPI_THREAD_FUNNELED` thread-safety level in combination with dynamic asynchronous task dependency graph for computations.
- Dynamically schedules the computations while the main thread is communicating, and with it latency hiding and asynchronous execution.

Performance Model

- Extended AMG performance model. [2]

$$T_R^l = \begin{cases} 2\frac{C_{l+1}}{P}\bar{s}_l t_l + \bar{\rho}_l \alpha + \bar{n}_l \beta, & l < M-1 \\ 2\frac{C_{l+1}}{P}\bar{s}_l t_l + \bar{\rho}_l \alpha + \bar{n}_l \beta + 2\frac{C_l}{P} + \rho_l \alpha + n_l \beta, & l_{add_p} = 0 \\ 0, & l = M-1 \end{cases}$$

$$T_P^l = \begin{cases} 2\frac{C_{l-1}}{P}\bar{s}_{l-1} t_l + \bar{\rho}_{l-1} \alpha + \bar{n}_{l-1} \beta, & l > 0 \\ 0, & l = 0 \end{cases}$$

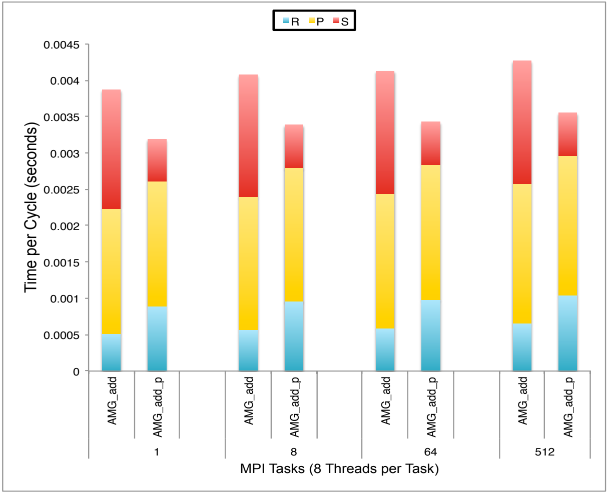
$$T_S^l = \begin{cases} 6\frac{C_l}{P} s_l t_l + 3(\rho_l \alpha + n_l \beta), & l < l_{add \vee add_p} \\ 4\frac{C_l}{P} s_l t_l + \rho_l \alpha + 2 n_l \beta + \max\limits_{0 \leq j \leq M} \rho_j \alpha, & l_{add} = 0 \\ 2\frac{C_l}{P} s_l t_l + n_l \beta + \max\limits_{0 \leq j \leq M} \rho_j \alpha, & l = l_{add} \vee l_{add_p} = 0 \\ 2\frac{C_l}{P} s_l t_l + n_l \beta, & l_{add \vee add_p} > 0 \end{cases}$$

$$T_{ADDAMG} = \Sigma_{l=0}^{M-1} T_R^l + \Sigma_{l=0}^{M} T_S^l + \Sigma_{l=0}^{M-1} T_P^l$$

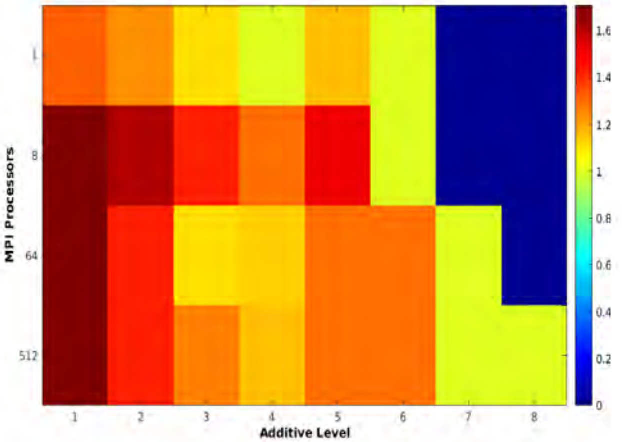
$$T_{ADD_pAMG} = \Sigma_{l=0}^{M-1} T_R^l + \max_{0 \leq l \leq M} T_S^l + \Sigma_{l=0}^{M-1} T_P^l$$

[2] Gahvari, Baker, Schulz, Yang, Jordan, and Gropp. Modeling the Performance of an Algebraic Multigrid Cycle on HPC Platforms. ACM ICS (2011)

Performance Model Results I



Performance Model Results II



- A model that helps finding the sweet spot from the improvement of load balance and concurrency as the size decreases and the overhead increase proportional to the number of tasks (on going work).

1.1 Algebraic Multigrid

1.2 Task-based Parallelism

1.3 Taskification of the VYAMG Additive V-cycle

1.4 Intranode Concurrency

1.5 Hybrid Distributed and Shared Memory Model

1.6 Performance Results

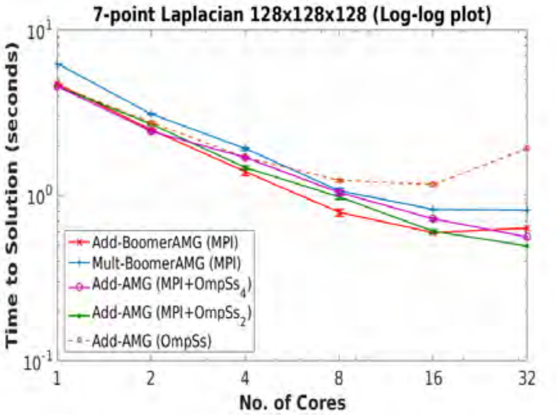
1.7 Summary - AMG

Experimental Setup

- We implement our approach within the BoomerAMG in the hypre library of LLNL.
- We use VYAMG as a preconditioner of conjugate gradient (CG) iterative solver.
- The combination of coarsening and interpolation is HMIS coarsening, and the ext+i interpolation truncated to at most 4 elements per row and one level of aggressive coarsening.
- All the test runs were performed on Shaheen Cray XC40.

Intranode Performance

- Strong scaling of BoomerAMG (MPI), Add-VYAMG (OmpSs), and Add-VYAMG (MPI+OmpSs), where the x-axis is MPI tasks× OmpSs workers.

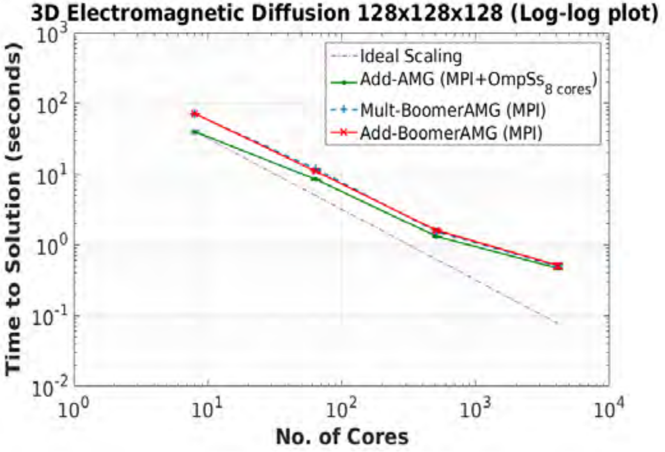


Internode Performance AMS

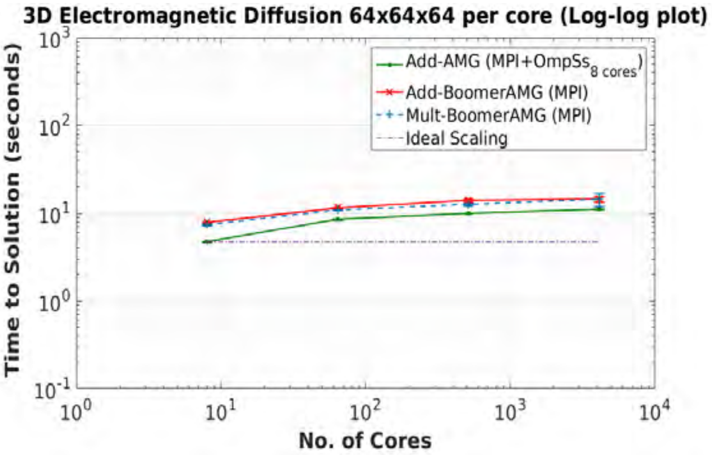
- Auxiliary-space Maxwell Solver (AMS) is a variational form of Hiptmair-Xu auxiliary space preconditioner for edge finite element discretization of the curl-curl formulation of the Maxwell equations.
- AMS employs unstructured mesh AMG preconditioners (i.e. BoomerAMG).
- We employ the DAG-based Add-VYAMG instead of BoomerAMG within the AMS preconditioner for a 3D electromagnetic diffusion on a smooth hexahedral meshing of the unit cube.

$$\nabla \times \nabla \times \vec{E} + \vec{E} = 1$$

Internode Performance: AMS Strongscaling



Internode Performance: AMS Weakscaling



Summary - AMG

- We presented first results for algebraic multigrid in a hybrid distributed-shared implementation using a directed acyclic graph tasked-based decomposition.
- We profiled the hypre BoomerAMG with MPI+OmpSs in the Intel multicore environment.
- The strong scaling results on 32 cores per single Haswell node of Cray XC40 demonstrate the feasibility of this approach to exploit dynamic scheduling of the domain-decomposed TDG, even for the scalar Laplacian operator, which has low arithmetic intensity.
- Faster strong and weak scaling results across nodes are also obtained for the 3D electromagnetic diffusion test case.

Asynchronous Task-Based Iterative Algebraic Solvers

1. Asynchronous Algebraic Multigrid
2. Linear Preconditioner in Reservoir Simulator
3. Additive Schwarz Preconditioned Inexact Newton
4. Summary

2.1 Reservoir Simulator

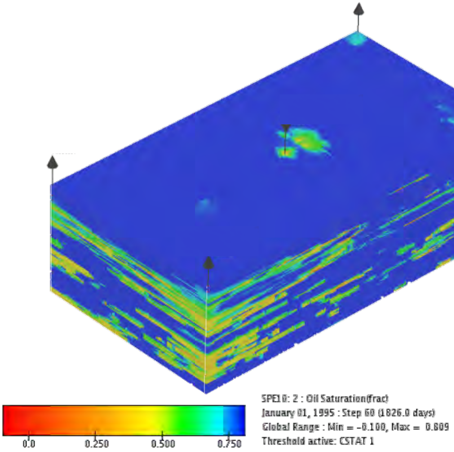
2.2 Independent Tasks for Hybridization

2.3 Performance of MPI+OpenMP using SPE10 Model

2.4 Summary - Reservoir Preconditioner

Reservoir Simulator Overview 1

- SPE10 Model:



Reservoir Simulator Overview 2

Two-Phase Flow

$$-\nabla\cdot(\lambda_t K \nabla p_w) = q_t + \nabla\cdot(\lambda_n K \nabla p_c) - \nabla\cdot((\lambda_n \rho_n + \lambda_w \rho_w) K g) \quad (1)$$

$$\frac{\partial S_w \phi}{\partial t} + \nabla\cdot(\lambda_w K(\nabla p_w - \rho_w g) = q_w \quad (2)$$

The Jacobian matrix A is decomposed:

$$A = P + E \quad (3)$$

where P is block-tridiagonal with grid cells ordered first in the Z-direction. E contains the remaining non-zero data block.

Linear Solver System

An approximate inverse preconditioner for A is:

$$A^{-1} \approx M_N^{-1} = [I + \sum_{k=1}^{N} (-1)^k (P^{-1}E)^k] P^{-1} \qquad (4)$$

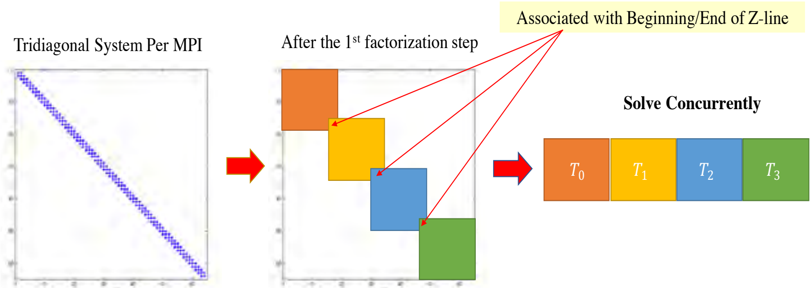
where N is the series term, and the action of P^{-1} is computed by a sparse direct solver, e.g., LU.

Constrained Pressure Residual uses Equation (4) as base-preconditioning.

Independent Tasks for Hybridization

- P is constructed by assembling the linear equations associated with the cells of maximum transmissibility traces in the horizontal XY plane plus the vertical-transmissibility terms of each Z line.
- A set of Z-line systems can be solved independently to construct the base preconditioner system.
- The main computational kernel is Z-line solve with upwards/dowards sweep coupled with matrix-vector multiplication (SpMV).

Independent Tasks for Hybridization

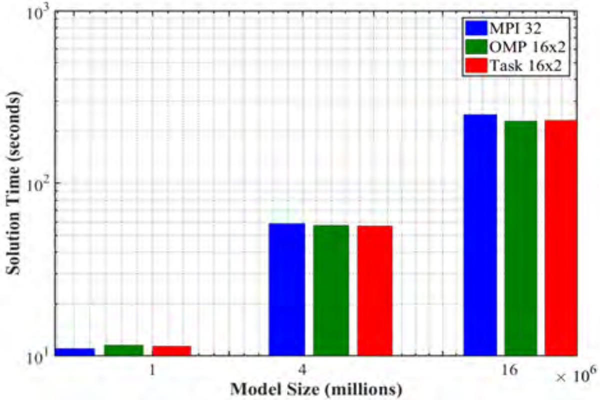


Linear Solver Implementation

- MPI: MPI everywhere implementation with no overdecomposition of the linear system.
- OMP: MPI+OpenMP implementation of the linear solver with the multithreaded overdecomposition of the system.
- Task: MPI+OpenMP with task-based implementation for the asynchronous progress of the P2P communication in the SpMV kernel.

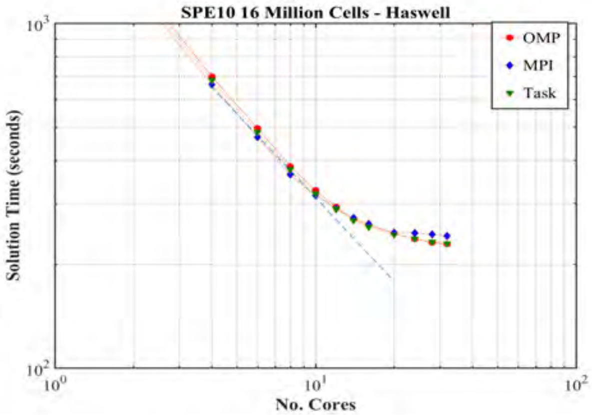
Haswell

- Intel Haswell (Xeon E5-2698v3), with 2 CPU sockets per node, and 16 processor cores per socket



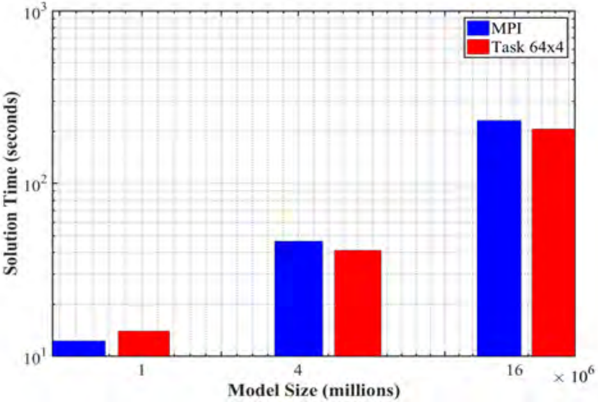
Haswell

- Intel Haswell (Xeon E5-2698v3), with 2 CPU sockets per node, and 16 processor cores per socket.



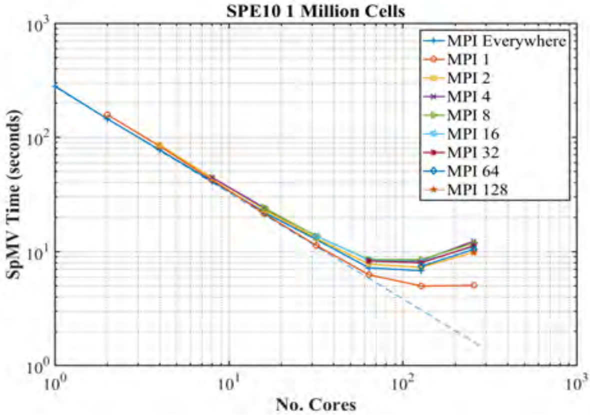
Knights Landing Node

- Intel Xeon Phi (Knights Landing 7210), which is equipped with 64 hardware cores.



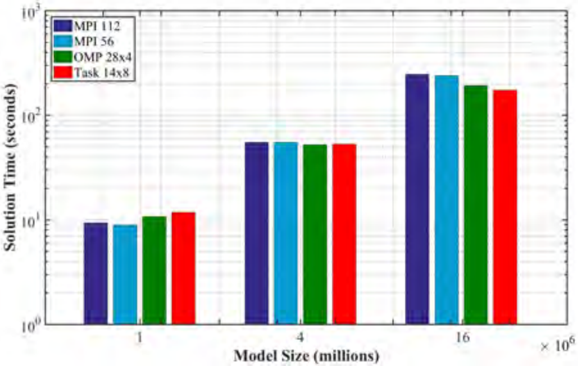
Knights Landing Node

- Intel Xeon Phi (Knights Landing 7210), which is equipped with 64 hardware cores.

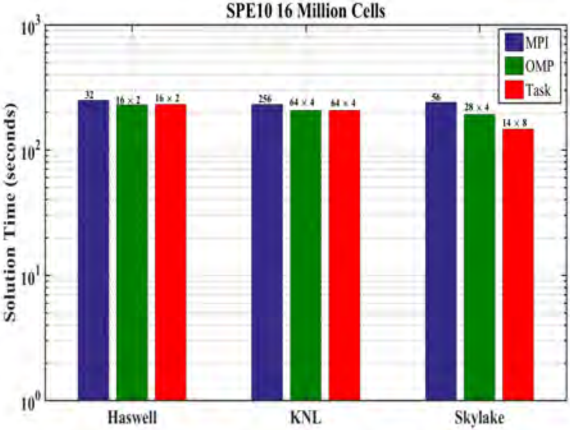


Skylake Node

- Intel Skylake (Xeon 8176), the successor of Haswell, with two sockets and 28 cores per socket.



Across architectures: Haswell, Skylake, and KNL



Summary - Reservoir Preconditioner

- We developed shared memory parallelization of the preconditioner and SpMV kernels as an execution time option to the distributed memory MPI model.
- Our results indicate that hybrid model can outperform MPI-everywhere per-node on Intel Haswell, Knights Landing, and Skylake.
- Benefits of a hybrid model are the most pronounced on Skylake, where the performance boost can reach 50% using 112 logical cores.

Asynchronous Task-Based Iterative Algebraic Solvers

1. Asynchronous Algebraic Multigrid
2. Linear Preconditioner in Reservoir Simulator
3. Additive Schwarz Preconditioned Inexact Newton
4. Summary

3.1 Nonlinear Preconditioning

3.2 Hybrid Model for Load Balancing

3.3 Preliminary Performance Result

3.4 Summary - ASPIN

Nonlinear Preconditioning

- Newton-like methods (e.g. Newton-Krylov-Schwarz) are often favored for the solution of nonlinear systems.
- Global Newton-like methods may waste considerable computational resources for problems that are nonlinearly stiff.
- Additive-Schwarz Preconditioned Inexact Newton (ASPIN) is a domain decomposition method for solving large, sparse nonlinear systems of equations. [3]
- It solves potentially unbalanced Newton problems on subdomains to derive a new nonlinear system with the same root as the original.

[3] Nonlinearly Preconditioned Inexact Newton Algorithms, X.-C. Cai and D. E. Keyes. SIAM J Sci. Comput. Vol. 24, pp. 183 - 200.

ASPIN Algorithm

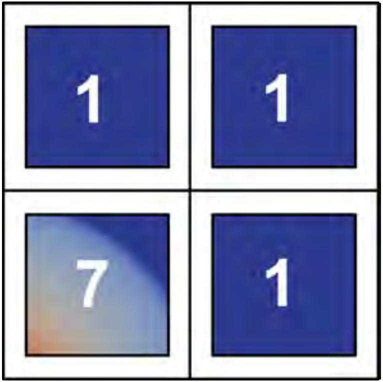
- Concurrent local solve, however, the computational cost for each subdomain may varies.
- For parallel implementation, as the number of subdomains increases, the overhead of load imbalance computation increases.

Algorithm 1 ASPIN

---

for $k = 0, 1, 2, \ldots$ until convergence **do**
    Compute $\mathcal{F}(x^k)$ as follows:
    Find $g_i^{(k)} = T_{\Omega_i}(u^{(k)})$ by solving local systems:
    $F_{\Omega_i}(x^{(K)} - g_i^{(K)}) = 0, i = 1, , N$
    Form the global residual $\mathcal{F}(x^k) = \sum_{I=1}^{N} g_i^{(K)}$
    Check the stopping condition on $\mathcal{F}(x^k)$
    Find $d^{(K)}$ by solving $\hat{\mathcal{J}} d^{(K)} = \mathcal{F}(x^k)$, where $\hat{\mathcal{J}} = \sum_{I=1}^{N} J_{\Omega_i}^{-1} J$
    Set $x^{(k+1)} = x^{(k)} - \lambda^{(K)} d^{(K)}$
end for

Two-phase Flow



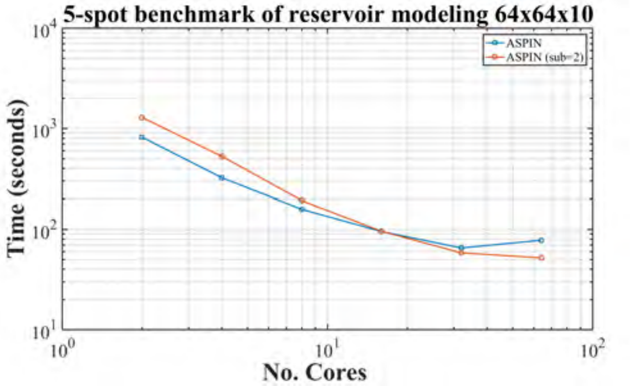
- Computational cost for each domain varies as the wetting phase front advances.

Hybrid Model for Load Balancing

- Dynamic load balancing between MPI processes can be hard and costly overhead.
- By design, domain decomposition implementation is often based on the assumption that executing the same computation on different data on every core will take the same time on every core.
- Hybrid MPI+OpenMP implementation may be able to mitigate these effects by using dynamic scheduling within each node and keeping number of MPI rank small.
- MPI provides communication primitives to exploit inter-node parallelism while OpenMP exploits parallelism within a node (always intra-node).
- Using this methodology facilitates the overlapping of communication and computation phases, improving the application performance.

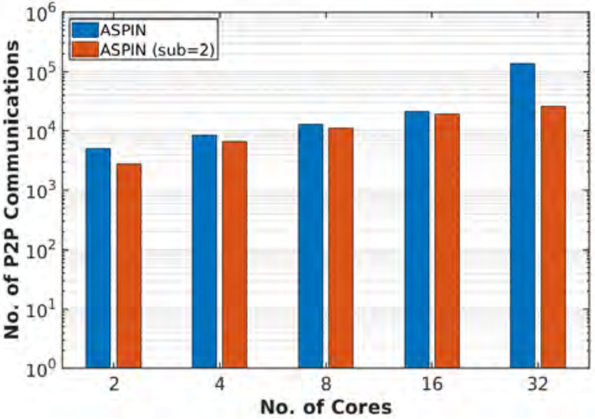
Performance Result I

- The quarter 5-spot benchmark of reservoir modeling with a 64x64x10 grid and homogeneous permeability.



Performance Result II

- The quarter 5-spot benchmark of reservoir modeling with a 64x64x10 grid and homogeneous permeability.



Summary - ASPIN

- We explored hybrid parallelization of ASPIN and load balancing using dynamic runtime system on Skylake node.
- ASPIN offers concurrent local solve, however, the computational cost for each local problem may vary.
- Number of ASPIN iterations is not sensitive to either the number of local subproblems or the number of unknowns.
- Computational cost is much higher in the subdomain near the wetting phase front advances than the other subdomains.
- Asynchronous implementation applied to a new modified ASPIN algorithm (joint work with Lulu Liu).

Asynchronous Task-Based Iterative Algebraic Solvers

1. Asynchronous Algebraic Multigrid
2. Linear Preconditioner in Reservoir Simulator
3. Additive Schwarz Preconditioned Inexact Newton
4. Summary

Summary

- Motivations for trying hybrid models:
  - Avoiding message passing within node/socket.
  - Avoiding memory copies required in MPI.
  - Load balancing without explicit data movement and complex implementation.
  - Overlapping computation and communication explicitly via asynchronous progress of communication.
  - Adapting to many-core emerging architecture.
- Tips for hybridization:
  - Take advantage of the first touch allocation policy (parallel initialisation).
  - Invoking MPI calls within asynchronous task is not trivial (MPI and OmpSs/OpenMP are totally decoupled).
  - If thread safety is implemented using global locks, the locking overheads and serialisation of MPI calls may outweigh any potential performance benefits.

Thank You!

Qs?

Email: amani.alonazi@kaust.edu.sa

https://github.com/ecrc

Special Thanks to my collaborators: ECRC and ARAMCO

Thanks to:

- ***hypre*** **group of Lawrence Livermore National Laboratory**
- **PETSc group of Argonne National Laboratory**
- **OmpSs and Tools (Paraver, Extrae) groups of Barcelona Supercomputing Center**