Solving Heat Equation Of Higher-Dimensional Of Partial Differential Equations

Ahmed Sabah Ahmed Al-Jilawi
University Of Babylon / College of Education / Ibn Hayan

Abstract

The main objective of this paper is to solve the equation of heat that is part of the partial differential equations and large-scale (three dimensions) through the adoption of the method of division of memory and compressed and analysis of the field in the form of arrays for storage and collection of data and are evaluated technology through methods (subspace Krylov) The results showed that this method achieved a desirable speed through the speed of communication and exchange of data between processors, as well as the distribution of data among all participating processors by (Matlab Software)

الخلاصة ان الهدف الاساسي من هذا البحث هوحل معادلة الحرارة التي هي جزء من المعادلات التفاضلية الجزئيه واسعة النطاق (ثلاثي الابعاد) من خلال اعتماد اسلوب تقسيم الذاكره وضغطها وتحليل المجال على شكل مصفوفات لاغراض التخزين وجمع البيانات ويتم تقيم التكنلوجيا من خلال اساليب (فضاء جزئي كريلوف) واظهرت النتائج ان هذه الطريقة تحقق تسريع مرغوب فيه من خلال سرعة الاتصال وتبادل البيانات بين المعالجات وكذالك توزيع البيانات بين جميع المعالجات المشاركه عن طريق

(**Matlab Software**)

1. Introduction

A domain decomposition algorithm based on finite difference for heat equation was presented in [*K. A. Gallivan, and etc*]. The whole area is partitioned into several sub-domains, each of which is solved by the implicit method on the internal elements while handled by the classical explicit method on the boundary grids. A refined real-time parallel algorithm for time discretization of a partial differential evolution equation was proposed in [*U. Schendel, 1984*]. The method based on an Euler scheme combines coarse resolutions and independent fine resolutions in time in the same spirit as standard spatial approximations.

Despite the significant progress in developing parallel iterative solvers to increase the stability and convergence rate of algorithms , the storage and partition of data remains a challenge in handling large-scale heat equation and little work has been carried out on it. This could result from the strong correlation of data and large amounts of global communication between processors.

This paper describes a massive parallel computing scheme for heat equation. The primary goal of this research is to present a scalable parallel algorithm which could meet the intense demands on modeling capability. This is achieved by optimizing the following procedures: (1) Efficient domain decomposition method; (2) Efficient memory sharing and data distribution among all participating processors; (3) optimal parallelized iterative solver and preconditioners for handling linear systems of equations; (4) fast communication and data exchange among processors.

2. Classification of Partial Differential Equations

High-order partial differential equations, and systems of second-order PDEs, can usually be classified as parabolic, [*V. Hernandez, J.E. Roman, A.*] hyperbolic or elliptic. This classification gives an intuitive insight into the behaviour of the system itself. The general high-order PDE is of the form

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + ... = 0$$

Similar to the technique used to obtain an analytical solution,

$B^2 - 4AC < 0$ ——> Elliptic (e.g. Laplace Eq.)
$B^2 - 4AC = 0$ ——> Parabolic (e.g. Heat Eq.)
$B^2 - 4AC > 0$ ——> Hyperbolic (e.g. Wave Eq.)

Each category describes different phenomena. Mathematical operties correspond to those phenomena. Prototype problem, Heat Equation

1D $$\left[\frac{\partial^2 u}{\partial x^2}\right]k = \frac{\partial u}{\partial t}, \qquad (x) \in \Omega,\ t > 0$$

2D $$\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right]k = \frac{\partial u}{\partial t}, \qquad (x, y) \in \Omega,\ t > 0$$

3D $$\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right]k = \frac{\partial u}{\partial t}, \qquad (x, y, z) \in \Omega,\ t > 0$$

Where $\Omega = (0, M) \times (0, N) \times (0, L)$

Given initial temperature distribution as well as boundary temps.
(or rate of change of temp.) with

$$k = \frac{k'}{C\rho} = \text{Coefficient of thermal diffusivity}$$

Where k' = coef. of thermal conductivity
C = heat capacity
ρ = density

2 . Overview of Three-dimensional Heat Equation

The three-dimensional heat equation can be written as

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \frac{\partial u}{\partial t}, \quad (x, y, z) \in \Omega, t > 0 \quad , (1)$$

on the domain $\Omega = (0, M) \times (0, N) \times (0, L)$, with the initial condition $u = u^0, \quad t = 0$

and Dirichlet boundary conditions $u = g, \quad (x, y, z) \in \partial\Omega, t \geq 0$ where $u = u(x, y, z, t)$, $u^0 = u^0(x, y, z)$, $g = g(x, y, z, t)$.

This is a linear, second-order, parabolic partial differential equation. We discretize the left-hand side using seven-point stencil finite differences method and set $h_x = \frac{M}{IM}, h_y = \frac{N}{JN}, h_z = \frac{L}{KL}$, equation (1) can be rewritten as

$$\frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{h_x^2} + \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{h_y^2} + \frac{u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}}{h_z^2} = \frac{\partial u}{\partial t} \qquad (2)$$

where $0 \leq i \leq IM, 0 \leq j \leq JN, 0 \leq k \leq KL$.

Considering $h_x = h_y = h_z$, equation (2) can be simplified to the follow equation:

$$-6u_{i,j,k} + u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k} + u_{i,j,k+1} + u_{i,j,k-1} = \frac{\partial u}{\partial t} \cdot h_x^2 \qquad (3)$$

Applying equation $AU_i = -6u_{i,j,k} + u_{i,j+1,k} + u_{i,j-1,k} + u_{i,j,k+1} + u_{i,j,k-1}$ (4) to equation (3) yields equation $AU_i + u_{i+1,j,k} + u_{i-1,j,k} = h_x^2 \frac{\partial u}{\partial t}$ (5) which could be written in detail

$$\begin{bmatrix} A & I_1 & & & & \\ I_1 & A & I_1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & I_1 & A & I_1 \\ & & & & I_1 & A \end{bmatrix} \begin{bmatrix} U_1 \\ \\ \vdots \\ \\ \\ U_{m-1} \end{bmatrix} = h_x^2 \frac{\partial u}{\partial t}, \qquad (6)$$

where I_1 refers to a square matrix of order $(n-1)(l-1)$ and $U_i = \left[u_{i,1,1}, u_{i,1,2}, \cdots, u_{i,1,\ l-1}, u_{i,n-1,l-1} \right]^T$.

Applying equation $BU_{ij} = -6u_{i,j,k} + u_{i,j+1,k} + u_{i,j-1,k} + u_{i,j,k+1} + u_{i,j,k-1}$ (7) to equation (4) yields two equations $AU_i = BU_{ij} + u_{i,j+1,k} + u_{i,j-1,k}$ (8)

and $A=\begin{bmatrix} \mathrm{B} & \mathrm{I}_2 & & & & & \\ \mathrm{I}_2 & B & \mathrm{I}_2 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \mathrm{I}_2 & B & \mathrm{I}_2 & \\ & & & & \mathrm{I}_2 & B & \end{bmatrix}$ (9)

where I_2 refers to a square matrix of order $l-1$, $U_{ij}=\left[u_{i,\mathrm{j},1},\cdots,u_{i,j,l-1}\right]^T$ and

$$B=\begin{bmatrix} -6 & 1 & & & & & \\ 1 & -6 & 1 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -6 & 1 & \\ & & & & 1 & -6 & \end{bmatrix}_{(l-1)\times(l-1)} \quad (10).$$

For time discretization, it is handled by the backward Euler method which is a fully implicit method. Thus we have equation

$$\begin{cases} \dfrac{u_{i,j,k}^{n+1}-u_{i,j,k}^{n}}{\Delta t}=\dfrac{u_{i+1,j,k}^{n+1}-2u_{i,j,k}^{n+1}+u_{i-1,j,k}^{n+1}}{h_x{}^2}+ \\ \dfrac{u_{i,j+1,k}^{n+1}-2u_{i,j,k}^{n+1}+u_{i,j-1,k}^{n+1}}{h_y{}^2}+\dfrac{u_{i,j,k+1}^{n+1}-2u_{i,j,k}^{n+1}+u_{i,j,k-1}^{n+1}}{h_z{}^2} \\ 1\le i\le \mathrm{IM}-1, 1\le j\le \mathrm{JN}-1, \\ 1\le k\le \mathrm{KL}-1, \mathrm{n}\ge 0 \\ u_{i,j,k}^{n+1}=g_{i,j,k}^{n+1}, i=0 \text{ or } i=\mathrm{IM} \text{ or } j=0 \\ \text{or } j=\mathrm{JN} \text{ or } k=0 \text{ or } k=\mathrm{KL}, \mathrm{n}\ge 0 \end{cases} \quad (11)$$

where Δt represents the time step and $u_{i,j,k}^{n}=u_{i,j,k}\mid_t=n\Delta t$, $g_{i,j,k}^{n}=g_{i,j,k}\mid_t=n\Delta t$.

3 . Methodology and Implementation

3.1 Parallel partitioning of data

Developing an efficient data decomposition method is a first and critical step for a successful parallel scheme when it comes to a detailed, multimillion-to-multibillion-cell computational model. For the purpose of obtaining optimal performance, the program should balance the computation in each processor as well as minimize the communication resulting from the placement of adjacent elements to different processors. In this paper, the scheme of distributed array for

parallel-computing systems is adopted to store the structured grids which hide the data layout information and communication information behind. As shown in Figure.1, the whole data area is represented by a set of nodes which could be further divided into the local node and the ghost node. Ghost node is a node located at the bordering portions owned by neighboring processes while local node refers to the node located at the internal portions.

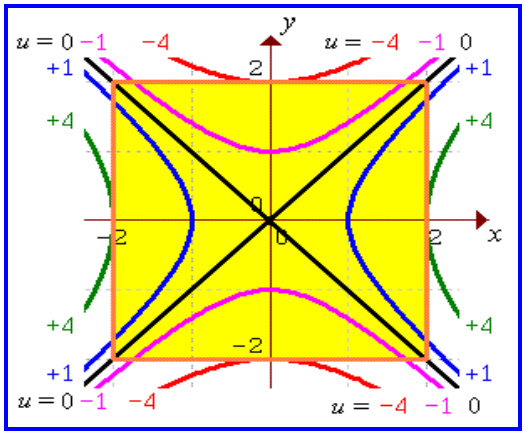

Figure.1. partitioning of data

3.2 The Sparse Matrix

The sparse matrix resulted from the finite-difference discretization of partial-differential equations is stored in a compressed matrix storage format such as AIJ which could also be varied at runtime to achieve optimal performance. Only nonzero items and their global entries of a submatrix for a partitioned mesh domain are stored on the processors in order to reduce the memory requirement. [*Zeyao Mo, Xiaowen Xu Volume 33, Issue 3, April 2007*]

As computational domain becomes larger, the assembling of sparse matrix becomes the key issue of achieving high performance. Figure.2 illustrates the layout of a sparse square matrix of order 5*5*5 from the finite-difference discretization of partial-differential equation of three-dimensional heat equation. As the whole domain is partitioned into several sub-domains, we develop a method to assemble the sparse matrix parallel based on distributed matrix storage. Defining *Istart* and *Iend* as the lower bounder and upper bounder of global index respectively, *Ii* as the index of the current row and assigning values to matrix row by row, then as illustrated in equations (6)-(10), the assembly of sparse matrix could be schematically outlined as follow.

```
for Ii := Istart to Iend do
 i := Ii /(n*l); j =: (Ii – i*n*l); k := Ii – i*n*l – j*l;
   if (i > 0) then Position(Ii, Ii – n*l) := 1;
   if (i < m – 1) then Position(Ii, Ii + n*l) := 1;
   if ( j > 0) then Position(Ii, Ii – l) := 1;
   if ( j < n – 1) then Position(Ii, Ii + l) := 1;
   if ( j > 0) then Position(Ii, Ii – 1) := 1;
   if ( j < l – 1) then Position(Ii, Ii + 1) := 1;
 Position(Ii, Ii) := –6;
endfor;
```

4. Performance Evaluation

The cluster is composed of four nodes, each of which contains a multi-core Intel Xeon CPU (two cores) and connected by kilomega Ethernet. The main frequency of CPU is 3.0GHz and memory space per node is 4GB. [Haberman, Richard 2003] We adopted structured mesh for spatial discretization, restarted GMRES as the iterative method, preconditioned with block Jacobi method (with one block per process, each of which is solved with ILU(0)).The convergence criterion used for iterative linear solver is based on the l_2-norm of the residual. Convergence is detected at iteration k if $\|r_k\|_2 < \max(10^{-5} * \|b\|_2, 10^{-50})$ where $r_k = b - Ax_k$, r_k and b are the residual and right-hand side, respectively.

Experiment was first carried out with discretization of 100*100*100 and 100 time steps. The speedup of backward and forward Euler method versus processor number is shown in Figure.2. The plot shows the speedup is almost linear as the processor number varies from 2 to 4 while there is a large deviation as the processor number increases to 8. This behavior was extensively analyzed that the limited memory space rather than the performance of CPU is mainly responsible for the decline in efficiency. As the number of processors increases, the size of the diagonal part of local domain and computational effort for computing the preconditioners on each processor decrease as ILU factorization is only performed on the diagonal. Therefore, the speedup of backward Euler method is slightly higher than forward Euler method as processor number increases.

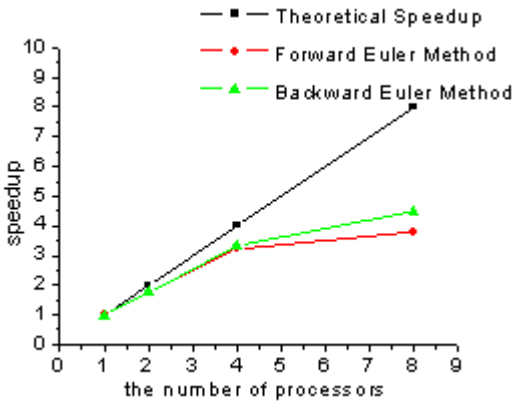

Figure.2. Speedup as processor number varies

Then we adjusted the spatial discretization from 100*100*8 to 100*100*128; and time discretization is based on fully implicit backward Euler method. The speedup and efficiency are shown in Table.1. From the table, the computational efficiency is 66.58% when the mesh size is 100*100*8 and it maintains at nearly 73%-74%. This could be attributed to the proportion of input/out and initialization would decrease as the problem scale expended. From our analysis, another reasonable reason could be that the memory space on single node is not enough as the demand for storage space may increase sharply. There are lots of data swapping in the process of computation and it would decrease the efficiency of serial program.

Table.1. Speedup and efficiency as the problem scale varies

Grids	Escaping Time (s)	Speedup	Efficiency
100*100*8	45.17	2.66	66.58%
100*100*16	79.09	2.92	73.03%
100*100*32	146.72	2.90	72.37%
100*100*64	286.50	2.99	74.84%
100*100*128	591.95	2.95	73.73%

5 - Application of Heat Equation

MATLAB, which is short for Matrix Laboratory, incorporates numerical computation, symbolic computation, graphics, and programming. As the name suggests, it is particularly oriented towards matrix computations, and it provides both state-of-the-art algorithms and a simple, easy to learn interface for manipulating matrices. In this tutorial, I will touch on all of the capabilities mentioned above: numerical and symbolic computation, graphics, and programming.

5-1 Example:

A 100 cm iron bar, with ρ=7.88, c=0.437, and κ=0.836, is chilled to an initial temperature of 0 degrees, and then heated internally with both ends maintained at 0 degrees. The heat source is described by the function

$$F(x,t) = 10^{-8} tx(100-x)^2.$$

The temperature distribution is the solution of

$$\rho c \frac{\partial u}{\partial t} - \kappa \frac{\partial^2 u}{\partial x^2} = F(x,t),\ 0 < x < 100,\ t > 0,$$
$$u(x,0) = 0,\ 0 < x < 100,$$
$$u(0,t) = 0,\ t > 0,$$
$$u(100,t) = 0,\ t > 0.$$

We will use standard piecewise linear basis functions and the techniques introduced in Section 5.6 of this tutorial to compute the mass and stiffness matrices:

```
n=100
n =
   100
h=100/n
h =
     1
k=0.836
k =
    0.8360
p=7.88
p =
    7.8800
c=0.437
c =
    0.4370

K=sparse(n-1,n-1);
for ii=1:n-1
   K(ii,ii)=2*k/h;
end
for ii=1:n-2
   K(ii,ii+1)=-k/h;
   K(ii+1,ii)=K(ii,ii+1);
end

M=sparse(n-1,n-1);
for ii=1:n-1
   M(ii,ii)=2*p*c*h/3;
```

```
end
for ii=1:n-2
   M(ii,ii+1)=p*c*h/6;
   M(ii+1,ii)=M(ii,ii+1);
end
```

(Note that, for this constant coefficient problem, we do not need to perform any integrations, as we already know the entries in the mass and stiffness matrices.)

Now I compute the load vector. Here is the typical entry:

```
clear ii
syms x t ii
phi1=(x-(ii-1)*h)/h
phi1 =
x-ii+1
phi2=-(x-(ii+1)*h)/h
phi2 =
-x+ii+1
F=10^(-8)*t*x*(100-x)^2
F =
1/100000000*t*x*(100-x)^2
int(F*subs(phi1),x,ii*h-h,ii*h)+int(F*subs(phi2),x,ii*h,i
i*h+h)
ans =
1/500000000*t*(ii^5-(ii-1)^5)+1/4*(-1/500000*t+1/10000000
0*t*(-ii+1))*(ii^4-(ii-1)^4)+1/3*(1/10000*t-1/500000*t*(-
ii+1))*(ii^3-(ii-1)^3)+1/20000*t*(-ii+1)*(ii^2-(ii-1)^2)-
1/500000000*t*((ii+1)^5-ii^5)+1/4*(1/500000*t+1/100000000
*t*(ii+1))*((ii+1)^4-ii^4)+1/3*(-1/10000*t-1/500000*t*(ii
+1))*((ii+1)^3-ii^3)+1/20000*t*(ii+1)*((ii+1)^2-ii^2)
simplify(ans)
ans =
-1/3000000*t+1/100000000*t*ii^3-1/500000*t*ii^2+20001/200
000000*t*ii
```

Now I need to turn this formula into a vector-valued function that I can pass to **beuler**. I write an M-file function

```
function y=f(t,n)
ii=(1:n-1)';
y=-1/3000000*t+1/100000000*t*ii.^3-1/500000*t*ii.^2+...
   20001/200000000*t*ii;
```

(Note the clever MATLAB programming in **f6**: I made **ii** a vector with components

equal to 1,2 ,...,*n*-1.
Then I can compute the entire vector in one command rather than filling it one component at a time in a loop.)
Next I create the initial vector **a0**. Since the initial value in the IBVP is zero, **a0** is the zero vector:

```
a0=zeros(n-1,1);
```

Now I choose the time step and invoke **beuler**:

```
dt=2;
N=180/dt;
[a,t]=beuler(M,K,'f6',a0,N,dt);
```

The last column of a gives the temperature distribution at time *t*=180 (seconds). I will put in the zeros at the beginning and end that represent the Dirichlet conditions:

```
T=[0;a(:,N+1);0];
xx=linspace(0,100,n+1)';
```

Here is a plot of the temperature after 3 minutes:

```
plot(xx,T)
```

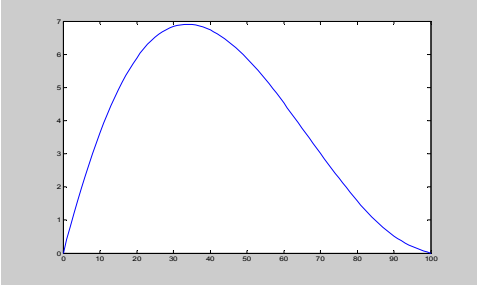

Fig 3 Multiple spatial dimensions

6. Conclusions

In this paper, The compressed matrix storage format is introduced and parallel assembly of large scale sparse matrix is implemented in our work. has achieved great improvement in computational efficiency and could significantly enhance the modeling capability. an efficient parallel method for large-scale three-dimensional equations is developed which integrates all kinds of most up-to-date parallel iterative solvers and preconditioners for solving large sparse matrix systems of discrete equations. Through the scheme of domain decomposition and distributed memory technology, the requirements of intensive computational ability and large amounts of memory space are distributed among and share by all processors of cluster. To further improve parallel performance, we devote intensive effort to the storage and assembly of sparse matrix.

References

Gallivan, K. A. Michael T. ,Heath, Esmond Ng, Ortega Applied Partial Differential Equations ,1983

U. Schendel, Introduction to Numerical Methods for Parallel Computers, Ellis Horwood Limited, 1984

R. J. Plemmons, Charles H. Romine, A. H. Sameh, Robert G. Voigt, Parallel Algorithms for Matrix Computations, Society for Industrial and Applied Mathematics, 1990

Haberman, R. Applied Partial Differential Equations. Edition 4.Prentice

Hall, 2003

MO Zeyao, Xiaowen Xu, Relaxed RS0 or CLJP Coarsening Strategy for Parallel AMG, Parallel Computing, Volume 33, Issue 3, April 2007

V. Hernandez, J.E. Roman, A. Tomas, Parallel Arnoldi Eigensolvers with Enhanced Scalability Via Global Communications Rearrangement, Parallel Computing, Volume 33, Issues 7-8, August 2007