Parallel implementation in Chapel for the numerical solution of the 3D Poisson problem

Anna Caroline Felix Santos de Jesus
carolinefelix@usp.br
Instituto de Ciências Matemáticas e de Computação, University of São Paulo
São Carlos, Brazil

Willian Carlos Lesinhovski
wlesin@yahoo.com.br
Department of Environmental Engineering, Federal University of Paraná
Curitiba, Brazil

Livia S. Freire
liviafreire@usp.br
Instituto de Ciências Matemáticas e de Computação, University of São Paulo
São Carlos, Brazil

Nelson Luis Dias
nldias@ufpr.br
Department of Environmental Engineering, Federal University of Paraná
Curitiba, Brazil

ABSTRACT

In this study, we present a parallel implementation of the numerical Poisson equation with domain decomposition in three directions using the Chapel programming language. Our goal is to study the potential of Chapel as an easy-to-implement alternative to a code originally developed in Fortran+MPI. The numerical experiments were performed on the cluster of the *Instituto de Ciências Matemáticas e de Computação* of the University of São Paulo, on a grid 130^3 points, for a single node only. The performance of the Chapel version was 2-15% faster than the Fortran+MPI version, when up to 32 threads were used.

KEYWORDS

Chapel, Fortran+MPI, Poisson equation, domain decomposition

1 INTRODUCTION

The 3D Poisson equation has many relevant applications in science and engineering, including the modeling of heat transfer in solids with steady state properties and the simulation of incompressible flow problems. In the latter case, by taking the divergence of the momentum equation (Navier-Stokes Equation) and using the incompressibility constraint, one obtains a Poisson equation for the pressure field in the fluid. The general form of the Poisson equation is given by

$$\nabla^2 f = T, \tag{1}$$

with ∇^2 representing the Laplacian operator, f representing a scalar field and T its source term. In this study, we test the use of the Chapel programming language in solving the parallel 3D Poisson equation by comparing it with a Fortran+MPI code.

2 METHODS

In order to verify and test the efficiency of the numerical codes, we used the method of the manufactured solution (MMS) [3], by providing an analytical model $T = -3\sin(x)\cos(y)\sin(z)$ which results in the analytical solution for $f = \sin(x)\cos(y)\sin(z)$ on $\Omega = [0,1]^3$. The exact values of f are imposed on the cells adjacent to the domain boundary in all three directions.

The Poisson equation was discretized using the second-order centered Finite Difference Method (FDM) [2] resulting in a system of linear algebraic equations that were solved iteratively using the successive over-relaxation (SOR) method

$$f^{n+1} = (1-\omega)f^n + \omega f^*, \tag{2}$$

where f^* is the solution of the discretized equations and n is the iteration step. In this work, $\omega = 1.3$ was used.

Considering the tridimensional domain uniformly spaced, we have $\Omega_i = \Delta x \Delta y \Delta z = \frac{1}{imax} \times \frac{1}{jmax} \times \frac{1}{kmax}$, where Ω_i is a subdomain of the computational mesh of $\Omega = [0,1]^3$, and $imax \times jmax \times kmax$ are the number of grids.

The numerical implementation in Chapel is given by

```
config const imax, jmax, kmax : int;
const Lx = 1.0, Ly = 1.0,  Lz = 1.0 : real;
const dx = Lx/imax, dy = Ly/jmax, dz = Lz/kmax : real;
const w  = 1.30 : real;
const dx2 = dx*dx, dy2 = dy*dy, dz2 = dz*dz: real;
const beta1 = dx2/dy2 : real;
const beta2 = dx2/dz2 : real;
const w1 = (1.0 - w) : real;
const beta3 = 2.0*(beta1 + beta2 + 1.0) : real;
const tol = 1E-8: real;
var f, tf, residuo, errof : [1..imax, 1..jmax, 1..kmax] real;
```

Listing 1: Setup problem

The source term was implemented as

```
for i in 1..imax  do{
  x =  (i-0.5)*dx;
  for j in 1..jmax  do{
    y =  (j-0.5)*dy;
      for k in 1..kmax  do{
        z =  (k-0.5)*dz;
        tf(i,j,k) = tf_exact(x,y,z);
    }
  }
}
```

Listing 2: Implementation of the source term

with

```
proc tf_exact(x: real, y: real, z: real){
  var tff : real;
  tff =  -3.0*sin(x)*cos(y)*sin(z);
  return tff;
}
```

Listing 3: Source term

Imposing the known values of f on the cells adjacent to the boundary, for example, for the x-direction we have

```
// x - direction
for j in 1..jmax do{
  y = (j-0.5)*dy;
  for k in 1..kmax do{
    z = (k-0.5)*dz;

    x = 0.5*dx;
    f(1,j,k) = f_exact(x,y,z);

    x = (imax-0.5)*dx;
    f(imax,j,k) = f_exact(x,y,z);
  }
}
}
```

Listing 4: Imposing f in x-direction

where

```
proc tf_exact(x: real, y: real, z: real){
var tff : real;
  tff =  sin(x)*cos(y)*sin(z);
  return tff;
}
```

Listing 5: Analytical solution

For the other directions the implementation is analogous.

The methodology used consists of subdividing the computational domain into subdomains, where each subdomain is resolved by a process. For each process, the calculation for the iterative SOR scheme is performed, requiring communication with the surrounding processes. In MPI, this communication has to be implemented directly. In Chapel, to imitate the domain decomposition implemented in the Fortran+MPI code, we provide the following alternative using data parallelism on a single locale:

```
config const npx, npy, npz: int;
const ptsx = (imax-2)/npx, ptsy = (jmax-2)/npy,
      ptsz = (kmax-2)/npz : int;
```

Listing 6: Domain Decomposition - alternative Chapel

In this work, we paid particular attention to the fact that intent `ref` should be used in procedure returns, because this directly impacts the performance of the Chapel for procedures within loops, as shown in Listing 7.

```
proc sor_method(ref f: [] real, ref tf: [] real) ref {

coforall (ii,jj,kk) in {2..imax-1, 2..jmax-1, 2..kmax-1}
                                  by (ptsx, ptsy, ptsz){
  for i in ii..ii + ptsx-1 do{
    for j in jj..jj + ptsy-1 do{
      for k in  kk..kk + ptsz-1  do{
        f(i,j,k) =  w1*f(i,j,k) +
                    w * (-dx2*tf(i,j,k) + f(i-1,j,k) + f(i+1,j,k)
                    + beta1*( f(i,j-1,k) + f(i,j+1,k) )
                    + beta2*( f(i,j,k-1) + f(i,j,k+1) ))/ beta3 ;

      }
    }
  }
}
  return f;
}
```

Listing 7: Chapel coforall task-parallel construct

The range `{2..imax-1, 2..jmax-1, 2..kmax-1}` represents the computational domain, excluding edges, and `ptsx`, `ptsy` and `ptsz` represents the number of unknowns to be calculated in the x,y and z directions respectively. For example, if `ptsx=ptsy=ptsz =64` then 8 tasks will be created, since

```
{2..imax-1, 2..jmax-1, 2..kmax-1} by (ptsx, ptsy, ptsz)
// 64 x 64 x 64 :  decomposition into 2x2x2 blocks => 8 tasks
```

Listing 8: Domain decompostion using coforall loops

Thus, `npx=npy=npz=2` implies that `NP=8`.

It was observed that, because in Fortran the filling of the 3D matrix in memory is performed along the columns (column-major) whereas in Chapel the elements are aligned along the rows (row-major), the different combinations of domain subdivisions influence the performance comparison between the languages. Therefore, the best combinations of domain decomposition that favor each of the languages was chosen for each number of threads evaluated. Furthermore, to ensure better performance of the implemented versions, the chapel code was compiled with the `--fast` flag and the Fortran + MPI code with the `-O3` optimization flag. Table 1 presents the tools/ libraries and optimization flags used for compiling the programs.

Table 1: Summary of the tools/libraries and optimization flags used for compilation and execution

Tools/libraries	Version
C compiler gcc	12.2.1
Chapel	1.30.0
Chapel optimization flag	`--fast`
C compiler gcc	4.9.2
mpich	3.1.4
Fortran optimization flag	`-O3`

The numerical experiment was performed on the Euler Cluster of the *Instituto de Ciências Matemáticas e de Computação* of the University of São Paulo (whose configuration is shown in Tab. 2) with 28 cores with hyper-threading (for a total of 56 threads). For more details, see https://euler.cemeai.icmc.usp.br/documentacao/sistema. In particular, for `Chapel 1.30.0`, the environment variations used for single node are organized in Tab. 3.

Table 2: Machine and Operating System.

Machine	Cluster Euler
Memory	128 GB DDR3 1866MHz
Processador	Intel Xeon E5-2680v4 de 2.4 GHz
Operating System	CentOS Linux release 7.2.1511

Because this machine has 28 cores with hyper-threading, we measured the execution time with `NP = 2,4,8,16` and 32 threads by changing the setting of the `CHPL_RT_NUM_ THREADS_PER_LOCALE` environment variable. For the Fortran+MPI code, the number of processes was controlled using the following command line instruction,

```
mpiexec -np $NP ./exec
```

where `NP` is the desired number of processes.

Table 3: Summary of the environment configuration for single-locale execution and compilation.

Variable	Value
CHPL_RT_NUM_THREADS_PER_LOCALE	NP
CHPL_TASKS	*qthreads*
CHPL_TARGET_CPU	*native*
CHPL_HOST_PLATFORM	*linux64*
CHPL_LLVM	*none*
CHPL_COMM	*none*

3 RESULTS

We visualize the numerical solution as shown in Fig. 1 with the aid of Paraview software [1]. The best combinations of domain decomposition that favor each of the languages are shown in Tab. 4. Each code was run 3 times and the average times for each NP are shown in Fig. 2.

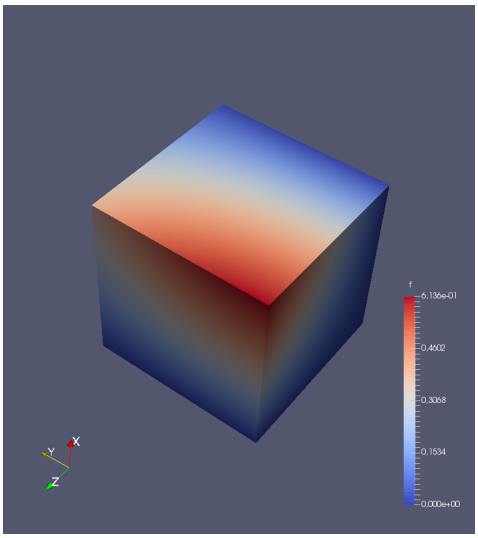

Figure 1: Numerical Solution.

Table 4: Result of the best domain decomposition obtained for Chapel and Fortran+MPI.

	Chapel			Fortran		
Threads	npx	npy	npz	npx	npy	npz
2	2	1	1	1	1	2
4	1	4	1	1	1	4
8	8	1	1	1	1	8
16	16	1	1	2	2	4
32	8	4	1	2	4	4

More precisely, when we divide the Fortran+MPI code execution time by the Chapel code execution time, we observe that Chapel's performance is better than Fortran+MPI for all cases tested. The best result was for 8 threads, when the Chapel version was 15% faster than the Fortran+MPI version. We explicitly show the values of the percentages in Tab. 5.

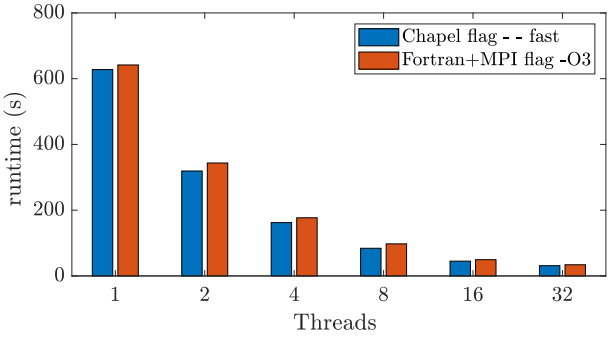

Figure 2: Runtime (in seconds) of the two codes as a function of the number of threads.

Table 5: Chapel's performance compared to Fortran + MPI

Threads	1	2	4	8	16	32
CPU time (%)	102.2	107.6	109.1	115.9	109.8	109.2

The speedup is calculated as $S(NP) = T_1/T_{NP}$, where T_1 is the corresponding sequential time and T_{NP} corresponds to the runtime using NP threads or process. In Fig. 3, we see that both implementations have the a similar speedup trend.

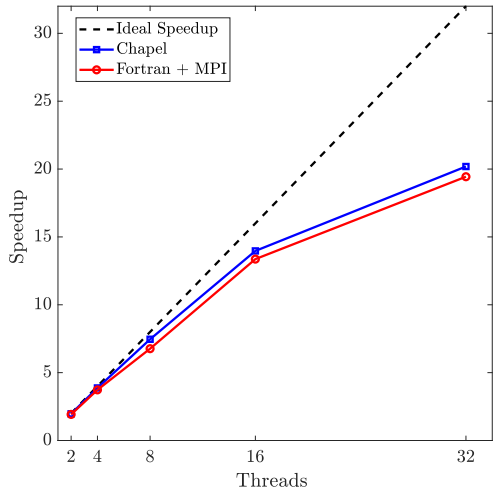

Figure 3: Speedup in linear scale. The dashed line corresponds to an ideal speedup.

4 CONCLUSION

In this paper, we investigate the potential of Chapel as an alternative to the conventional code implementation in Fortran+ MPI for the 3D Poisson problem. Chapel provides a cleaner and easier to program code, and we found that Chapel's implementation performs better

than Fortran+MPI's when return using `ref` intents in procedures (or subroutines). In addition, the chapel code provides an approximately 50% reduction in implemented lines compared to the Fortran+MPI code. Future work should extend this study to more nodes (multi locales in Chapel).

5 ACKNOWLEDGMENTS

We thank Leonardo Martinussi, Engin Kayralki, Damian McGuckin, Jeremiah Corrado, Vass Litvinov and the all Chapel community your great support!. This study was funded by the Coordination for the Improvement of Higher Education Personnel (CAPES grant PROEX 88887.671252/2022-00) and the São Paulo Research Foundation (FAPESP grant Nº. 2018/24284-1). The computational resources of the Center for Mathematical Sciences Applied to Industry (CeMEAI) is funded by FAPESP (grant 2013/07375-0)

REFERENCES

[1] James Ahrens, Berk Geveci, and Charles Law. 2005. *Visualization Handbook*. Elsevier Inc., Burlington, MA, USA, Chapter ParaView: An End-User Tool for Large Data Visualization, 717–731. https://www.sciencedirect.com/book/9780123875822/visualization-handbook

[2] R.J. LeVeque. 2007. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Society for Industrial and Applied Mathematics.

[3] Patrick J. Roache. 2001. Code Verification by the Method of Manufactured Solutions. *Journal of Fluids Engineering* 124, 1 (11 2001), 4–10. https://doi.org/10.1115/1.1436090 arXiv:https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/124/1/4/5901562/4_1.pdf