A Schur Complement Preconditioner for Scalable Parallel Fluid Simulation

Jieyu Chu
Shanghai Jiao Tong University, Oriental DreamWorks
and
Nafees Bin Zafar
Oriental DreamWorks
and
Xubo Yang
Shanghai Jiao Tong University

We present an algorithmically efficient and parallelized domain decomposition based approach to solving Poisson's equation on irregular domains. Our technique employs the Schur complement method, which permits a high degree of parallel efficiency on multi-core systems. We create a novel Schur complement preconditioner which achieves faster convergence, and requires less computation time and memory. This domain decomposition method allows us to apply different linear solvers for different regions of the flow. Subdomains with regular boundaries can be solved with an FFT based Fast Poisson Solver. We can solve systems with 1024^3 degrees of freedom, and demonstrate its use for the pressure projection step of incompressible liquid and gas simulations. The results demonstrate considerable speedup over preconditioned conjugate gradient methods commonly employed to solve such problems, including a multigrid preconditioned conjugate gradient method.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Parallel*

General Terms: Algorithms, Simulation

Additional Key Words and Phrases: parallel computing, Schur complement, Poisson solver, domain decomposition, fluid simulation.

1. INTRODUCTION

Fluid simulation is widely used in visual effects. Incompressible flows have proven to be very effective at generating many compelling effects in computer graphics including free surface flows, multi-phase and viscoelastic fluids, smoke, and fire element [Foster and Metaxas 1996; Stam 1999; Bridson 2015].

The projection method for incompressible fluid flow problems requires solving a Poisson problem to enforce the divergence free constraint. There are several numerical methods for solving this equation, like the Finite Difference Method (FDM) and the Finite Elements Method (FEM) [Press 2007]. The resulting linear system can be solved with an iterative algorithm like the preconditioned conjugate gradient (PCG) method [Golub and Van Loan 1996]. The number of iterations required for convergence in commonly used iterative linear solvers increases drastically with the size of the problem, making it impractical for simulations with very high resolutions [Hughes et al. 2007].

Two broad approaches have been developed in computational sciences to simulate at high resolutions. Solvers parallelized in a distributed computing model allow simulations to run faster and utilize larger memory resources. Adaptive solvers support different resolutions in different parts of the flow, thereby providing finer grained control over where the computation time is spent. Methods like adaptive mesh refinement (AMR) provide greater solution accuracy in "important" flow regions [Berger and Oliger 1984]. In the context of computer graphics visually interesting regions, such as areas of boundary interaction or regions close to the camera, are important. Losasso et al. [2004] developed a method using an octree grid representation for the flow domain. Several methods utilizing adaptive tetrahedral discretizations were developed by [Klingner et al. 2006; Chentanez et al. 2007; Batty et al. 2010]. Ando et al. [2013] introduce a hybrid method for highly adaptive liquid simulations using Eulerian tetrahedral meshes. They point out that the adaptive BCC mesh used is quite expensive to compute. They also note that artifacts may develop if the adaptivity is too high. English et al. [2013] propose a novel spatially adaptive method for simulating free surface incompressible flows using Chimera grid embedding. Ferst et al. [2014] present a method for liquid simulation on an adaptive octree grid using a hexahedral finite element discretization. Kim et al. [2009], Bojsen-Hansen et al. [2013], and Goldade et al. [2016] provide a clever compromise using low resolution computational solves with high resolution surface tracking. Chentanez et al. [2014] couple 3D Eulerian, height field, and particle methods for interactive simulations of certain types of flows. Two general realizations from surveying previous research are that the use of complicated dynamic data structures can have a substantial impact on solver performance and smaller timesteps needed due to higher

Emails: chujieyu@sjtu.edu.cn, nb@nafees.net, yangxubo@sjtu.edu.cn

DOI 10.1145/XXXXXXX.YYYYYYY
http://doi.acm.org/10.1145/XXXXXXX.YYYYYYY

resolutions combined with low-order numerical schemes deployed in graphics leads to excessive numerical dissipation.

Parallelizing numerical methods also pose significant challenges. ICPCG cannot be effectively parallelized because it involves sparse back-substitution. There are several options for solving the Poisson problem in parallel, such as the multigrid method. The multigrid method can be used as the preconditioner for the conjugate gradient method [Ashby et al. 1994; McAdams et al. 2010; Chentanez and Mueller-Fischer 2012; Tatebe 1993]. Implementing the multigrid method in a production environment requires some care in handling thin boundary features when transitioning from fine to coarse resolutions. These problems are recognized in literature as a weakness of the geometric multigrid method. They can be addressed by employing boundary aware interpolation and restriction operators as shown in Dick et al. [2016], or by using the more complex algebraic multigrid method [Falgout 2006]. The Fast Poisson Solver algorithm based on the fast Fourier transform (FFT) [Golub et al. 1998] offers an elegant high performance solution with an algorithmically lower complexity cost. However, this method requires a system with constant coefficients, thus it cannot support boundaries inside the flow domain. Henderson [2012] combines the iterated orthogonal projections (IOP) of Molemaker et al. [2008], with a Fast Poisson Solver to achieve fast, high-resolution gas simulations. Yang et al. [2016] modify the IOP framework by redistributing the divergence to improve convergence and reduce the iteration count. Unfortunately, IOP cannot ensure a strict divergence free condition, making it impractical for liquid simulations.

One of our research aims is to devise a parallel Poisson solver for liquid simulations which can also harness the benefits of an FFT based solver. We turn to the domain decomposition method [Mathew 2008; Cai 2003] as the framework for this solver. The domain decomposition method is ideal for exploiting parallelism and is also very efficient for irregular voxelized domains. The computational speedup is achieved while maintaining the accuracy of the result. Golas et al. [2012] use domain decomposition to couple vortex singularity methods and Eulerian velocity simulations.

We use the Schur complement decomposition method because the Schur reduced system is symmetric positive definite, and better conditioned than the original system [Gallier 2010]. Schur complement is based on a decomposition of the domain into non-overlapping subdomains. It obtains a reduced system of the Dirichlet boundary values for subdomains, and once the problem is solved the global solution can be obtained by solving a local boundary value problem on each subdomain. Since each subdomain is independent, the local solves can be executed in parallel. Henderson [1997] exploits the Schur complement method to eliminate rows and columns associated with element interiors. The Direct LU factorization method used by Henderson is impractical for high resolution 3d Poisson problems, because of its $O(n^3)$ complexity.

Liu et al. [2016] also present a Schur complement based fluid solver with good performance characteristics. Their work contributes an optimized implementation of an existing preconditioner on a hybrid CPU-GPU-many core architecture. They also do not exploit the independent nature of the subdomains, by using solvers that are algorithmically more efficient.

We create a novel preconditioner for the Schur complement system which is faster than previous work, making this method practical for large fluid simulations. The solver demonstrates high parallel efficiency on multi-core and multi-processor systems, and takes less time and memory than a comparable ICPCG solve. Different matrix solvers are utilized for different sub-problems in Schur complement system according to matrix properties. Regions inside the flow with simple boundaries utilize the Fast Poisson Solver, while subdomains with interacting solid boundaries can be represented with a high quality variational discretization, and solved with PCG [Batty et al. 2007]. Our method obtains the same accuracy as an ICPCG based solution of the original matrix.

We incorporate our solver into incompressible liquid, and gas flow solvers, and compare the simulation results with a standard solver. The Poisson problem is not the only performance bottleneck for high resolution liquid simulations. The FLIP method is the most common liquid simulation algorithm in animation and visual effects [Zhu and Bridson 2005]. This method requires large numbers of particles which consume a lot of memory, and the transfer of simulation quantities between particles and grids requires a lot of processing time. Thus we employ the recently introduced narrow band FLIP fluid simulation method (NB-FLIP) for our liquid simulations because it uses far fewer particles, while providing a similar visual result [Ferstl et al. 2016].

The contributions of this paper are:

—A novel Schur complement preconditioner which makes the Schur system converge faster, and requires significantly less time and memory than the existing Schur preconditioners.

—A framework to apply different solvers to inner subdomains to allow faster solution of large problems.

—The determination of the different choices of linear solvers for different decompositions of the Schur complement structure.

—The application of the Schur complement Poisson solver to solve the pressure projection in liquid simulations with complex boundaries, with higher parallel efficiency than commonly used methods.

2. SCHUR COMPLEMENT POISSON SOLVER

The Schur complement method parameterizes the global solution in terms of the Dirichlet values on the subdomain boundaries to obtain a reduced system. The preconditioned conjugate gradient method is employed to solve this system. Several alternative multi-subdomain preconditioners for a Schur complement system can be applied [Mathew 2008]. Our research focuses on the pressure projection step of an incompressible Navier-Stokes simulation which requires solving the Poisson equation in 3D. We choose the Finite Difference Method (FDM) because of its simplicity and proven reliability in graphics.

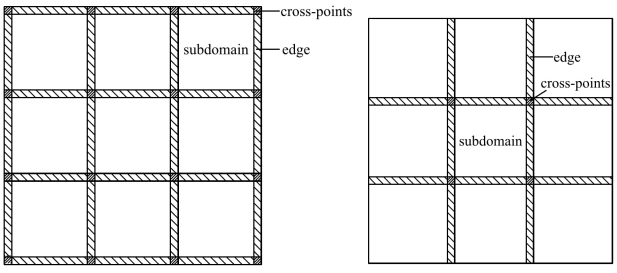

Fig. 1. Simple 2D decomposition.

The domain is decomposed into several subdomains as Figure 1. There will be $\Omega_1, \Omega_2, \cdots, \Omega_n$ subdomains. Ω_B is the set of the boundary points surrounding the subdomains. Using this decomposition to form the FDM matrix A, we can get the matrix equation in a block form as follows.

Table I. Symbols used in our paper

Symbol	Representation
Ω_i	Subdomain
Ω_B	Boundary set
W	Wirebasket set
ν	Cross-points set
F_l	Face set

$$\begin{pmatrix} A_{11} & & & A_{1B} \\ & A_{22} & & A_{2B} \\ & & \ddots & \vdots \\ & & A_{nn} & A_{nB} \\ A_{1B}^T & A_{2B}^T & \cdots & A_{nB}^T \quad A_{BB} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ x_B \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \\ b_B \end{pmatrix}. \quad (1)$$

The sub-matrices A_{ii}, for $i = 1 \cdots n$, are the subdomain matrices. A_{BB} is edge boundary matrix. The sub-matrices A_{iB} for $i = 1 \cdots n$, encode the interactions of the subdomains with the edge boundary set. The corresponding matrix visualization is shown in Figure 2. The solution of this system can be sought formally by block Gaussian elimination, as in Equation 2 and Equation 3.

$$\begin{cases} A_{11}x_1 + A_{1B}x_B = b_1, \\ A_{22}x_2 + A_{2B}x_B = b_2, \\ \cdots\cdots, \\ A_{nn}x_n + A_{nB}x_B = b_n. \end{cases} \quad (2)$$

$$A_{1B}^T x_1 + A_{2B}^T x_2 + ... + A_{nB}^T x_n + A_{BB}x_B = b_B. \quad (3)$$

We can rearrange Equation 2 to get x_i.

$$x_i = A_{ii}^{-1}(b_i - A_{iB}x_B). \quad (4)$$

Plugging x_i into Equation 3, we get the following equation for x_B.

$$Sx_B = b, \quad (5)$$

where S and b are

$$S = A_{BB} - \sum_{i=1}^{n} A_{iB}^T A_{ii}^{-1} A_{iB}. \quad (6)$$

$$b = b_B - \sum_{i=1}^{n} A_{iB}^T A_{ii}^{-1} b_i. \quad (7)$$

Given x_B, the inner subdomains can be solved independently in parallel according to Equation 4. Thus the main goal is to find a solution to the linear system in Equation 5. The S matrix must be explicitly assembled if a direct solver is employed to solve this system. Calculating the inverse of a matrix has $O(n^3)$ complexity, rendering this approach impractical for our needs. Another approach is to calculate $A_{ii}^{-1}A_{iB}$ in a column-by-column fashion. This is comparatively better, but this still acts as an algorithmic performance bottleneck.

It has been proven that S is symmetric positive definite and better conditioned than the original matrix A [Mathew 2008]. The PCG method can be employed to solve the Schur system. The PCG algorithm does not need the explicit assembly of the Schur complement matrix, instead only requiring the computation result of the multiplication of S with different vectors. For instance, Sw_B, given w_B, may be computed by first solving $A_{ii}w_i = A_{iB}w_B$ in parallel for w_i. Then Sw_B can be computed by the expression:

$$Sw_B = A_{BB}w_B - \sum_{i=1}^{n} A_{iB}^T w_i$$

The overall algorithm is shown in Algorithm 1.

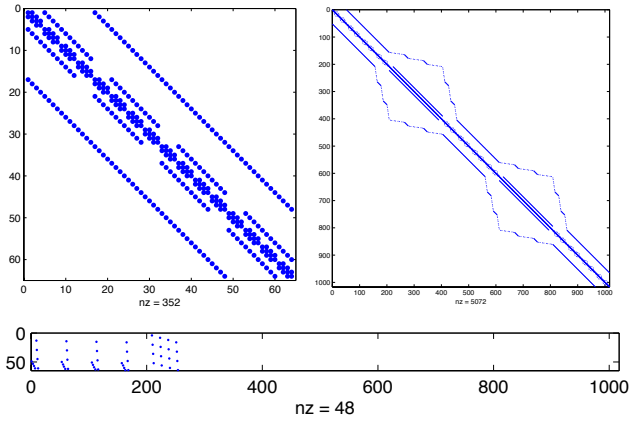

Fig. 2. Matrix visualization. The upper-left is subdomain matrix A_{ii}, the upper-right is edge boundary matrix A_{BB}, the lower one is transform matrix A_{iB}.

3. SCHUR COMPLEMENT PRECONDITIONER

3.1 Existing Schur Complement Preconditioner

A good preconditioner is required to accelerate the convergence of the PCG solver used to solve Equation 5. Several preconditioners have been developed for this system, such as Block-Jacobi [Bramble et al. 1986] with Dirichlet-Neumann [Bjørstad and Widlund 1986], Block-Jacobi with Neumann-Neumann [Bourgat et al. 1989], balancing domain [Mandel 1993], algebraic preconditioner with probing [Chan and Mathew 1992] and the Eigendecomposition preconditioner [Tong et al. 1991].

Figure 1 illustrates a 2D domain's decomposition. A 3D domain is divided into several parts: inner subdomains, face sets and wirebasket set. Figure 3 illustrates these components of a 3D decomposition, and Table I defines the corresponding symbols used in this paper. In the state of the art preconditioners we surveyed, Ω_B represents the set of wirebasket and face points, and all the existing preconditioners are based on this structure. We describe the Block Jacobi Preconditioner (BJP) [Mathew 2008] as an example. The BJP for S can be defined based on the partition of Ω_B into several face sets, $F_1, F_2, \cdots, F_q$, and the wirebasket set W. In matrix form, such a preconditioner will correspond to the block diagonal of the following matrix M:

$$M = \begin{pmatrix} S_{F_1F_1} & & & 0 \\ & \ddots & & \\ & & S_{F_qF_q} & \\ 0 & & & S_{WW} \end{pmatrix}. \quad (8)$$

Algorithm 1 Schur complement conjugate gradient method

function SCHURMULTIPLYVEC($schur, vectorB$)
  $result \leftarrow schur.A_{BB} * vectorB$
  for each $i \in [1, n]$ in parallel **do**

$$\begin{cases} b_i \leftarrow schur.A_{iB} * vectorB \\ solve : schur.A_{ii} x_i = b_i \\ u_i \leftarrow schur.A_{iB}^T * x_i \\ result \leftarrow result - u_i \end{cases}$$

  end for
  Output: result
end function

function SCHURPRECONDITIONEDCG($schur, x_0, b$)
  $k \leftarrow 0$
  $r0 \leftarrow b - SchurMultiplyVec(schur, x_0)$
  while $r_k \neq 0$ **do**
    $z_k \leftarrow SchurPreconditioner(schurprecond, r_k)$
    $k \leftarrow k + 1$
    if $k = 1$ **then**
      $p_1 \leftarrow r_0$
    else
      $\beta_k \leftarrow r_{k-1}^T z_{k-1} / r_{k-2}^T z_{k-2}$
      $p_k \leftarrow z_{k-1} + \beta_k p_{k-1}$
    end if
    $q_k \leftarrow SchurMultiplyVec(schur, p_k)$
    $\alpha_k \leftarrow r_{k-1}^T z_{k-1} / p_k^T q_k$
    $x_k \leftarrow x_{k-1} + \alpha_k p_k$
    $r_k \leftarrow r_{k-1} - \alpha_k q_k$
  end while
  $x \leftarrow x_k$
  Output: x
end function

The matrix M Equation 8 corresponds the Block Jacobi preconditioner and the action of the inverse of the preconditioner satisfies:

$$M^{-1} = \sum_{l=1}^{q} R_{F_l}^T S_{F_l F_l}^{-1} R_{F_l} + R_W^T S_{WW}^{-1} R_W. \quad (9)$$

R_{F_l} and R_W are defined by the interface restriction and extension matrices [Mathew 2008] between nodes on boundary set Ω_B and nodes on face set F_l or wirebasket set W. We have to get the approximation of the action of the inverses of the sub-matrices $S_{F_l F_l}$ and S_{WW}, since the Schur complement matrix S is not assembled. For the approximation of $S_{F_l F_l}^{-1}$, either the Dirichlet-Neumann or the Neumann-Neumann algorithm [Mathew 2008] can be used. Ω_{i_1} and Ω_{i_2} are the two inner subdomains which are connected with the face set F_l. The Dirichlet-Neumann mechanism uses Ω_{i_1} to get the preconditioner for F_l.

$$\begin{pmatrix} A_{i_1 i_1} & A_{i_1 l} \\ A_{i_1 l}^T & A_{ll} \end{pmatrix} \begin{pmatrix} v_{i_1} \\ v_l \end{pmatrix} = \begin{pmatrix} 0 \\ r_l \end{pmatrix} \quad (10)$$

The subdomain stiffness matrix in Equation 10 corresponds to the discretization of an elliptic equation on Ω_{i1} with Neumann boundary data on F_l [Mathew 2008]. The output v_l will be the preconditioned result for the elements of the face set F_l. Thus solving this system results in a single component of x_B in Equation 5. The Neumann-Neumann preconditioner uses both Ω_{i_1} and Ω_{i_2} to solve Equation 10. It receives two results v_l^1, v_l^2, and then averages them with a weight α which is commonly set to 0.5. The block sub-matrix S_{WW} equals A_{WW} when the original domain Ω is rectangular, and the subdomains are boxes. A seven point stencil is used for the finite difference discretization because of the property that for seven point stencils the nodal values on the wirebasket will not influence the interior Dirichlet solution in a box subdomain [Mathew 2008].

The Block Jacobi preconditioner is a very commonly used preconditioner [Mathew 2008], but it is still quite expensive to compute. For example, a Schur complement system has Q face sets $F_1, F_2, \cdots, F_q$, so the BJP will have to solve Q matrix equations for every precondition step. Each of the face solutions are entirely independent of the other solutions, thus there is no transfer of information between neighbors which impacts convergence.

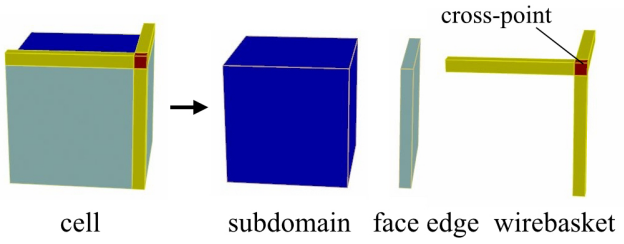

Fig. 3. Simple 3D decomposition.

3.2 Our New Schur Complement Preconditioner

We want to develop a preconditioner that converges faster, and requires less memory than the existing options. First we analyze the relationship amongst the different parts shown in Figure 3, including the subdomains Ω_i, the face sets F_l and the wirebasket set W. In the original Schur structure, the boundary set is the set of the face points and the wirebasket points. In order to get better performance, we want the size of the boundary set to be smaller. We recognize that the subdomains are only connected with the face sets, and disconnected with the wirebasket set because of the use of the 7-point finite difference Laplacian stencil. Therefore the wirebasket set W can be treated as another subdomain set, and the boundary set $\Omega_{B'}$ contains only the face points F_l. The matrix is structured as:

$$\begin{pmatrix} A_{11} & & & & A_{1B'} \\ & \ddots & & & \vdots \\ & & A_{nn} & & A_{nB'} \\ & & & A_{WW} & A_{WB'} \\ A_{1B'}^T & \cdots & A_{nB'}^T & A_{WB'}^T & A_{B'B'} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ x_W \\ x_{B'} \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \\ b_W \\ b_{B'} \end{pmatrix}. \quad (11)$$

The Schur complement matrix S' is shown as follows:

$$S' = A_{B'B'} - \sum_{i=1}^{n} A_{iB'}^T A_{ii}^{-1} A_{iB'} - A_{WB'}^T A_{WW}^{-1} A_{WB'}. \quad (12)$$

Applying this new structure with BJP will improve the performance, however the runtime, memory use, and convergence rates are still not ideal. We recognize that the wirebasket set W is connected only with face sets and can be used as the subdomain to approximate the whole boundary set, $\Omega_{B'}$, which includes only the

face sets. Neumann boundary conditions are used for other subdomains Ω_i, details in Section 3.1. Since only the boundary points are affected by the Neumann boundary condition, we only need to modify the $A_{B'B'}$ submatrix. This new preconditioner looks like:

$$\begin{pmatrix} A_{WW} & A_{WB'} \\ A_{WB'}^T & N_{B'B'} \end{pmatrix} \begin{pmatrix} v_W \\ v_{B'} \end{pmatrix} = \begin{pmatrix} 0 \\ r_{B'} \end{pmatrix} \tag{13}$$

The new preconditioner helps face sets, F_l, communicate with each other, thus achieving a high convergence rate. It requires less memory and computation time than the existing Schur preconditioners like BJP, because it only needs to save and solve one linear equation while BJP needs to save and solve Q linear equations (the number of subdomains which is >1).

The size of the matrix in Equation 13 will be large with many decomposition subdomains and degrees of freedom. Using standard PCG to solve Equation 13 would offer very poor parallel scalability. We recognize that the face sets F_l are all disconnected with each other, and another Schur system can be made for the matrix in Equation 13. The wirebasket set W will be the boundary set of the preconditioner's Schur system. The matrix in Equation 13 will be adjusted to the following form:

$$\begin{pmatrix} N_{F_1F_1} & & & & A_{F_1W} \\ & N_{F_2F_2} & & & A_{F_2W} \\ & & \ddots & & \vdots \\ & & & N_{F_qF_q} & A_{F_qW} \\ A_{F_1W}^T & A_{F_2W}^T & \cdots & A_{F_qW}^T & A_{WW} \end{pmatrix} \begin{pmatrix} x_{F_1} \\ x_{F_2} \\ \vdots \\ x_{F_q} \\ x_W \end{pmatrix} = \begin{pmatrix} b_{F_1} \\ b_{F_2} \\ \vdots \\ b_{F_q} \\ b_W \end{pmatrix} \tag{14}$$

We separate the cross-points set ν from the wirebasket set W. Thus Equation 14 becomes:

$$\begin{pmatrix} N_{F_1F_1} & & & & A_{F_1W'} \\ & \ddots & & & \vdots \\ & & N_{F_qF_q} & & A_{F_qW'} \\ & & & A_{\nu\nu} & A_{\nu W'} \\ A_{F_1W'}^T & \cdots & A_{F_qW'}^T & A_{\nu W'}^T & A_{W'W'} \end{pmatrix} \begin{pmatrix} x_{F_1} \\ \vdots \\ x_{F_q} \\ x_\nu \\ x_{W'} \end{pmatrix} = \begin{pmatrix} b_{F_1} \\ \vdots \\ b_{F_q} \\ b_\nu \\ b_{W'} \end{pmatrix} \tag{15}$$

We solve the Schur system and obtain the formula:

$$\widetilde{S} x_{W'} = r_{W'} \tag{16}$$

where the $\widetilde{S}$ is:

$$\widetilde{S} = A_{W'W'} - \sum_{l=1}^{q} A_{F_lW'}^T N_{F_lF_l}^{-1} A_{F_lW'} - A_{\nu W'}^T A_{\nu\nu}^{-1} A_{\nu W'} \tag{17}$$

We again use PCG to solve the Schur system in Equation 16. We can use the cross points system to obtain the preconditioned system for $\widetilde{S}$. The new wirebasket preconditioner becomes:

$$\begin{pmatrix} A_{\nu\nu} & A_{\nu W'} \\ A_{\nu W'}^T & N_{W'W'} \end{pmatrix} \begin{pmatrix} v_\nu \\ v_{W'} \end{pmatrix} = \begin{pmatrix} 0 \\ r_{W'} \end{pmatrix}. \tag{18}$$

This preconditioner corresponds to Algorithm 2, Line 17. The matrix in Equation 16 will only be of the size of the wirebasket set, and it will also have a simple banding structure. We can employ sparse Cholesky factorization to solve it.

Our preconditioner's overall algorithm is shown in Algorithm 2 which is very similar to Algorithm 1. The subdomain matrices and the edge boundary matrix of the preconditioner have a different banding structure than the generic solver. The subdomain matrices have a 5-point stencil, which is the 2D FDM discretization. We measured the convergence rates of different solvers for this type of problem and chose sparse Cholesky factorization to solve this system. This provides a good tradeoff between performance and resource utilization. Our convergence tests are discussed in further detail in section 4.

In order to make our preconditioner better for global information transfer, the surrounding points of the entire region are added into the edge set Ω_B and the wirebasket set W. For example, we will choose the left decomposition method in Figure 1 for 2D decomposition.

Table II shows the comparison between existing Schur preconditioners and our preconditioner. We use the energy equation $f = x^2 + y^2 + z^2$ as the test problem, and perform the tests on a machine with a single 6-core, 3.5GHz processor. A multipre-

Algorithm 2 Our new Schur complement preconditioner

1: **function** SCHURPCMULTIPLYVEC($schurpre, vectorB$)
2: $result \leftarrow schurpre.A_{W'W'} * vectorB$
3: **for** each $l \in [1, q+1]$ in parallel **do**
4: **if** $l = q+1$ **then**

$$\begin{cases} b_l \leftarrow schur.A_{\nu W'} * vectorB \\ solve: schur.A_{\nu\nu} x_l = b_l \\ u_l \leftarrow schur.A_{\nu W'}^T * x_l \end{cases}$$

5: **else**

$$\begin{cases} b_l \leftarrow schur.A_{F_lW'} * vectorB \\ solve: schur.A_{F_lF_l} x_l = b_l \\ u_l \leftarrow schur.A_{F_lW'}^T * x_l \end{cases}$$

6: **end if**
7: $result \leftarrow result - u_l$
8: **end for**
9: Output: result
10: **end function**
11:
12: **function** SCHURPRECONDITIONER($schurpre, R, MaxIter$)
13: $k \leftarrow 0$
14: $r0 \leftarrow R$
15: **while** $r_k \neq 0$ and $k < MaxIter$ **do**
16: * Equation 18 *
17: $z_k \leftarrow SchurPcPreconditioner(schurpcpre, r_k)$
18: $k \leftarrow k + 1$
19: **if** $k = 1$ **then**
20: $p_1 \leftarrow r_0$
21: **else**
22: $\beta_k \leftarrow r_{k-1}^T z_{k-1} / r_{k-2}^T z_{k-2}$
23: $p_k \leftarrow z_{k-1} + \beta_k p_{k-1}$
24: **end if**
25: $q_k \leftarrow SchurPcMultiplyVec(schurpre, p_k)$
26: $\alpha_k \leftarrow r_{k-1}^T z_{k-1} / p_k^T q_k$
27: $x_k \leftarrow x_{k-1} + \alpha_k p_k$
28: $r_k \leftarrow r_{k-1} - \alpha_k q_k$
29: **end while**
30: $Z \leftarrow x_k$
31: Output: Z
32: **end function**

conditioned conjugate gradient scheme, [Bridson and Greif 2006; Spillane 2016], may also be useful here to adapt the solver to different types of Poisson problems.

Table II. Comparison of Schur preconditioners(512^3)

	iterations	parallel Time
Block-Jacobi with Dirichlet-Neumann	78	4977.33s
Block-Jacobi with Neumann-Neumann	78	7882.56s
Balancing Domain Decomposition	97	4334.88s
Our Preconditioner	10	243.52s

4. SUBDOMAIN SOLVER CHOICE

One of the benefits of the Schur complement decomposition scheme is that it affords us the ability to choose different direct and iterative linear solvers depending on the structure of the particular subdomain. Depending on the structure of the particular subdomain one could use multiple methods such as PCG, FFT, multigrid, Cholesky factorization [Golub and Van Loan 1996], ILUT [Saad 1994] and so on. In our system we use a finite difference discretization and focus on three different solvers: PCG, sparse Cholesky factorization, and the FFT based Fast Poisson Solver algorithm.

The computational complexity of the CG algorithm for a 3D problem is $O(n^{4/3})$, and the complexity of the Fast Poisson Solver is $O(nlog(n))$ [Demmel 1997]. Though the FFT based fast Poisson solver method has low computational complexity, it can be used only when the original PDE has constant coefficients and boundaries that coincide with the coordinate lines. The sparse Cholesky factorization has high memory and computational costs in some cases. PCG can handle complex internal boundaries, which is one of the reasons for its popularity.

Table III shows the condition numbers of test matrices generated by 2D and 3D FDM discretizations. The test examples were obtained from Mitchell and McClain [2010]. In order to ensure the matrices have the same size, we compare the condition number under the condition that they have equal numbers of degrees of freedom. The results show that the matrix generated by a 2D FDM discretization is ill conditioned. The Cholesky factorization of such a matrix has an acceptable number of nonzero elements, and is not much more expensive than the PCG method.

In the Schur complement formulation this linear system, $A_{ii}x_i = b_i$, has to be solved many times with different b_i, but the same A_{ii}. Therefore computing and reusing the Cholesky factorization of this matrix is an efficient approach. We found that in general for a 2D system, sparse Cholesky factorization is a suitable alternative. However, using it for a 3D domain is very expensive, since the number of nonzero elements in the factored matrices is relatively high. Furthermore, if the matrix for a 3D problem is well conditioned, the conjugate gradient algorithm is efficient.

For the 3D case, we employ a combination of the linear equation solvers in our algorithm. The Schur complement system has multiple subdomains, some of which meet the requirements for using the FFT method. For such subdomains, fast FFT-based Poisson solver can be employed and we choose the PCG method for the rest. Any preconditioner for PCG method is available such as incomplete Cholesky preconditioner, modified incomplete Cholesky preconditioner or diagonal preconditioner as long as the memory is enough. If there is not enough memory for the incomplete Cholesky preconditioner, we can choose the diagonal preconditioner, or avoid using any preconditioner.

Table III. Condition number test

Degrees of freedom	2D		3D	
	N	Condition Number	N	Condition Number
64	8	196.829	4	94.9691
729	27	62741.6	9	1399.99
4096	64	79567.4	16	9696.85
15625	125	565988	25	43741.8

5. IMPLEMENTATION AND PARALLELIZATION

Multi-threading: Most operations can be run in parallel in our solver, and the most expensive step is designed to be parallel. The subdomain linear systems can be solved independently of other subdomains. Our implementation is designed for symmetric multiprocessing systems with multiple cores, and we utilize the Intel Thread Building Blocks library for parallelizing operations [Reinders 2007]. It has to be noted that we consider the step $result = result - u_i$ in the SCHURMULTIPLYVEC function in Algorithm 1 to be parallel. Given that any element in the $result$ vector may be updated by at most 3 threads, and the critical region is only a write operation, we could implement this with a spinlock however we have not yet done so. A distributed implementation would require synchronized communication between hosts. The boundary set, Ω_B, would need to be communicated to the individual hosts solving the subdomains. This will incur additional performance overhead, but we hypothesize that it will not change the basic performance characteristics of our algorithm. We aim to develop a distributed version of our solver in future work.

Fluid solver: We choose the narrow band FLIP method [Ferstl et al. 2016] for liquid simulations, and employ our Schur complement solver for the pressure projection step. Our method can be used in any fluid simulation algorithm which requires solution to a Poisson problem. To show the scalability of our algorithm, we also demonstrate two buoyant smoke simulations.

Domain partitioning: We use a simple regular decomposition method at present. Details of our decomposition strategy are shown in Figure 1 and Figure 3. We expect that there are far more optimal partitioning strategies, and plan to explore this in future work.

Solver Parameters: The settings for the convergence tolerance and maximum iteration limit for each solver has a significant impact. The tolerance of the subdomain solver should be set smaller than the tolerance for the Schur iterative method. We use a low number of iterations for the preconditioner solver, typically 5. This is a tradeoff between convergence accuracy and runtime performance.

6. EXAMPLES AND TEST RESULTS

6.1 Poisson Solver Test

The energy equation is used to test our Schur complement Poisson solver compared with the ICPCG solver, and the Poisson equation $\Delta f = 6$ is the input. In order to study the effect of the Schur complement decomposition, we compare our solver, using ICPCG for the inner domains, against a baseline ICPCG based Poisson solver. We test the execution time and accuracy on a 24 core, 2 processor system, with 128GB memory. The performance of the test problem using the ICPCG solver, and our Schur solver (including FFT-based inner solvers) is shown in Table IV. Our Schur complement solver demonstrates a significant performance advantage over the ICPCG solver when utilizing multiple cores.

Table V demonstrates the fast convergence of our method with the number of iterations and the RMS error of the results. Our new

preconditioner has better convergence rate with more domains, because with more domains the preconditioner is a better approximation of the original matrix. Thus the algorithm will converge faster with 16^3 subdomains than with 8^3 subdomains. However with more subdomains, the degrees of freedom in the preconditioner also increases, along with the complexity of the related data structures. Therefore the performance does not continue to improve in an unbounded fashion.

Table IV. Runtime comparison (sec) between standard ICPCG and Schur complement with different decomposition sizes

Resolution: 512^3	CPU			
Inner solver: ICPCG	1	8	16	24
ICPCG	2846.18	1997.04	1942.07	1934.64
8*8*8 subdomains	6166.51	954.67	548.42	449.71
16*16*16 subdomains	2607.32	417.15	259.83	230.91
8*8*8 subdomains(FFT)	456.75	80.40	55.82	52.43
16*16*16 subdomains(FFT)	463.29	87.92	65.15	58.35

Table V. The iteration number and residual's RMS error

Resolution: 512^3 **Inner solver: ICPCG**	Iterations	RMS
ICPCG	505	4.14E-06
8*8*8 subdomains	21	4.25E-06
16*16*16 subdomains	16	6.05E-06
8*8*8 subdomains(FFT)	21	6.65E-05
16*16*16 subdomains(FFT)	16	1.14E-04

Figure 4 shows the runtime of our Schur complement solver with different decomposition configurations and subdomain solvers for a 1024^3 resolution problem. It is worth noting that a standard ICPCG solver could not solve this problem, because it ran out of memory on our 128GB test system. Figure 5 shows the parallel speedup for this solve, as more cores are used. Another interesting observation is the relatively low parallel speedup of the FFT based solvers. The algorithmic speedup in these solvers, as seen in the runtime graph, makes the overhead in our system more apparent. Table VI shows the raw runtimes, and Table VII shows the iteration number of RMS error of each solver. The convergence is not affected by the high resolution.

The ICPCG solver was able to complete successfully for a 512^3 resolution problem, and the runtime is graphed in Figure 6. Figure 7 demonstrates the relative speedup of our solver over the standard ICPCG solver. Our method does have some setup overhead, which makes it slower in certain configurations when using a small number of cores.

Both problem sizes demonstrate the algorithmic acceleration afforded by the use of the FFT-based Poisson solver. However the use of this solver depends on the problem domain. If none of the decompositions meet the requirements of the FFT method, then all the subdomains will use ICPCG. Our data shows that the solver in this worst case scenario will still be able to utilize the parallelism due to the domain decomposition. For our test problem, utilizing 24 cores, using only ICPCG for the subdomains, our solver was a factor of 8 faster than a standard ICPCG solve. Our algorithm has some setup overhead, thus a pure FFT-based solve of an appropriate single large domain will be faster. However our implementation allows varying tile sizes, and a single large subdomain can almost entirely mitigate this overhead cost.

We observed that in practice our solver requires less memory. The FFT-based solver avoids the need to form explicit sparse matrices, thus decreasing memory use for simulations with many regular subdomains. Even if the PCG solver is used, the interior domains that have identical boundary conditions can share the same sparse matrix data structure. Finally, each subdomain is much smaller than the full system, and the temporary vectors required for PCG is bound by the number of subdomain computation threads.

We would like to point out that the ability to utilize different solvers for the subdomains opens up the possibility of trading increased computation time for reduced memory usage at a fine grained level. For example if the system does not have enough memory to utilize ICPCG for all subdomains, we can choose to deploy something very simple like a diagonal preconditioner in some subdomains. In our test examples with resolution 1024^3, the Incomplete Cholesky Preconditioned Conjugate Gradient (ICPCG) method and the Diagonal Preconditioned Conjugate Gradient (DPCG) method is combined to be the inner solvers, according to the available memory we have. The standard ICPCG needs at least 136GB memory, and is not available to solve this problem because of the 128GB memory limitation of our machine.

Table VI. Runtime comparison (sec) between different subdomain solvers in the Schur complement framework, with different decomposition sizes

Resolution: 1024^3	CPU			
Inner solver: ICPCG + DPCG	1	8	16	24
16*16*16 subdomains	25601.60	4218.24	2711.08	2248.59
32*32*32 subdomains	10874.5	1899.22	1232.69	1156.91
16*16*16 subdomains(FFT)	1811.63	317.73	204.21	177.45
32*32*32 subdomains(FFT)	1839.33	389.19	284.11	262.47

Table VII. The iteration number and residual's RMS error

Resolution: 1024^3 **Inner solver: ICPCG + DPCG**	iterations	RMS
16*16*16 subdomains	12	4.63E-05
32*32*32 subdomains	9	6.10E-05
16*16*16 subdomains(FFT)	12	1.34E-04
32*32*32 subdomains(FFT)	9	2.44E-04

We compare our work with McAdams' MG-PCG implementation in Table VIII. The two test problems had a sinusoidal right hand side, one with an immersed Dirichlet boundary and one with an immersed Neumann boundary. The results with these test problems show moderate performance gains for our method. We consider that multigrid method has some difficulties in the special treatments required to support thin boundaries.

Table VIII. Comparison of iterations and runtimes between MGPCG and Schur complement solver (16^3 subdomains).

	MGPCG iter	MGPCG time	Schur iter	Schur time
Scene1	23	43.35s	15	24.56s
Scene2	24	44.03s	14	22.99s

6.2 Fluid Simulation

We performed tests to show that our solver works well for high resolution incompressible flow simulations. We choose the FLIP algorithm [Zhu and Bridson 2005] for liquids, and the Semi-Lagrangian advection based scheme introduced by Stam for gaseous flows [Stam 1999]. Considering that we focus on fluid simulation with high resolution and using normal FLIP will produce too many particles, the narrow band FLIP fluid simulation solver [Ferstl et al. 2016] is employed. The following scenes are all simulated on a 2.5GHz dual processor, 24 core system with 128GB memory.

Table IX shows the averaged time spent for the pressure projection step of a FLIP liquid solver, comparing a standard PCG solver with a Modified Incomplete Cholesky preconditioner (MIC0) [Bridson and Greif 2006] for a dam break problem. The PCG implementation used contains some trivial parallelization for vector-vector and vector-scalar operations. Table X shows the runtimes corresponding to a smoke simulation. Given the high convergence rate of our preconditioner and the ability to apply different solvers in different subdomains, even our serial Schur solver is considerably faster than a standard MIC0-PCG solve. The resource requirements of our solver are also less.

We used our solver for liquid simulations with fluid-fluid and fluid-solid interactions (Figure 8, Figure 10, Figure 9). The surface extraction is performed using the OpenVDB library [Museth 2013], and no additional filtering is applied to the surface data. We also tested our solver for gas simulations with and without internal solid boundaries (Figure 12, Figure 13). In all cases our Schur complement solver does not exhibit any visual artifacts. The gas simulation examples have greater speedup over a standard MIC0-PCG based solve, because there are more regular subdomains where the Fast Poisson Solver can be utilized.

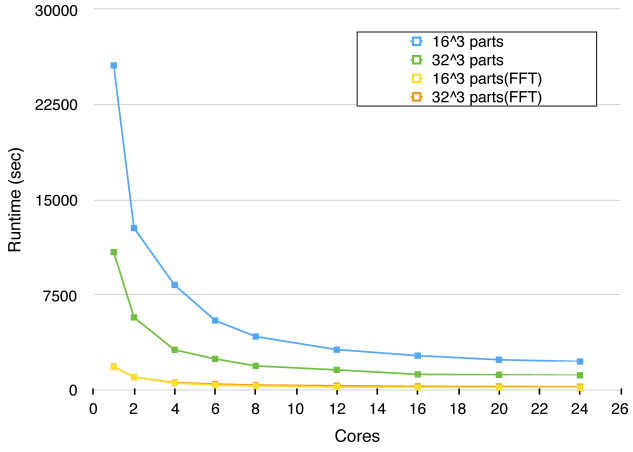

Fig. 4. Runtime for a 1024^3 problem. Standard ICPCG solver ran out of memory on our 64GB test system.

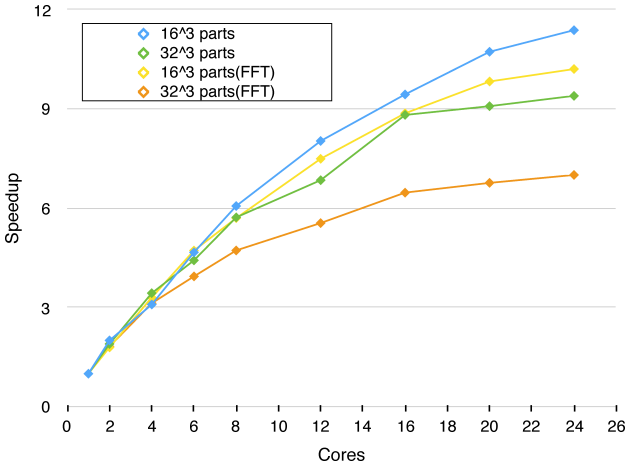

Fig. 5. Parallel speedup for 1024^3 problem.

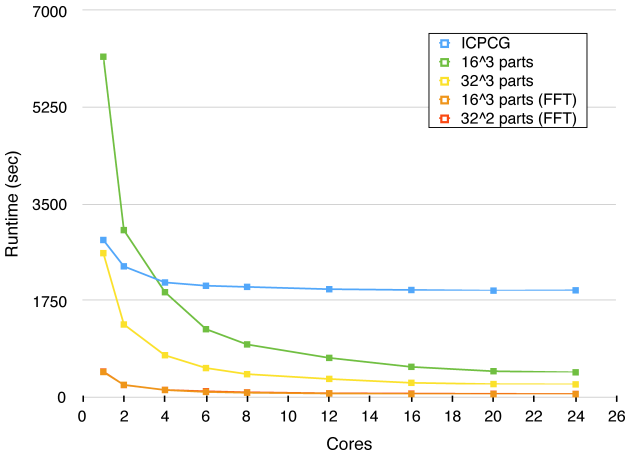

Fig. 6. Runtime for a 512^3 problem. Our algorithm has a performance overhead in certain subdomain configurations, at low CPU counts.

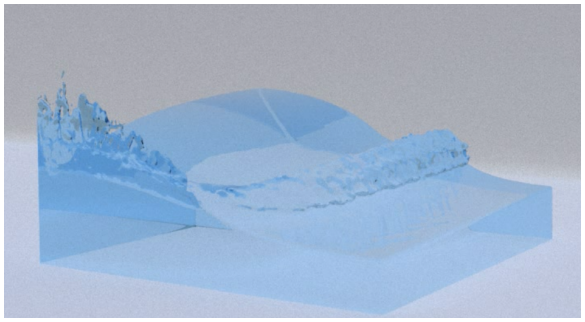

Fig. 8. Dam break (512^3).

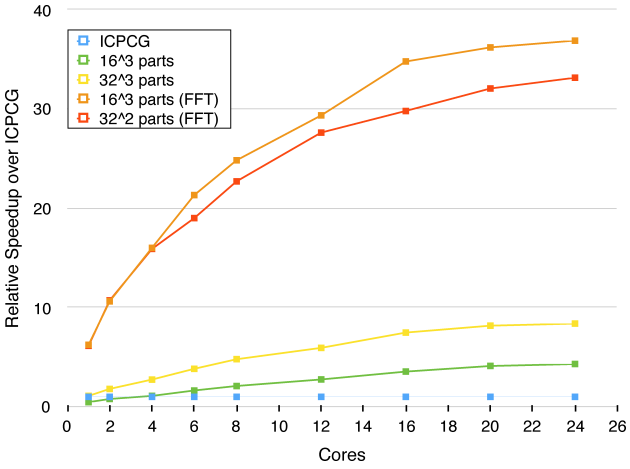

Fig. 7. Speedup compared to a standard ICPCG solve for a 512^3 problem.

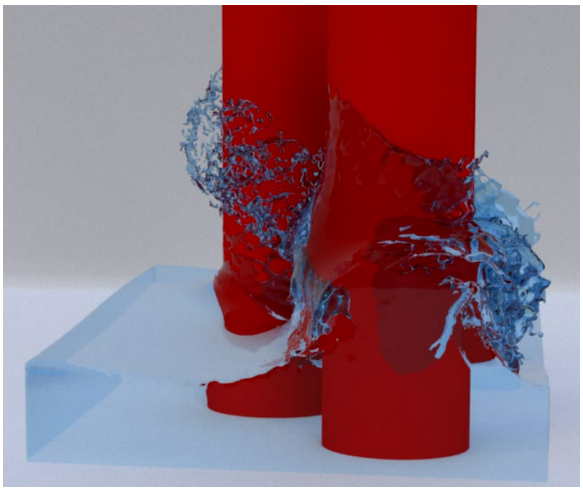

Fig. 9. Obstacles (512^3).

Fig. 10. Moving boundary interaction ($417 \times 65 \times 517$).

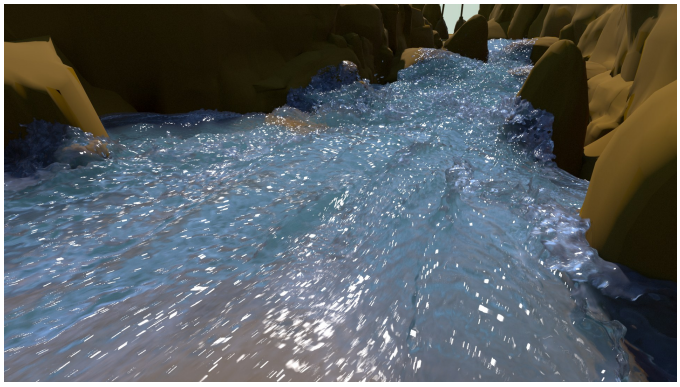

Fig. 11. Long river ($1000 \times 175 \times 1200$).

Table IX. Liquid Simulation Time. Resolution 512^3

Fluid Scene	MIC0-PCG ParalletT	Schur SerialT	Schur ParallelT	Speedup over MIC0-PCG
Dam break	261.49s	128.59s	14.96s	17.47
Double dam break	160.44s	126.85S	14.51s	11.05
Drop objects	163.63s	92.83s	11.47s	14.26
Obstacles	371.50s	187.66s	20.08s	18.50

Table X. Smoke Simulation Time. Resolution $512 \times 768 \times 512$

Smoke scene	MIC0-PCG ParallelT	Schur ParallelT	Speedup over MIC0-PCG
Plume	678.99s	47.36s	14.33
Plume with sphere	1108.16s	49.66s	22.31

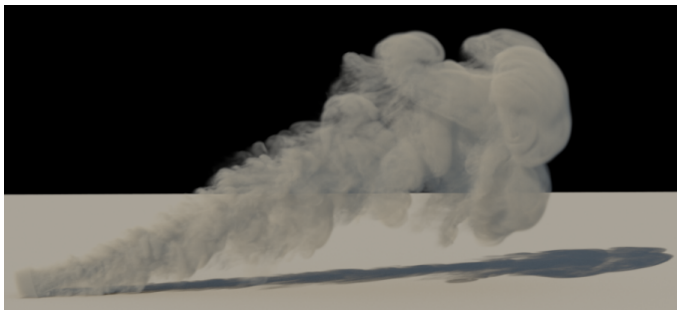

Fig. 12. Plume ($512 \times 768 \times 512$).

Fig. 13. Obstacles ($512 \times 768 \times 512$).

7. LIMITATIONS

The speedup achieved with our method is due to increased parallelism, the fast convergence achieved by our new preconditioner, and the ability to use a fast FFT-based Poisson solver in interior regions. However our preconditioner has a serial portion, and if the preconditioner system is very large the parallel efficiency will be negatively impacted. It is important to note that for a sufficiently large problem, the multigrid algorithm will overcome the performance advantage of our solver. Using PCG to solve the Schur complement boundary regions will also impact achievable parallel efficiency of a distributed implementation. The scalability is also less attractive for small resolutions, though the implementation allows varying tile sizes at runtime to mitigate this problem. The complexity of the algorithm makes a pure GPU implementation a significant challenge. We believe that a hybrid CPU-GPU implementation is a promising direction of future research. Because of our simple decomposition method, the degrees of freedom for the FFT solve may not be a power of 2. This is suboptimal for highly optimized FFT solver implementations [Henderson 2012]. A robust partitioner needs to consider optimal tile sizes, while avoiding creation of sliver domains which can impact the convergence rate of the system.

8. CONCLUSIONS AND FUTURE WORK

We create a novel preconditioner for the Schur complement method which makes it practical to use this technique for high resolution fluid simulation problems. Using a domain decomposition approach allows for greater parallelism. The use of different linear equation solvers in different flow regions accelerates the pressure projection step and reduces the memory consumption. Our solver is applicable in a shared memory multiprocessor environment and in distributed systems, with great potential for scalability. Our method can be used to solve any problem using a Poisson equation.

Our solver focuses on large scene simulation, and performs well for high resolution problems 1024^3 that are commonly encountered. Developing a robust domain partitioner will be key to achieving better performance. Our algorithm has high parallel efficiency and low global communication requirement. Therefore we believe that our solver will perform well in a distributed computing environment. Though there is greater implementation complexity, our approach minimizes the amount of data that needs to be transferred between hosts. In addition, we can utilize the memory of each machine in a distributed system, making simulations with even higher resolutions possible. The work of Liu et al. [2016] provides compelling details for creating a highly optimized implementation on a hybrid CPU-GPU architecture. We plan to incorporate their insights into our implementation.

ACKNOWLEDGMENTS

Xubo Yang and Jieyu Chu are partially supported by the National Natural Science Foundation of China (No.61373085, 61173105), and the National High Technology Research and Development Program of China (No. 2015AA016404). The authors would like to thank Peizhi Huang and Jiajun Lin for help with simulating and rendering. We are grateful to David Hill and Ron Henderson for explaining the Schur complement method, and Jeff Lait, Gene Lin, Jingxiang Li, Wang Gang, Chengdong Gui for useful discussions. This work would not have been possible without the support and encouragement of the R&D department and our colleagues at Oriental DreamWorks. Finally we would like to express our gratitude to the anonymous reviewers who patiently helped us write a better paper.

REFERENCES

ANDO, R., THÜREY, N., AND WOJTAN, C. 2013. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. Graph 32,* 4, 103.

ASHBY, S. F., FALGOUT, R. D., SMITH, S. G., AND FOGWELL, T. W. 1994. Multigrid preconditioned conjugate gradients for the numerical simulation of groundwater flow on the cray t3d. Tech. rep., Tech. report UCRL-JC-118622, Lawrence Livermore National Laboratory, Livermore, CA.

BATTY, C., BERTAILS, F., AND BRIDSON, R. 2007. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph 26,* 3, 100.

BATTY, C., XENOS, S., AND HOUSTON, B. 2010. Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. *Computer Graphics Forum 29,* 2, 695–704.

BERGER, M. AND OLIGER, J. 1984. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics 53*, 484–512.

BJØRSTAD, P. E. AND WIDLUND, O. B. 1986. Iterative methods for the solution of elliptic problems on regions partitioned into substructures. *SIAM Journal on Numerical Analysis 23,* 6, 1097–1120.

BOJSEN-HANSEN, M. AND WOJTAN, C. 2013. Liquid surface tracking with error compensation. *ACM Trans. Graph 32,* 4 (July), 68:1–68:13.

BOURGAT, J.-F., GLOWINSKI, R., LE TALLEC, P., AND VIDRASCU, M. 1989. Variational formulation and algorithm for trace operator in domain decomposition calculations. *Tony Chan, Roland Glowinski, Jacques P eriaux, and Olof Widlund, editors, Domain Decomposition Methods, Philadelphia, PA*.

BRAMBLE, J. H., PASCIAK, J. E., AND SCHATZ, A. H. 1986. The construction of preconditioners for elliptic problems by substructuring. i. *Mathematics of Computation 47,* 175, 103–134.

BRIDSON, R. 2015. *Fluid simulation for computer graphics*. CRC Press.

BRIDSON, R. AND GREIF, C. 2006. A multipreconditioned conjugate gradient algorithm. *SIAM Journal on Matrix Analysis and Applications 27,* 4, 1056–1068.

CAI, X. 2003. Overlapping domain decomposition methods. In *Advanced Topics in Computational Partial Differential Equations*. Springer, 57–95.

CHAN, T. F. AND MATHEW, T. P. 1992. The interface probing technique in domain decomposition. *SIAM Journal on Matrix Analysis and Applications 13,* 1, 212–238.

CHENTANEZ, N., FELDMAN, B. E., LABELLE, F., O'BRIEN, J. F., AND SHEWCHUK, J. R. 2007. Liquid simulation on lattice-based tetrahedral meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '07. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 219–228.

CHENTANEZ, N. AND MUELLER-FISCHER, M. 2012. A multigrid fluid pressure solver handling separating solid boundary conditions. *IEEE Trans. Vis. Comp. Graph 18,* 8, 1191–1201.

CHENTANEZ, N., MÜLLER, M., AND KIM, T.-Y. 2014. Coupling 3d eulerian, heightfield and particle methods for interactive simulation of large scale liquid phenomena. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '14. ACM, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 1–10.

DEMMEL, J. W. 1997. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

DICK, C., ROGOWSKY, M., AND WESTERMANN, R. 2016. Solving the fluid pressure Poisson equation using multigrid–evaluation and improvements. *IEEE Trans. Vis. Comp. Graph 22,* 11, 2480–2492.

ENGLISH, R. E., QIU, L., YU, Y., AND FEDKIW, R. 2013. Chimera grids for water simulation. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '13. ACM, ACM, New York, NY, USA, 85–94.

FALGOUT, R. D. 2006. An introduction to algebraic multigrid computing. *Computing in Science Engineering 8,* 6 (Nov), 24–33.

FERSTL, F., ANDO, R., WOJTAN, C., WESTERMANN, R., AND THUEREY, N. 2016. Narrow band FLIP for liquid simulations. *Computer Graphics Forum (Proc. Eurographics) 35,* 2, 225–232.

FERSTL, F., WESTERMANN, R., AND DICK, C. 2014. Large-scale liquid simulation on adaptive hexahedral grids. *IEEE Trans. Vis. Comp. Graph 20,* 10, 1405–1417.

FOSTER, N. AND METAXAS, D. 1996. Realistic animation of liquids. *Graphical Models and Image Processing 58,* 5, 471–483.

GALLIER, J. 2010. The Schur complement and symmetric positive semidefinite (and definite) matrices. `http://www.cis.upenn.edu/~jean/schur-comp.pdf`.

GOLAS, A., NARAIN, R., SEWALL, J., KRAJCEVSKI, P., DUBEY, P., AND LIN, M. 2012. Large-scale fluid simulation using velocity-vorticity domain decomposition. *ACM Trans. Graph 31,* 6, 148.

GOLDADE, R., BATTY, C., AND WOJTAN, C. 2016. A practical method for high-resolution embedded liquid surfaces. *Computer Graphics Forum (Proc. Eurographics 2016).*

GOLUB, G. H., HUANG, L. C., SIMON, H., AND TANG, W.-P. 1998. A fast Poisson solver for the finite difference solution of the incompressible navier–stokes equations. *SIAM Journal on Scientific Computing 19,* 5, 1606–1624.

GOLUB, G. H. AND VAN LOAN, C. F. 1996. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA.

HENDERSON, R. D. 1997. Nonlinear dynamics and pattern formation in turbulent wake transition. *Journal of Fluid Mechanics 352*, 65–112.

HENDERSON, R. D. 2012. Scalable fluid simulation in linear time on shared memory multiprocessors. In *Proceedings of the Digital Production Symposium*. ACM, 43–52.

HUGHES, C. J., GRZESZCZUK, R., SIFAKIS, E., KIM, D., KUMAR, S., SELLE, A. P., CHHUGANI, J., HOLLIMAN, M., AND CHEN, Y.-K. 2007. Physical simulation for animation and visual effects: Parallelization and characterization for chip multiprocessors. *SIGARCH Comput. Archit. News 35,* 2 (June), 220–231.

KIM, D., SONG, O.-Y., AND KO, H.-S. 2009. Stretching and wiggling liquids. *ACM Trans. Graph. 28,* 5 (Dec.), 120:1–120:7.

KLINGNER, B. M., FELDMAN, B. E., CHENTANEZ, N., AND O'BRIEN, J. F. 2006. Fluid animation with dynamic meshes. *ACM Trans. Graph. 25,* 3 (July), 820–825.

LIU, H., MITCHELL, N., AANJANEYA, M., AND SIFAKIS, E. 2016. A scalable Schur–complement fluids solver for heterogeneous compute platforms. *ACM Trans. Graph 35,* 6 (Nov.), 201:1–201:12.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Trans. Graph 23,* 3 (Aug.), 457–462.

MANDEL, J. 1993. Balancing domain decomposition. *Communications in numerical methods in engineering 9,* 3, 233–241.

MATHEW, T. 2008. *Domain decomposition methods for the numerical solution of partial differential equations*. Vol. 61. Springer Science & Business Media.

MCADAMS, A., SIFAKIS, E., AND TERAN, J. 2010. A parallel multigrid Poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, 65–74.

MITCHELL, W. F. 2010. A collection of 2d elliptic problems for testing adaptive algorithms. Tech. Rep. NISTIR 7668, NIST. February.

MOLEMAKER, J., COHEN, J. M., PATEL, S., AND NOH, J. 2008. Low viscosity flow simulations for animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, 9–18.

MUSETH, K. 2013. VDB: High-resolution sparse volumes with dynamic topology. *ACM Trans. Graph 32,* 3, 27.

PRESS, W. H. 2007. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

REINDERS, J. 2007. *Intel Threading Building Blocks*, First ed. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

SAAD, Y. 1994. ILUT: A dual threshold incomplete LU factorization. *Numerical linear algebra with applications 1,* 4, 387–402.

SPILLANE, N. 2016. An Adaptive Multipreconditioned Conjugate Gradient Algorithm. working paper or preprint.

STAM, J. 1999. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 121–128.

TATEBE, O. 1993. The multigrid preconditioned conjugate gradient method. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. NASA Conference Publication, 621–634.

TONG, C. H., CHAN, T. F., AND KUO, C. J. 1991. A domain decomposition preconditioner based on a change to a multilevel nodal basis. *SIAM journal on scientific and statistical computing 12,* 6, 1486–1495.

YANG, Y., YANG, X., AND YANG, S. 2016. A fast iterated orthogonal projection framework for smoke simulation. *IEEE Trans. Vis. Comp. Graph 22,* 5, 1492–1502.

ZHU, Y. AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Trans. Graph 24,* 3 (July), 965–972.