Fast Hybridized PDE Solvers

J. McLAUGHLIN, B. ERICKSON, J. CHOI

ABSTRACT– we present a parallel algorithm for the hybridized method for solving partial differential equations (PDEs) on multi-core CPUs, demonstrating hybridization as a powerful performance optimization strategy. Hybridization is a technique that is commonly used to reduce the degrees-of-freedom required to represent a numerical PDE, allowing larger problems to be solved using limited memory. Our algorithm leverages hybridization's ability to reduce the memory footprint of PDE solvers to improve their performance. Specifically, our algorithm improves data reuse through a symbolically blocked sparse data format, and improves cache utilization and load balancing through asynchronous task execution. We also develop a performance model that determines the configuration of a hybridize PDE solver that optimally utilizes the hardware resources of modern multi-core CPUs. We demonstrate that our optimally configured algorithm achieves up to 91× and 12× speedup over conjugate gradient (CG) and multi-frontal sparse QR methods, respectively. Our parallel algorithm enables hybridization as an effective performance optimization strategy for high-performance PDE solvers on modern multi-core CPUs.

Introduction.

Numerical PDEs often solve systems of equations that take the form

$$\mathbf{Au} = \mathbf{b}, \tag{1}$$

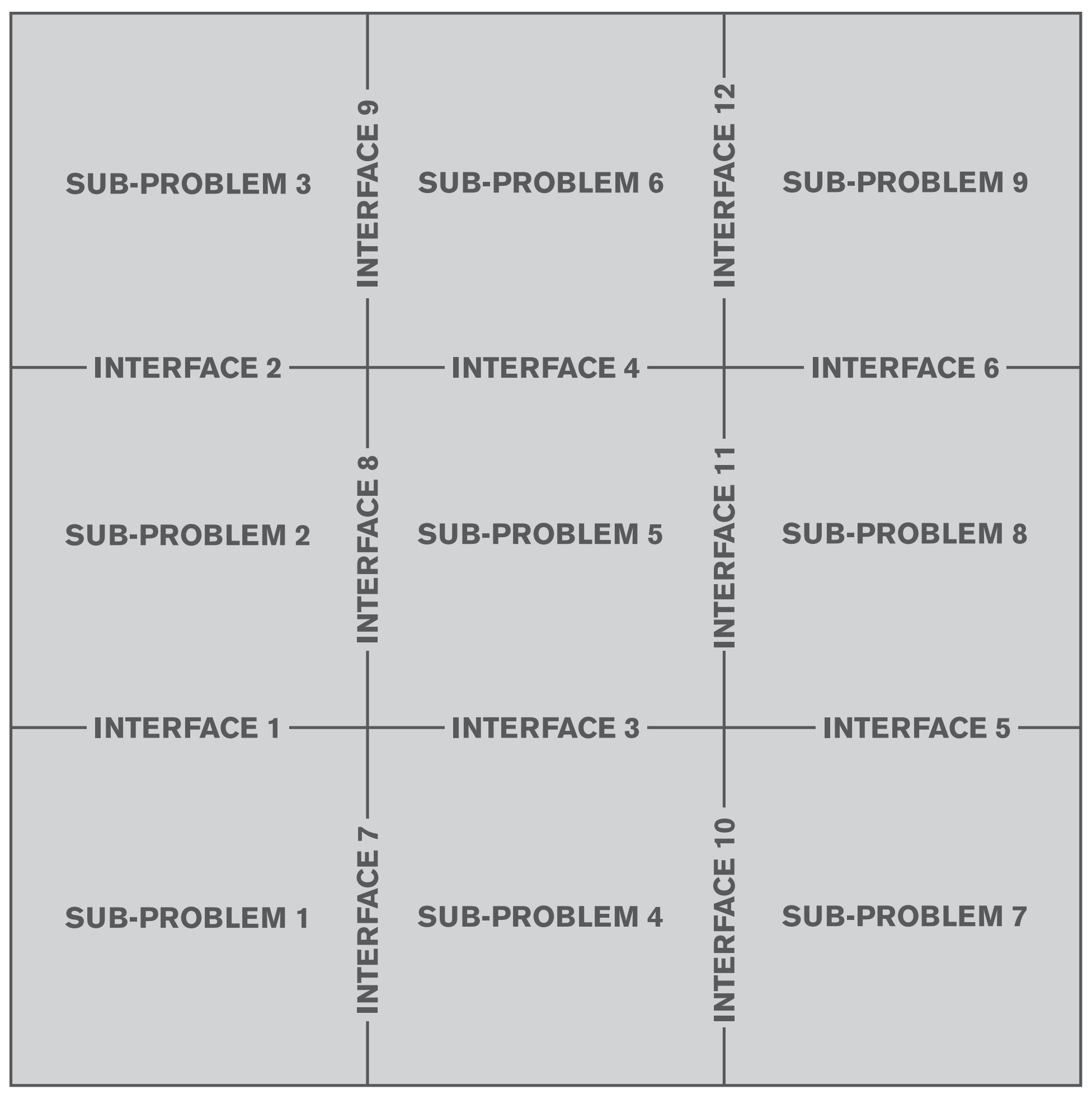
where the solution **u** is unknown, **A** is a matrix, often modelling the contraints of the problem, and the right-hand side **b** is a vector, typically comprised of input data and additional constraints [3,4]. Solving these equations is often computationally complex and such solver methods vary dramatically depending on the composition of **A** and **b**.

Hybridization [1] is a domain decomposition method for solving numerical steady state PDEs that rewrites Eq. 1 in the form

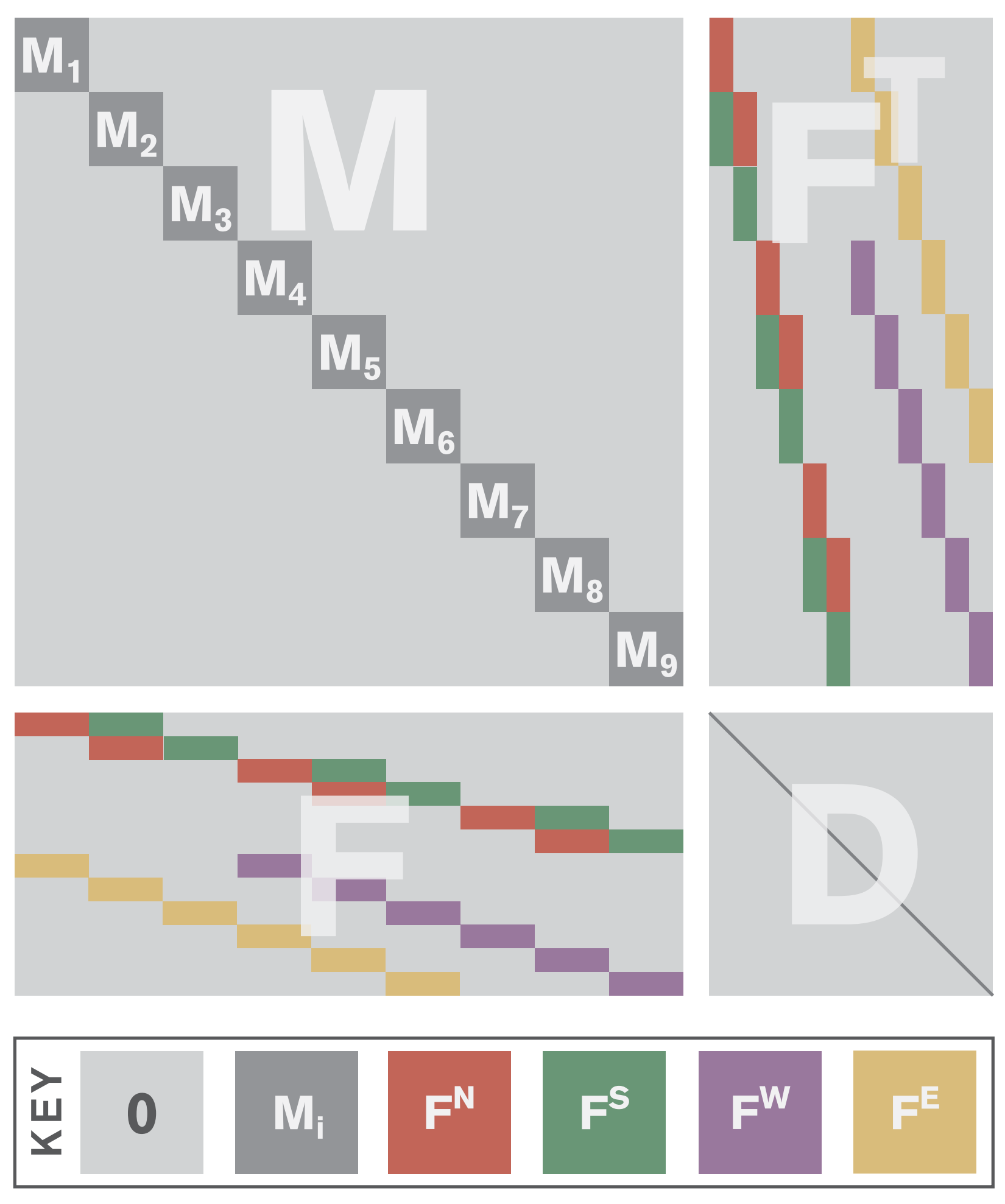
$$\begin{bmatrix} \mathbf{M} & \mathbf{F}^T \\ \mathbf{F} & \mathbf{D} \end{bmatrix}\begin{bmatrix} \mathbf{u} \\ \mathbf{u}_\delta \end{bmatrix}=\begin{bmatrix} \mathbf{b} \\ \mathbf{b}_\delta \end{bmatrix} \tag{2}$$

where M is a block diagonal version of **A**, **F** contains the off-diagonal entries of **A**, **D** is a diagonal matrix, and $\mathbf{u}_\delta$ and $\mathbf{b}_\delta$ are vectors relating to the interfaces between blocks. This form allows u to be computed using the Schur complement:

$$\mathbf{A}_\delta=\mathbf{D}-\mathbf{F}^T\mathbf{M}^{-1}\mathbf{F}\ (3) \qquad \mathbf{y}_\delta=\mathbf{b}_\delta-\mathbf{F}^T\mathbf{M}^{-1}\mathbf{b}\ (4) \qquad \mathbf{A}_\delta\mathbf{u}_\delta=\mathbf{y}_\delta\ (5) \qquad \mathbf{Mu}=\mathbf{b}-\mathbf{Fu}_\delta\ (6)$$



(a) 2D problem domain.



(b) **A** matrix pattern.

Figure 1*–The problem domain (a) and* ***A*** *matrix pattern (b) of a 2D, 9-block model hybridized problem. Note that each colored* $\mathbf{F}^j$ *matrix in (b) corresponds to an interface of a sub-problem in (a).*

Methods.

We design a new parallel algorithm for hybridized PDE solvers that reduces execution time by improving data reuse, load balancing, and synchronization. We achieve improved data reuse through the use of a symbolically blocked compressed sparse row (SBCSR) storage format and improved load balancing and synchronization by asynchronously computing low arithmetic intensity tasks between stages i and ii, and between stages iv and v, as shown in Fig. 3. Global barriers are still required before and after stage iii.

Symbolically blocked CSR

To represent a hybridized PDE solver using SBCSR storage, we decompose the factor matrix F from Eq. 2 into four unique sub-matrices (illustrated by the shaded blocked in Fig. 1b. Each sub-matrix corresponds to an internal interface of a sub-problem; as each sub-problem is identically sized there are only four unique internal interfaces/sub-matrices. The symbolic arrangement of interfaces in F is patterned and can be computed on-the-fly.

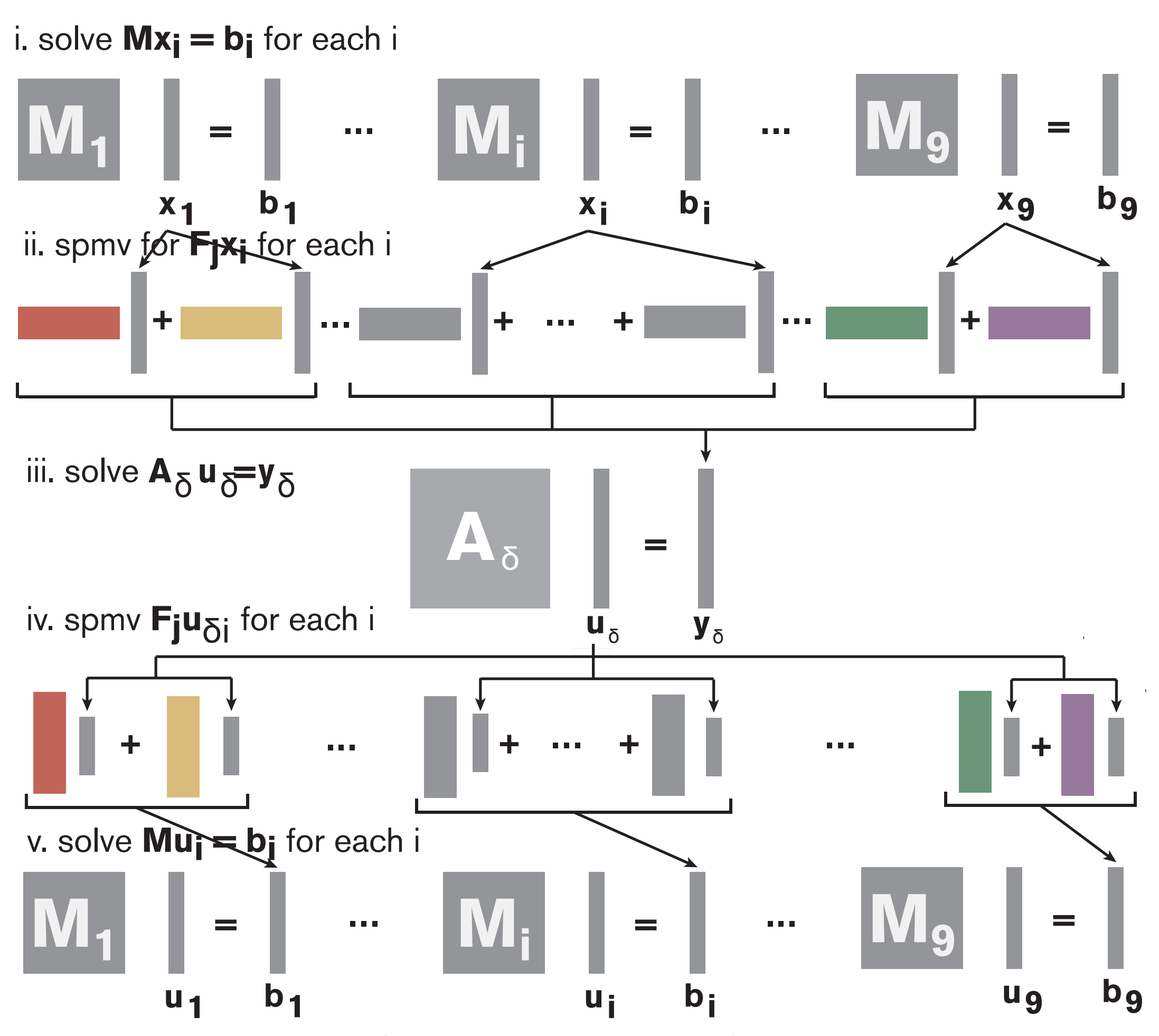

i. solve $\mathbf{Mx_i = b_i}$ for each i
M_1 x_1 b_1 M_i x_i b_i M_9 x_9 b_9
ii. spmv for $\mathbf{F_i x_i}$ for each i
iii. solve $\mathbf{A_\delta u_\delta = y_\delta}$
A_δ u_δ y_δ
iv. spmv $\mathbf{F_j u_{\delta i}}$ for each i
v. solve $\mathbf{Mu_i = b_i}$ for each i
M_1 u_1 b_1 M_i u_i b_i M_9 u_9 b_9

Figure 2*–Dataflow diagram of a 2D, 9-block model hybridized problem.*

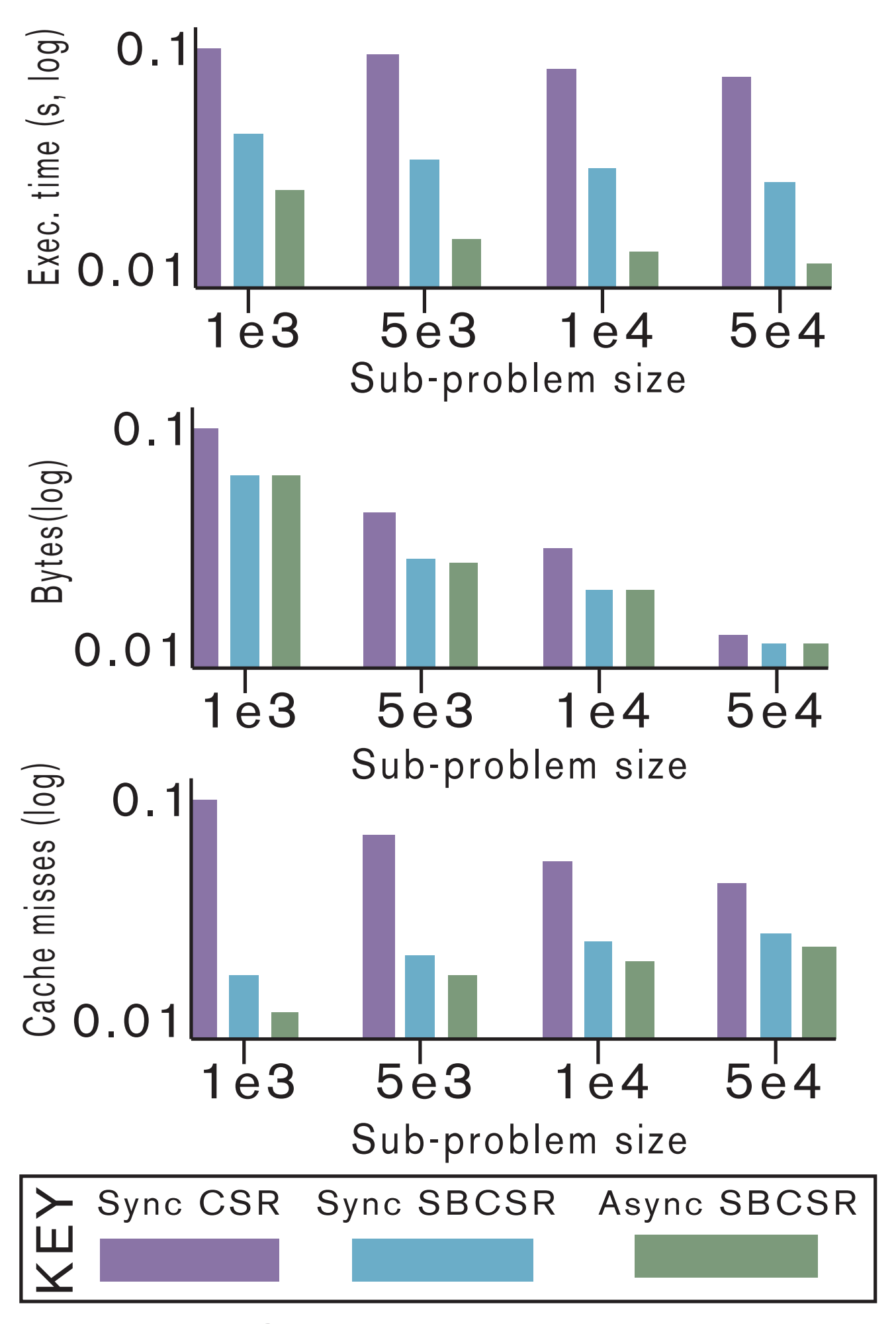


Figure 3*–Comparison of 3 variant parallel hybridized solvers, a naive, SBCSR, and asychronous SBCSR.*

Asynchronous parallelism. By analyzing the dependency between tasks in stages i and ii, and between stages iv and v, we can eliminate the barrier between each pair of stages; this reduces the synchronization overhead and exposes more parallel tasks for better load balancing. Additionally, some intermediate data (e.g., $\mathbf{x_1}$ from stage i in Fig. 2) is consumed by more tasks in the subsequent stage (e.g., stage ii). By prioritizing the execution of tasks whose intermediate data is consumed by more tasks in the subsequent stage, we improve its temporal locality to improve the overall cache utilization.

Performance model.
A key parameter that determines the performance of hybridized PDE solvers is the sub-problem density, i.e., the number of sub-problems. To maximize the performance of our parallel algorithm, we develop a performance model that selects the optimal sub-problem density for a given global volume size and processor. Our model derives from the Roofline model [2], which predicts performance based on the arithmetic intensity of the algorithm and the peak compute and memory performance of the processor.

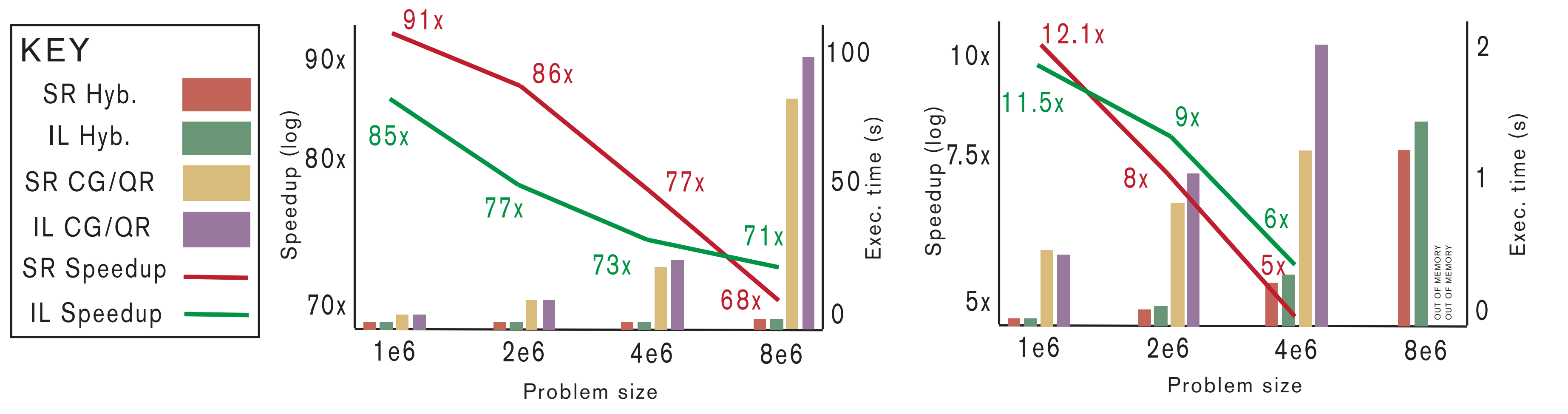


Figure 4*–Speedup achieved by our paralllel hybridized solver against non-hybridized sparse parallel solvers, i.e., conjugate gradient (CG) and multi-frontal sparse QR on Intel Sapphire Rapids (SR) and Ice Lake (IL) CPU platforms.*

Results.

We compare our hybridized PDE solver and sub-problem density performance model against both an iterative method, CG, and a direct method, multi-frontal QR. Fig. 4 shows the speedup achieved by our solver against both methods, for problem sizes ranging from 1e6 to 8e6, on our two test platforms. Against CG, our parallel hybridized PDE solver achieves speedup up to 91× on the Sapphire Rapids platform, and speedup up to 85× on the Ice Lake platform, demonstrating signficantly better performance. Against multi-frontal QR, our solver achieves a more modest speedup up to 12.1× on the Sapphire Rapids platform, and speedup up 11.5× on the Ice Lake platform. For large problems, direct methods can be impractical due to large memory usage. This is shown in Fig. 4, where we see that multi-front QR runs out of memory for the largest case.

Summary.

We develop a new parallel algorithm for a hybridized PDE solver that employs the SBCSR format and asynchronous maximize data reuse, and asynchronous task execution, prioritized by intermediate data reuse, eliminates redundant synchronization, and improves load balancing and data reuse. Together, the two optimizations deliver up to 9.1× speedup over a naively parallel implementation (Fig. 3). Further optimizations using our performance model deliver up to 91× speedup compared to other sparse solvers. This work is a key step towards developing high-performance implementations of hybridized PDE solvers. Future work will approach more complex problems, including problems that may have varied sub-problem sizes, complex geometrical domains, and variable coefficients.

References.

[1] Jeremy E Kozdon, Brittany A Erickson, and Lucas C Wilcox. 2021. 1066 Hybridized summation-by-parts finite difference methods. Journal 1067 of Scientific Computing 87, 3 (2021), 85.
[2] Samuel Williams, Andrew Waterman, and David Patterson. 2009. 1097 Roofline: an insightful visual performance model for multicore archi- 1098 tectures. 52, 4 (2009).
[3] H-O Kreiss and Godela Scherer. 1974. Finite element and finite differ- 1069 ence methods for hyperbolic partial differential equations. In Mathematical aspects of finite elements in partial differential equations. Else- 1070 vier, 195–212.
[4] Ken Mattsson and Jan Nordström. 2004. Summation by parts oper- 1078 ators for finite difference approximations of second derivatives. J. 1079 Comput. Phys. 199, 2 (2004), 503–540.

JOSEPH McLAUGHLIN
SPRING 2025