International Journal of Database Theory and Application
Vol.7, No.6 (2014), pp.105-120
http://dx.doi.org/10.14257/ijdta.2014.7.6.10

An Iterative Space Alternative Tiling Parallel Algorithm for 3D Finite Difference Stencil Computations

Jing Shen[1234], Jilin Zhang[123*], Jian Wan[123], Li Zhou[123] and Ming Jiang[123]

[1] *School of Computer Science and Technology, Hangzhou Dianzi University, 310018, Hangzhou, Zhejiang, China*
[2] *Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, China*
[3] *Zhejiang Provincial Engineering Center on Media Data Cloud Processing and Analysis, Hangzhou, China*
[4] *School of Information Engineering, Hangzhou Dianzi University, Hangzhou, Zhejiang, China*

sj@hdu.edu.com, jilin.zhang@hdu.edu.cn

Abstract

Stencils are finite-difference algorithms for solving large-scale and high-dimension partial differential equations. Due to the data dependences among the iterative statements in Stencils, traditional Stencil computations are be executed serially, rather than in parallel. It's challenging to design an effective and scalable Stencil parallelized method. To address the issue of 3D data space computing, we present a serial execution model based on multi-layers symmetric Stencil method and time skewing techniques. Within this model, the iteration space is divided to multiple tiles based on time skewing, where the executive process is ordered by the sequence of tiles, and the nodes in each individual tile can be swept repeatedly to improve the data locality. In addition, we propose a novel 3D iterative space alternate tiling Stencil parallel method, which subdivides the iteration space along high dimension, and changes the execution sequence of tiles to reduce the data dependency and communication cost, where the partial order of tiles is still guaranteed. Experimental results demonstrate our proposed alternative tiling parallel method achieves better parallel efficiency and scalability compared with the domain-decomposition methods.

Keywords: *Stencil algorithm; alternative tiling; multi-layer symmetric; polyhedral model; communication optimization*

1. Introduction

Various scientific and engineering computations can be modeled as a large-scale high-dimension partial differential equation (PDE for short), such as scalar-wave propagation modeling, earth modeling, heat conduction [1-3], etc. Due to the features including high dimension, large scale and high precision requirement, the PDE computations are usually solved by finite differential equations. To accelerate the computation efficiency, parallel computing has been applied as a popular manner for solving finite differential algorithms. Therefore, solving large-scale PDE by finite differential algorithm including Stencil algorithms [4-6] in parallel environment has received much attention from the research field of scientific computing.

Stencil computations (SC for short) sweep over the given data grid and compute every node according to its neighbor nodes along every dimension multiply iterations, as shown in Figure 1. In the case of 3-dimension (3D for short) data grid, as the large-scale data grid and great reuse distance, the cache miss rate of SC is high. And the data dependency exists among

ISSN: 2005-4270 IJDTA

the grid nodes, which result in poor parallelism. Some researches achieved the good optimization performance of 1-dimension (1D for short) or 2-dimension (2D for short) SC, however they cannot be directly used for 3D SC because of the great reuse distance of 3D data grid. In addition, due to the compute capacity has increased through the number of cores, while memory bandwidth is increasing slowly, the efficiency of SC is bounded by memory bandwidth.

In this paper, we propose a novel iterative space alternative tiling parallel algorithm of 3D SC that performs hierarchical division to improve data locality and iterative space alternative tiling to enhance the efficiency of parallelism. Firstly, the algorithm divides the data grid into tiles along the highest dimension to reserve data locality of the other dimensions as much as possible. Secondly, it utilizes time skewing technique to reuse the data blocked in cache for several iterations and revises the execution order of tiles to reduce the data dependency among the tiles. Thirdly, the tiles are divided to some sub-tiles to exploit the data locality and improve the intra-tile data reuse. Finally, use alternative tiling algorithm to execute the tiles. The method realizes the parallelism of 3D SC and reduces the bandwidth requirement with similar convergence.

We apply the parallel algorithm to 7-point Stencil for 3D data grid. The performance of our alternative tiling parallel optimization achieves better parallel efficiency and scalability compared with the domain-decomposition methods.

2. Related Work

There are mainly three parallel methods of SC, red-black ordering [7], multi-color ordering [8] and domain decomposition method [9-11]. Xie *et al.* [9-10] proposed point parallel successive over-relaxation (PPSOR) method based on domain-decomposition method and inter-process communication. If the data grid is massive, the data locality of this method trends down. Dursum et al. [7] proposed hierarchical parallel optimization based on red-black ordering. Xu *et al.*[11] proposed a parallel iterative algorithm for solving 2D Poisson equation based on domain-decomposition method to improve the convergence rate and decrease the number of iterations by introducing a relaxation factor. However it wasn't fit to 3D SC. Therefore, we present a novel alternative tiling parallel algorithm for 3D SC based on domain-decomposition method.

To date, the data reuse and parallelism for Stencil computation are far from being optimized. The main reason is the large volume of data grid and the inter-dependence among the neighboring nodes in the grid, shown in Figure 2, as the capacity of cache is limited, the data needs to be swapped in and out continually. Therefore, loop tiling techniques are used to improve data locality in the literatures [12-14], which focus on the shape and size of tiles. Tiling is generally along the direction of the data dependence vectors to reduce the inter-dependency between tiles. The size of tile depends on the cache capacity, aiming to data reuse maximization. We utilize the polyhedral model to implement the tiling method, and the size of tile is set to the cache size to reduce the overhead of tiling and memory bandwidth.

However, the improvement in data reuse brought by spatial tiling is limited. A node stays in the cache for a short time, and it is swapped out from the cache before its next iteration if the data grid is large. The literatures [7, 13, 15] made efforts to time skewing techniques which is the temporal tiling. Time skewing divides the iteration space (data space and time space) across several time steps, result in the reuse of grid data in cache for several time steps before being swapped out cache and written back to memory.

The data dependence among tiles poses challenges on the parallel execution of tiles and limits scalability of algorithm. Therefore, alternative tiling methods [16] are used to decrease the dependence between tiles. There are two methods for alternative tiling, including overlapped tiling [17] and split tiling [18]. The former makes every tile overlaps border data with its neighbor tiles to ensure them compute independently. The proportion of the border size to tile size needs to be decreased if possible. However the more is the number of parallel

computing nodes, the less is tile size, and the higher is the proportion of the border size to tile size, which results in poor scalability. Grosser *et al.* [18] proposed 1-dimension(1D for short) and 2-dimension(2D for short) Stencil automatic parallelization on GPU based on split tiling. This method reduced tile dependency by dividing iteration space to different shapes with limited extra communication cost, while disregarding the data overlap. However, this method cannot be used directly for 3D SC. Figure 2 depicts the great differences between 3D data space and 1D/2D, the dark grey units in Figure 2 are the nodes to be calculated, while the light grey units are the neighbors of the dark grey units. Figure 2(c) shows the distance between two neighbors of a grid node in Z-axis direction is far more than that in case of 1D and 2D. Split tiling for 1D/2D cannot obtain the optimal data locality for 3D SC. Therefore, according to the feature of 3D data storage structure (as shown in Figure 2(c)), we propose an optimization method to reduces the dependence of tiles without sacrificing similar the convergence ratio.

The remainder of this paper is organized as follows. In Section 2, we describe 7-point finite difference algorithm of 3D SC and the memory problems which hinder the efficiency and parallelization of 3D SC. Section 3 describes the multi-layer symmetric method for 3D Stencil algorithm and utilize polyhedral model to implement the tiling algorithm. Section 4 describes iterative space alternative tiling parallel optimization for 3D Stencil algorithm. Section 5 presents the influence of the tile size and shape to the alternative tiling parallel optimization and performance comparisons with some other optimization methods. Finally, we draw the conclusions in Section 6.

3. Background and Problem Statements

Considering the 3D Poisson equation [19]

$$\nabla^2 \varphi = \frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} + \frac{\partial^2 \varphi}{\partial z^2} = f(x, y, z) \tag{1}$$

With the boundary condition

$$\{(x, y, z) \mid \ 0 \le x \le a, 0 \le y \le b, 0 \le z \le c\} \tag{2}$$

The solution domain (Equation (2)) is partitioned into a set of grids, the space interval in x axis direction is $\Delta x = a/(I+1)$, the space interval in y axis direction is $\Delta y = b/(J+1)$, the space interval in z axis direction is $\Delta z = c/(K+1)$, the grid nodes are

$$\{(x_i, y_j, z_k) \mid x_i = i\Delta x, i = 0, ..., I; y_j = j\Delta y, j = 0, ..., J; z_k = k\Delta z, k = 0, ..., K\}$$

We use $u_{i,j,k}(i = 0, ..., I, j = 0, ..., J, k = 0, ..., K)$ to denote the finite difference approximation of $u(i\Delta x, j\Delta y, k\Delta z)$. By using the 7-point finite difference approximation of Equation (1), we can obtain that

$$\frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{(\Delta x)^2} + \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{(\Delta y)^2} + \frac{u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}}{(\Delta x)^2} = f(i, j, k) \tag{3}$$

Given $\Delta x = 1, \Delta y = 1, \Delta z = 1$, then,

$$u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k} + u_{i,j,k+1} + u_{i,j,k-1} - 6u_{i,j,k} = f(i, j, k) \tag{4}$$

We describe the 3D Stencil algorithm in the case of 7-point finite difference iterative algorithm. PDE generates a set of linear iterative equations, the initial value $u_{i,j,k}^{(0)}$ are in row-major order, show as (5), t is the number of iterations.

```
for t=0 to T-1 do
  for k = 1to nz – 1do
    for j = 1to ny – 1do
      for i = 1to nx – 1 do
      //statement S1
        A[i][j][k] = A[i][j][k+1] + A[i][j][k-1] +A[i][ j+1][k]
                   +A[i][j-1][k]+A[i+1][j][k]
                   +A[i-1][j][k]- 6.0 * A[i][j][k] / (factor*factor);
```

Figure 1. Traditional 7-point Difference Iterative Algorithm

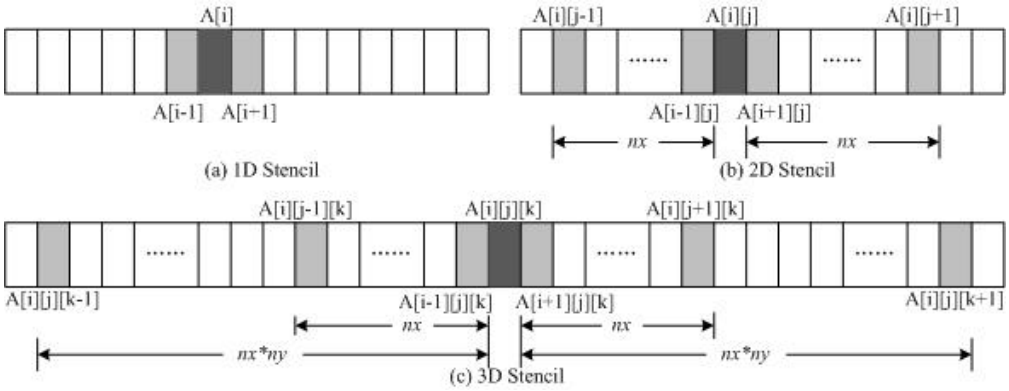


Figure 2. Comparison of 1D, 2D and 3D Stencil Memory Structures

$$u_{i,j,k}^{(t+1)} = u_{i,j,k+1}^{t} + u_{i,j,k-1}^{(t+1)} + u_{i,j+1,k}^{t} + u_{i,j-1,k}^{(t+1)} + u_{i+1,j,k}^{t} + u_{i-1,j,k}^{(t+1)} - 6u_{i,j,k}^{t} / f^{2} \quad i, j, k = 1, 2, 3, ..., n. \tag{5}$$

Figure 1 shows the traditional single-thread 7-point finite difference of the 3D Stencil algorithm. The algorithm sweeps over and updates all grid nodes order by its memory location multiple iterations.

The Execution of traditional algorithm has a high cache miss rate and poor data locality. The reason is the data updated this time have been already swapped out from cache before next access if the size of array is greater than cache available capacity. Furthermore, the 7 points are placed in 3 planes respectively, which against data reuse. Take the case of A[i][j][k+1] and A[i][j][k-1], if grid nodes are stored in row-major order, the reuse distance between the two nodes is 2 * *nx* * *ny* , that is, two *nx***ny*-sized planes need to be stored in the cache. Accordingly, assuming cache capacity is 32KB, the grid node is double words type, if *nx***ny* > 2048, a statement execution in a time needs to be swapped in and out for several times. Furthermore, in the above cases, the data address translation is accessed from TLB periodically, leading to a high TLB miss rate and a considerable performance decrease for large-scale arrays.

Therefore, we propose an iterative space multi-layer symmetric Stencil algorithm based on loop tiling, as shown in Figure 5, changing the traditional execution in iterative order to improve intra-tile data locality.

4. Iterative Space Multi-layer Symmetric 3D Stencil Algorithm

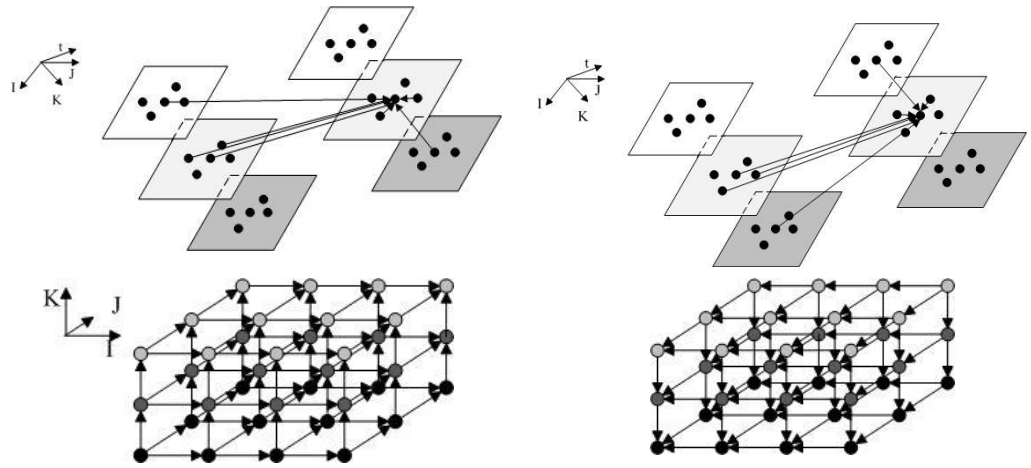
4.1. Multi-layer Symmetric Stencil Algorithm

In order to improve data locality of iterative tiles boundary, we transform the update sequence of unknown $u_{i,j,k}$ in Equation (5), the linear iterative equation can be expressed as,

$$u_{i,j,k}^{(t+1)} = u_{i,j,k+1}^{(t+1)} + u_{i,j,k-1}^{t} + u_{i,j+1,k}^{(t+1)} + u_{i,j-1,k}^{t} + u_{i+1,j,k}^{(t+1)} + u_{i-1,j,k}^{t} - 6u_{i,j,k}^{t} / f^2$$

$$i, j, k = n, n-1, n-2, \ldots 1. \quad (6)$$

The multi-layer symmetric 3D Stencil algorithm executes the calculation of Equation (5) or (6) alternatively layer by layer to solve the linear partial differential equation. Forward iterations are shown in Figure 3(a), where grid nodes $u_{i,j,k} = \{u_{i,j,k}^{t} \mid t \in [(p-1)T+1, p \times T]\}$, $p = 1, 3, 5, \ldots, 2P+1$ (P is a natural number) are forward iteratively computed (from front left bottom to back right top) with Equation (5), which is called *odd iteration*. Backward iteration is shown as Figure 3(b), gird nodes $u_{i,j,k} = \{u_{i,j,k}^{t} \mid t \in [(p-1)T+1, p \times T]\}$, $p = 2, 4, 6, \ldots, 2P$ (P is a natural number) are backward iteratively computed (from back right top to front left bottom) with Equation (6), which is called *even iteration*.



(a) forward iteration (b) backward iteration

Figure 3. Iterative Space Multi-layers Symmetric 7-point Stencil Method

4.2. Polyhedral Model Description

We use the polyhedral model to implement tiling hyperplane to minimize inter-tile dependency. Polyhedral model [20, 21] is a loop optimization method which uses iterative domains or affine transformations, *etc.*, to map loop iterations to polyhedral model in mathematics, then transforms the polyhedral model with complete polyhedral theory in mathematics, finally re-maps the results to loop in the execution. An iterative space polyhedron uses *n* nested loop as an *n*-dimension point set with a constraint $B.I \geq b$, where B is a set of boundary hyperplanes, I is the iterative vector. The statement S1 in the traditional 3D Stencil algorithm is shown in Figure 1, which depicts a 7-point differential computation. The iterative space of S1 can be expressed as the Equation (7), where T is the number of iterations.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} t \\ i \\ j \\ k \end{pmatrix} \geq \begin{pmatrix} 0 \\ -T+1 \\ 1 \\ -nx+1 \\ 1 \\ -ny+1 \\ 1 \\ -nz+1 \end{pmatrix} \quad (7)$$

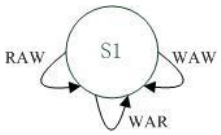


Figure 4. 3D Stencil Algorithm data Dependence Graph

The dependence of statement S1 is shown in Figure 4, which presents three dependences, WAR, RAW and WAW. According to these dependencies, we can obtain the optimal tile hyperplanes, which are (1,0,0,0), (1,1,0,0), (1,0,1,0) and (1,0,0,1), respectively. The affine transformation can be described as follows,

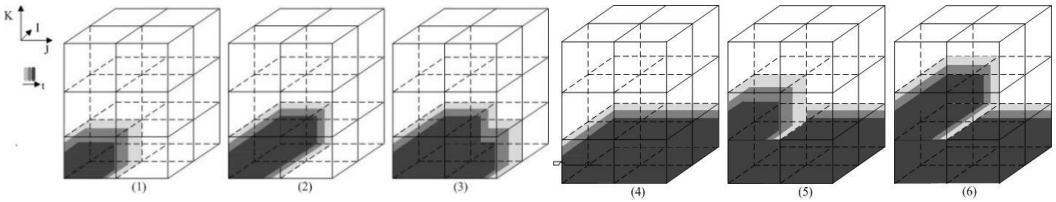
$$T_s\begin{pmatrix} t \\ i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} t \\ i \\ j \\ k \end{pmatrix} \tag{8}$$

Formula (8) describes the affine transformation of odd iteration, which is forward iteration. Formula (9) describes the affine transformation of even iterations, which are backward iterations. The optimal tile hyperplanes are (1,0,0,0), (1,-1,0,0), (1,0,-1,0) and (1,0,0,-1).

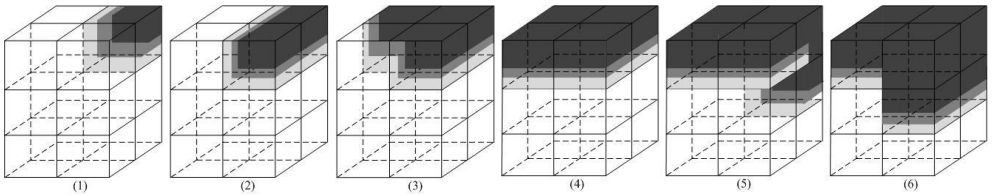
$$T_s\begin{pmatrix} t \\ i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{pmatrix}\begin{pmatrix} t \\ i \\ j \\ k \end{pmatrix} \tag{9}$$

4.3. Execution Model of Multi-layer Symmetric 3D Stencil Algorithm

Traditional SC sweeps and updates the entire iterative space for each iterative step, so the data locality is relative poor. We propose to introduce temporal dimension to the iterative space, use hyperplane matrixes (formula (8),(9)) to divide the iterative space into a number of grid tiles, and update them for multiple times to improve inter-tile data locality. Figure 5 shows the details of the procedure, where the grey blocks represent the tiles whose iterative updates are completed. The color depth denotes the iterative times. The darker are the blocks, the higher are the iterative times.



(a) Forward iteration(computation begins at front left bottom boundary, moves to back right top)



(b) Backward iteration(computation begins at back right top boundary,
(c) moves to front left bottom)

Figure 5. 3D Iterative Space Tiling Multi-layers Symmetric Stencil Method

To clarify the algorithm description, we give the formal definitions of data space and iteration space firstly.

Data space: *data_space*(*m*) indicates data space of iterative computation. It is composed of *m*-dimension grid nodes $x(i_1,\ldots,i_m)$, $i_1,\ldots,i_m$ indicate coordinate of grid nodes in $(I_1,\ldots,I_m)$ dimensions of data space, respectively.

Iterative space: *iter_space*$(I_1,\ldots,I_{n-1},T)$ indicates a *n*-dimensional iterative space consisted of a *n-1*-dimension data space *data_space*(*n-1*) and temporal dimension, which is composed of *n*-dimension nodes $u^t(i_1,\ldots,i_{n-1})$. $u^t(i_1,\ldots,i_{n-1})$ indicates *t* times iteration value of node $x(i_1,\ldots,i_{n-1})$. For example, *iter_space*(*I,J,K,T*) indicates the iterative space of 3D Stencil, which is consisted of *data_space*(3) and *T*, $u^t(i,j,k)$ indicates grid node of $x(i,j,k)$ in *t* iterations.

With the 3D Poisson equation, the execution of iterative space alternative tiling serial Stencil algorithm is performed by the following three steps:

(i) Data space division.

Divide the iterative space expressed as Equation (7) into several subspaces *sub_iter_space*(*p,q,r,*0) along IJK dimensions at $t = 0$, shown in Figure 5. The parameters *p, q, r* denote the tile numbers of the subspace along *IJK* dimensions respectively. For example, the grey blocks in Figure 5(b)(1) represent *sub_iter_space*(2,2,3,0). Assuming the number of grid nodes in subspace is $l_i*l_j*l_k$, where l_i, l_j and l_k are the numbers of nodes in subspace along with *I, J* and *K* dimensions respectively, these parameters meet the following Formula (10), where *T* is the number of one-direction iterations:

$$l_1>T \cap l_2> T \cap l_3> T \tag{10}$$

(ii) Iterative space division.

With time skewing, we employ the hyperplanes in Equation (8)(9) to divide the entire iteration space based on subspace *sub_iter_space*(*p,q,r,*0). Hyperplane in Equation (8) is for the iteration $t \in [(p-1)T+1,\ p\times T]$ (p is odd) and hyperplane in Equation (9) is for the iteration $t \in [(p-1)T+1,\ p\times T]$ (p is even). After division, we revise the boundary of tiles as the boundary gradient of grey blocks in Figure 4 (from light grey to dark grey). Revision algorithm is shown in Figure 6.

```
// revision algorithm of forward iteration
forall t ∈ [(p − 1)T + 1, p × T], p = 1,3,5,...,2P + 1 do
//np, nq, nr: the number of subspace of iter_space(I, J , K, 0) divided along IJK
//dimensions
  for r = 0 to nr-1do
    for q= 0 to nq -1 do
      for p = 0 to np-1 do
        if p != 0
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) + u^t (p*l_{i-1},j,k)
        if p != np-1
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) - u^t ((p+1)*l_{i-1},j,k)
        if q != 0
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) + u^t (i,q*l_{j-1},k)
        if q != nq-1
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) - u^t (i,(q+1)*l_{j-1},k)
        if r != 0
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) + u^t (i,j,r*l_{k-1})
        if r != nq-1
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) - u^t(i,j,(r+1)*l_{k-1})

// revision algorithm of backward iteration
```

```
forall t ∈ [(p − 1)T + 1, p × T], p = 2, 4, 6, ..., 2P do
  for r = nr-1to 0do
    for q= nq -1 to0do
      for p = np-1 to0do
        if p != 0
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) + u^t (p*l_{i-1},j,k)
        if p != np-1
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) - u^t ((p+1)*l_{i-1},j,k)
        if q != 0
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) + u^t (i,q*l_{j-1},k)
        if q != nq-1
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) - u^t (i,(q+1)*l_{j-1},k)
        if r != 0
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) + u^t (i,j,r*l_{k-1})
        if r != nq-1
          sub_iter_space(p,q,r,t) = sub_iter_space(p,q,r,t-1) - u^t(i,j,(r+1)*l_{k-1})
```

Figure 6. Pseudo-code of Boundary Revision Method

(iii) Compute Stencil order by tile number

There exists dependency among tiles, so the tiling algorithm needs to be executed in increasing order by the tile number during the forward iterations and decreasing order by the tile number during backward iterations. Figure 7 describes the procedures of forward and backward iterations. Forward and backward iteration are executed alternatively until the SC is completed. The order of tiles execution is shown in Figure 10(a).

```
//forward iteration
for r = 0 to nr-1 do
  for q = 0 to nq-1 do
    for p = 0 to np-1 do
      for t = beginT to endT do
        compute stencil in sub_iter_space(p,q,r,t)

//backward iteration
for r = nr-1 to 0 do
  for q = nq-1 to 0 do
    for p = np-1 to 0 do
      for t = beginT to endT do
        compute Stencil in sub_iter_space(p,q,r,t)
```

Figure 7. Pseudo-code of Multi-layer Symmetric Stencil Algorithm

The serial multi-layer symmetric algorithm contributes to the data locality of Stencil algorithm. However, according to the comparison between Figure 1 and Figure 7, we can find asymptotic time complexity of the algorithm is not optimal.

5. Parallel Alternative Tiling Stencil Algorithm

The multi-layer symmetric Stencil algorithm improves the traditional Stencil algorithm by loop tiling and time skewing technique, but there exists data dependence among the tiles, which hinders the process level parallel computation. Therefore, the tiles need to be rearranged to reduce the dependency.

5.1. Execution Model of Alternative Tiling Stencil Algorithm

Parallel alternative tiling Stencil algorithm can be described as follows, First, divide iterative space along *K* dimension at $t = 0$, and use the hyperplanes (Formula (11)) to divide the entire iterative space into several tiles based on the Step(i) in Section 4.3, which is represented by solid lines in Figure 8(a). Solid lines from light to black indicate the direction of iteration t increasing. The arrows indicate the dependencies among tiles.

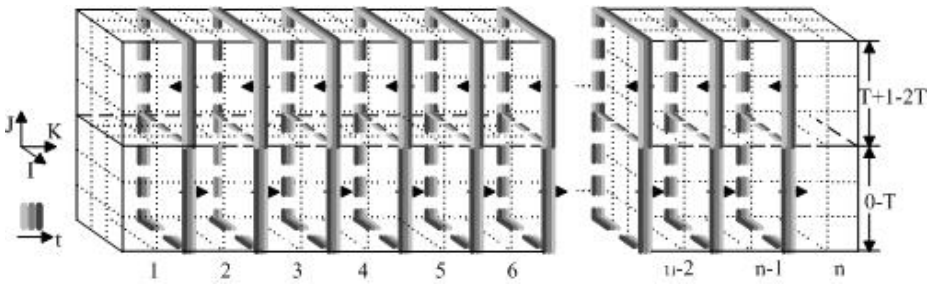
$$T_{s1}(t,k) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} t \\ k \end{pmatrix} \text{, } t \in [(p-1)T+1,\ p \times T] \text{, } p = 1,3,5,...,2P+1$$

$$T_{s1}(t,k) = \begin{pmatrix} 1 & 0 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} t \\ k \end{pmatrix} \text{, } t \in [(p-1)T+1,\ p \times T] \text{, } p = 2,4,6,...,2P \tag{11}$$

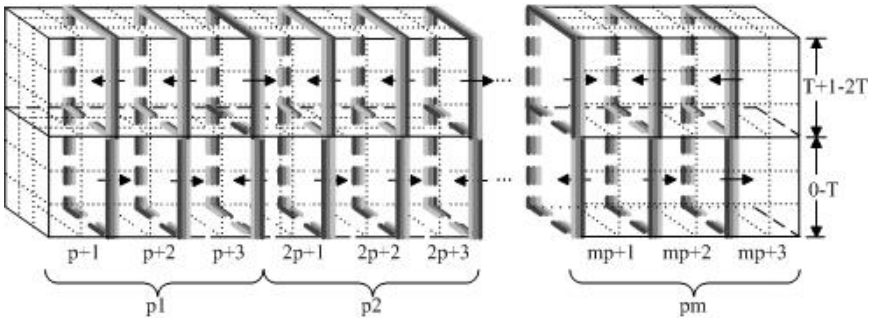
where *P* is a natural number.

We transform the execution order of tiles under the conditions of maintaining the partial orders of them, shown in Figure 8(b). The notations of *p1~pn* in Figure 8 indicate *n* processes. Compared with Figure 8(a), the dependency among tile *p*+3 and tile 2*p*+1 is changed, which makes *p*+1~ *p*+3, 2*p*+1~2*p*+3,…, *mp*+1~*mp*+3 able to be executed in parallel, so the degree of parallelism is *m*. However, the size of a tile is still large due to the complexity of the 3D data space. Therefore, we adopt loop tiling techniques in the internal iterations of tiles to improve data locality, which is shown as dash lines in Figure 8. Figure 10 shows that the data space is divided into 6 tiles and the execution order of these tiles computed by two processes in parallel. Figure 9 shows code rulers of tile number. Figure 10(b) shows the execution sequence of order-rearranged tiles. Two processes compute tiles in parallel, thus the degree of parallelism is 2. Obviously, the execution efficiency of the procedure shown in Figure 10(b) is twice faster than that shown in Figure 10(a). However, the procedure shown in Figure 10(b) requires communication and synchronization in processes, so the execution efficiency of the procedure shown in Figure 10(b) is less than twice of the one that shown in Figure 10(a). One of the biggest differences between Figure 10(a) and Figure 10(b) is the sequence of tiles in initialization, which makes the execution sequence of tiles changed and affects the dependency of tiles to improve the degree of parallelism.

The method shown in Figure 10(b) consumes communication and synchronization overhead during two processes implement iteration. In Figure 10(b), Processes 1 and 2 need to synchronize after implement of tile(2.1.1)~tile(2.2.2) to maintain inter-tiles data dependency in odd iterations. Process 1 needs to receive rear boundary data before computing tile(5.2.2) ~ tile(5.1.1). Similarly, Process 1 needs to send boundary data to Process 2 after computing tile(5.2.2) ~ tile(5.1.1), and Process 2 needs to receive these data before computing tile(2.1.1) ~ tile(2.2.2).



(a) Sequence from low to high along *k* axis



(b) (b) Sequence from reordering of tiles along k axis

Figure 8. Execution Sequence of Tiles along k Axis

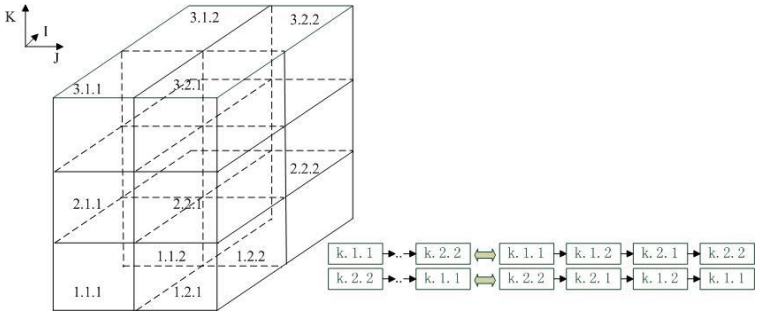
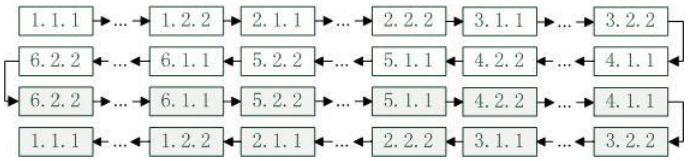
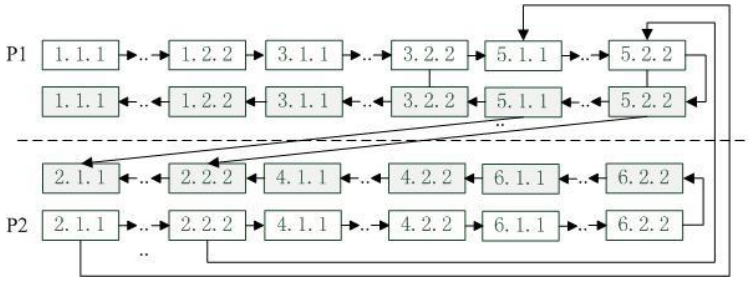


Figure 9. Tiling and the Rule of Allocate Tile Number



(a) Execution sequence of tiles from Fig 7(a)



(b) Execution sequence of tiles from Fig 7(b)

Figure 10. Execution Sequence of Tiles from Figure 4

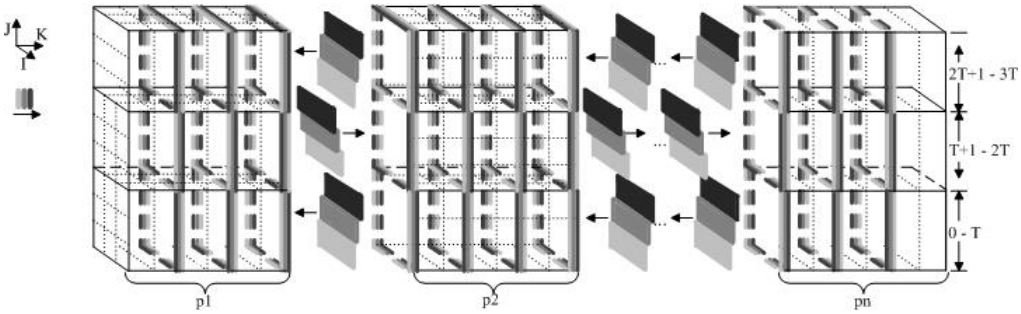


Figure 11. Parallel Execution Model of Multi Processes

5.2. Alternative Tiling Stencil Algorithm

The algorithm is extended to *m*-processor execution model in Figure 11, the execution can be described as follows.

(i) Divide the iterative space according to the number of processes.

We divide the iterative space into m subspaces along the K dimension at t = 0. According to the load balance, we should ensure that the number of grid nodes in the first subspace is equal to the last one. So the number of grid nodes in the first subspace increases by nx*ny*T/2, and the number of the last one reduces nx*ny*T/2, the number of grid nodes in the others is about nx*ny*nz/m. Then divide iterative space along the hyperplanes, which defined in Equation (11). The tile boundary revision algorithm is similar to Figure 6. The difference is that the boundary revisions are only along K dimension.

(ii) Reorder grid tiles.

Reorder all tiles in the condition of keeping the same partial order among the tiles.

(iii) Divide the subspaces.

Since the size of 3D data space is large, we need further subdivision to improve data locality. The division algorithm is similar to the Step (i) and Step (ii) in the multi-layer symmetric Stencil algorithm. The division algorithm uses hyperplanes defined in Formula (8)(9) respectively according to different iterative directions.

(iv) Execute odd iterations and even iterations alternatively for each tile.

Forward iterative computation is executed for odd T iterations and backward iterative computation for even T iterations. Specific algorithm is shown in Figure 11. The first and last tile in Figure 11 are the first and last tile in the execution space of a process, for example, tile(2*p*+1) and tile(2*p*+3) in Figure 8.

There is something to be worth noting for tile(1) and tile(*mp*+3) in Figure 8. A process need not to receive boundary data before it update tile(*mp*+3) and send boundary data after it update tile(1) during forward iterations. Similarly, the process need not to receive the boundary data before it updates tile(1) and send the boundary data after it updates tile(*mp*+3) during the backward iterations.

Our proposed iterative space alternative tiling method has acceptable convergence rate[22], because the method maintains the original partial order of tiles. The method can efficiently execute Stencil algorithm in parallel.

6. Result

In this section, we evaluate the impact of the tile size and shape to the alternative tiling parallel algorithm and conduct a performance comparison among three methods, including alternative tiling method, domain-decomposition method, and the origin method. The hardware configuration is 11-node Xeon Cluster, every node has two Intel Xeon 3.0GHz dual-core CPU, 4096Kb L2 cache and 2GB memory. The nodes network bandwidth is more than GB, the operating system is Redhat Linux 9.0. MPI runtime environment is mpich-1.2.7 from Argonne Lab. We use C language and PAPI to implement the algorithm.

6.1. Tile Size and Shape

The tile size and shape affect the data locality and execution efficiency of the algorithm. Let *n* be the number of nodes in a subspace, *ni*, *nj* and *nk* be the numbers of subspace along *I*, *J* and *K* axis respectively, so $n = ni*nj*nk$. After *T* iterations, the number of tiles has three classes values: *n-T*, *n* or *n+T*. Let *S* be the size of subspace for *T* iterations, the maximum value of *T* is expressed as Equation (12), *d* is the word length of double-precision floating-point.

$$S_{max} = ni*nj*(nk+T)*d \tag{12}$$

Table 1 lists the comparison results of L2 cache misses and execution time of 16M, 32M and 64M data grids with 100 iterations and S_{max} is 2*L*,*L*,1/2*L* and 1/4*L*, where *L* represents *L*2 cache size. As shown in Table 1, we can observe that the cache miss is minimum when $S_{max}=L/4$, where the execution efficiency is best when $S_{max} =L$. There are data space, code segments and some data used in execution, so L2 cache miss is not minimum when $S_{max} =L$, but in this case, the tile size is bigger, the number of tiles is smaller, which leads to overhead of tiling and communication is reduced. The performance is the best when $S_{max} =L$. Therefore, the tile size should be chosen the approximation of cache size and less than cache size.

Table 1. Comparisons of L2 Cache Misses and Execution Time in Different Partitions

S_{max}	16M		32M		64M	
	L2 cache misses	Time(s)	L2 cache misses	Time(s)	L2 cache misses	Time(s)
2L	8.38E+06	52.142	2.03E+08	116.450	3.01E+08	242.114
L	4.59E+06	52.032	1.43E+07	113.552	3.28E+07	237.408
1/2L	4.28E+06	52.188	9.91E+06	113.562	2.39E+07	237.553
1/4L	4.20E+06	52.240	9.27E+06	113.781	2.13E+07	238.070

Table 2 gives the comparison results of execution time in various shapes of partitions with 20 iterations. These partitions have same size but different shape, that is various numbers of nodes in *I*, *J* and *K* axis. When the size of nodes in plane *IJ* （*nx***ny***d*） is less than L2 cache size, subspace should be *nx***ny***nk*, that is, data space is only divided in *K* axis, the performance is the best. When the number of nodes in plane *IJ* is greater than L2 cache size, subspace size should be *nx***nj***nk*, that is, data space is divided in *J* and *K* axis, not *I* axis, the performance is the best. Division over the outer loop but not inner loop can preserve the data reuse and locality of low dimensions because reuse distance of outer loop is much longer than that of inner loop. In the case of 256*256*256, the performance is the best when tile shape is 256*128*4, because $T \in [1, 2]$ when *nk* is 2, thus time skewing technique is failed and inter-iterative data locality is reduced.

Table 2. Comparisons of L2 Cache Misses and Execution Time in Different Shape of Partitions

S	16*16*65536	16*32*32768	256*256*256
	Time(s)	Time(s)	Time(s)
8*8*2048	6.456	8.962	14.565
16*16*512	5.252	5.730	11.503
16*32*256	5.258	5.624	11.739
256*128*4	5.885	6.277	8.532
256*256*2	6.531	6.980	10.493

6.2. Comparison

Multi-color ordering and domain-decomposition are two common Stencil parallel algorithms. The former sets different colors to neighbor nodes and different color nodes can be computed in parallel. Degree of parallelism is same to the number of colors. The latter divides the nodes in iterative space to compute in parallel. They both require synchronization inter- and intra- iteration, which seriously affect parallel efficiency. But iterative space alternative tiling method executes iteration based on tile, introduces time skewing, and changes execution order of tiles to change inter-tile dependency. The tiles communicate and synchronize once with neighbors every T iterations. Therefore, the method use tiling to improve data locality and decrease communication start-up time to reduce communication, which result in improving efficiency of Stencil parallel algorithm. Furthermore, the method is not only applicable to shared memory machines but also distributed memory machines.

Table 3 lists execution time of alternate tiling and origin Stencil for four different sizes of data space. The performance of alternative tiling is 2X faster than original Stencil algorithm.

Table 3. Execution Time of Alternate Tiling and Original Stencil

Total Nodes	Alternative Tiling	Origin Stencil
16×16×32768	69.858	140.184
16×16×65536	134.93	280.461
256×256×256	184.34	359.715
16×32×65536	289.111	602.446

6.3. Parallelism and Scalability

We compare the speedup and efficiency of iterative space alternative tiling with domain-decomposition method for 16*16*65536. Figure 12 shows the tendency of speedup and efficiency when the number of processors increases. Efficiency of iterative space alternative tiling method is greater than that of domain-decomposition methods. The speedup of domain-decomposition method shows a trend of ascending followed by leveling off, but the execution time not decrease. The reason is that the overhead of communication and synchronization is too large. Our method adopts time skewing technique and changes inter-tile dependency to reduce the overhead efficiently.

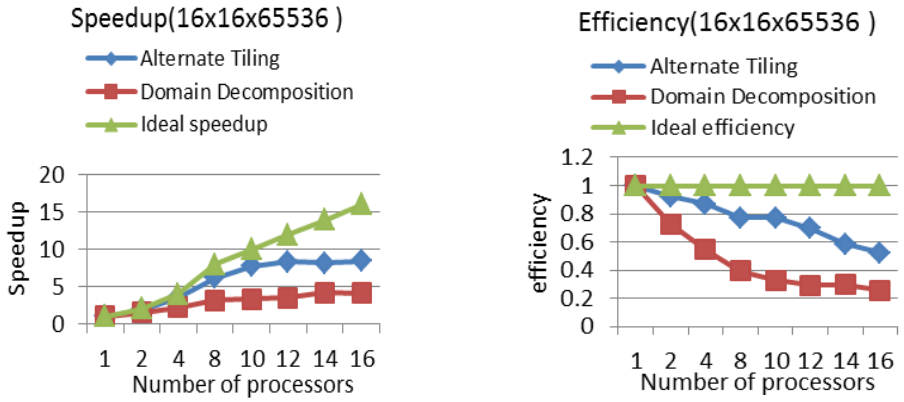


Figure 12. A Comparison of the Speedup and Efficiency between Alternate Tiling and Domain-decomposition Method on Different Numbers of Processors（16*16*65535）

7. Conclusion

In this paper, we analyzed the underlying reasons of low efficiency for traditional 3D Stencil algorithms and proposed an iterative space alternative tiling parallel method. In this method, the iterative space is divided in order to improve data locality. Moreover, the tiles of iterative space are reordered to reduce inter-tile dependency. Compared with the origin Stencil and domain-decomposition methods, our method achieved a lower overhead of communication start-up and synchronization without increasing the data exchange overhead, higher speed-up and better scalability. In future, we plan to analyze and model the performance of cluster storage structure, then design and realize general models of high dimensional Stencil algorithm to improve the parallelism and efficiency of iterative algorithm.

Acknowledgements

This research was supported by the Key Projects in the National Science-Technology Pillar Program (No. 2012BAH24B04), the National Natural Science Fund of China (No. 61202094, 61003077, 60873023, 60973029), the Fund of the Natural Science Foundation of Zhejiang province (No. Y13F020205), and Zhejiang Provincial Technical Plan Project (No. 2011C13008), the Open Project of Zhejiang Provincial Engineering Center (No.2012E10023-9); the Open Project of Zhejiang Provincial Engineering Center (No.2012E10023-9); the Commonweal Project Funds of Zhejiang Science & Technology Department (No.2013C31115).The computations of this research were performed in the Xeon Cluster of School of Computer Science and Technology, Hangzhou Dianzi University. The authors would like to express sincere appreciation for these means of support.

References

[1] S. R. Tan and L. J. Huang, "An efficient finite-difference method with high-order accuracy in both time and space domains for modelling scalar-wave propagation", Geophysical Journal International, vol. 197, no. 2, **(2014)**.
[2] J. Crank and P. Nicolson, "A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type", Advances in Computational Mathematics, vol. 1, **(1996)**,pp. 207-266.
[3] T. Gillberg, Ø. Hjelle and A. Magnus Bruaset, "Accuracy and efficiency of stencils for the eikonal equation in earth modelling", vol. 16, no. 4, **(2012)**, pp. 933-952.
[4] O. Davydov and D. T. Oanh, "Adaptive meshlesscentres and RBF stencils for Poisson equation", Journal of Computational Physics, vol. 230, no. 2, **(2011)**.
[5] J. Hietarinta and D. J. Zhang, "Hirota's method and the search for integrable partial difference equations", Journal of Different Equations and Applications, vol. 19, **(2013)**, pp. 1292-1316.
[6] K. K. Mattila, L. A. Hegele Jr. and P. C. Philippi, "High-Accuracy Approximation of High-Rank Derivatives: Isotropic Finite Differences Based on Lattice-Boltzmann Stencils", The Scientific World Journal, **(2014)**.
[7] H. Dursun, M. Kunaseth, K. I. Nomura, J. Chame, R. F. Lucas, C. Chen, M. Hall, R. K. Kalia, A. Nakano and P. Vashishta, "Hierarchical parallelization and optimization of high order stencil computation on multicore clusters", The Journal of Supercomputing, vol. 62, no. 2, **(2012)**, pp 946-966.
[8] L. Adams and J. Ortega, "A multi-color SOR method for parallel computation", International Conference on Parallel Processing, **(1982)**: Bellaire, MI, USA.
[9] D. Xie, "A new block parallel SOR method and its analysis", Siam Journal on Scientific Computing, vol. 27, no. 5, **(2006)**, pp. 1513-1533.
[10] D. Xie and L. Adams, "New parallel SOR method by domain partitioning", ICPP Workshops, **(2004)**, pp. 165-172.
[11] Q. Xu and W. Wang, "A new parallel iterative algorithm for solving 2D poisson equation", Numer. Methods Partial Differential Eq., **(2011)**, pp. 829-853.
[12] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf and K. Yelick, "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures", SC Proceedings of the ACM/IEEE conference on Supercomputing, IEEE Press, **(2008)**; Piscataway, NJ, USA.
[13] V. Bandishti, "Tiling stencil computations to maximize parallelism", SC '12 Proceedings of the International Conference on High Performance Computing, no. 40, **(2012)**.
[14] J. Shirako, K. Sharma, N. Fauzia, L. N. Pouchet, J. Ramanujam, P. Sadayappan and V. Sarkar, "Analytical Bounds for Optimal Tile Size Selection", Compiler Construction Lecture Notes in Computer Science, vol. 7210, **(2012)**,pp. 101-121.
[15] R. Strzodka, M. Shaheen, D. Pajak and H. P. Seidel, "Cache Accurate Time Skewing in Iterative Stencil Computations", International Conference on Parallel Processing, **(2011)**, pp. 571-581.

[16] S. Krishnamoorthy, M. Baskaran, U. Bondhugula,J. Ramanujam, A. Rountev and P. Sadayappan, "Effective automatic parallelization of stencil computations, **(2007),** pp. 235-244.
[17] J. Y. Meng and K. Skadron, "A Performance Study for Iterative Stencil Loops on GPUs with Ghost Zone Optimizations", International Journal of Parallel Programming, vol. 39, no. 1, **(2011),** pp. 115-142.
[18] T. Grosser, A. Cohen and P. H. J. Kelly, "Split Tiling for GPUs: Automatic Parallelization Using Trapezoidal Tiles", **(2013):** Houston, TX, USA.
[19] A. Shiferaw and R. C. Mittal, "Fast Finite Difference Solutions of the Three Dimensional Poisson's Equation in Cylindrical Coordinates", American Journal of Computational Mathematics.vol. 3, no. 4, **(2013).**
[20] L. N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, J. Ramanujam, P. Sadayappan and N. Vasilache, "Loop Transformations: Convexity, Pruning and Optimization", In 38th ACM ACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'11), **(2011),** pp. 549-562
[21] L. N. Pouchet, P. Zhang, P. Sadayappan and J. Cong, "Polyhedral-Based Data Reuse Optimization for Configurable Computing", FPGA '13 Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, **(2013),** pp. 29-38.
[22] V. K. Saul'yev, "Integration of Equations of Parabolic Type Equation by the Method of Net", Pergamon Press, **(1964).**

