A Distributed Memory Parallel Fourth-Order IADEMF Algorithm

Noreliza Abu Mansor[1], Ahmad Kamal Zulkifle[3]
College of Engineering
Universiti Tenaga Nasional
Selangor, Malaysia

Norma Alias[2]
Ibnu Sina Institute of Fundamental Science Studies, Universiti Teknologi Malaysia, Johor, Malaysia

Mohammad Khatim Hasan[4]
Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Selangor, Malaysia

***Abstract*—The fourth-order finite difference Iterative Alternating Decomposition Explicit Method of Mitchell and Fairweather (IADEMF4) sequential algorithm has demonstrated its ability to perform with high accuracy and efficiency for the solution of a one-dimensional heat equation with Dirichlet boundary conditions. This paper develops the parallelization of the IADEMF4, by applying the Red-Black (RB) ordering technique. The proposed IADEMF4-RB is implemented on multiprocessor distributed memory architecture based on Parallel Virtual Machine (PVM) environment with Linux operating system. Numerical results show that the IADEMF4-RB accelerates the convergence rate and largely improves the serial time of the IADEMF4. In terms of parallel performance evaluations, the IADEMF4-RB significantly outperforms its counterpart of the second-order (IADEMF2-RB), as well as the benchmarked fourth-order classical iterative RB methods, namely, the Gauss-Seidel (GS4-RB) and the Successive Over-relaxation (SOR4-RB) methods.**

Keywords*—Fourth-order method; finite difference; red-black ordering; distributed memory architecture; parallel performance evaluations

I. INTRODUCTION

The heat equation is a mathematical model that describes heat conduction processes of a physical system. Sahimi et al. [1] had proposed a finite difference scheme known as the Iterative Alternating Decomposition Explicit (IADE) method to approximate the solution of a one-dimensional heat equation with Dirichlet boundary conditions. The IADE scheme employs the fractional splitting of the Mitchell and Fairweather (MF) variant whose accuracy is of the order, $O\left((\Delta t)^2+(\Delta x)^4\right)$. The scheme, commonly abbreviated as the IADEMF, is developed by applying the second-order spatial accuracy to the heat equation. Due to the latter, in this paper, the IADEMF will also be referred to as the IADEMF2. It is a two-stage iterative procedure and has been proven to have merit in terms of convergence, stability and accuracy. It is generally found to be more accurate than the classical Alternating Group Explicit class of methods [2].

Several studies have later been developed based on the IADE method. Sahimi et al. [3, 4] developed new second-order IADE methods using different variants such as the D'Yakonov (IADEDY) and the Mitchell-Griffith variant (IADEMG). Each variant is of the order, $O\left((\Delta t)^2+(\Delta x)^4\right)$. The studies showed that the accuracies of the IADEDY and the IADEMG are comparable to the IADEMF. Alias [5] studied the parallel implementation of the IADEMF on distributed parallel computing using the parallel virtual machine. A fragmented numerical algorithm of the IADEMF method was designed by Alias [6] in terms of the data-flow graph where its parallel implementation using LuNA programming system was then executed. Sulaiman et al. [7, 8] proposed the half-sweep and the quarter-sweep IADEMF methods respectively, for the purpose of achieving better convergence rate and faster execution time than the corresponding full-sweep method. Alias [9] implemented the Interpolation Conjugate gradient method to improve the parallel performance of the IADEMF. Shariffudin et al. [10] presented the parallel implementation of the IADEDY for solving a two-dimensional heat equation on a distributed system of Geranium Cadcam cluster (GCC) using the Message Passing Interface.

A recent study made by Mansor [11] involved the development of a convergent and unconditionally stable fourth-order IADEMF sequential algorithm (IADEMF4). The proposed scheme is found to be capable of enhancing the accuracy of the original corresponding method of the second-order, that is, the IADEMF2. The IADEMF4 seems to be more accurate, more efficient and has better rate of convergence than the benchmarked fourth-order classical iterative methods, namely, the Gauss-Seidel (GS4) and the successive over-relaxation (SOR4) methods. However, the IADEMF4 may be too slow to be implemented especially when the problem involves larger linear systems of equations. It is thus justified to consider parallel computing to speed up the execution time without compromising its accuracy. The algorithm has explicit features which add to its advantage, thus it can be fully utilized for parallelization.

This paper attempts to parallelize the IADEMF4, by applying the Red-Black (RB) ordering technique, for solving large sparse linear systems that arise from the discretization of the one-dimensional heat equation with Dirichlet boundary conditions. It aims to effectively implement the IADEMF4-RB on parallel computers, with improved performance over its serial counterpart. The high computational complexity of the IADEMF4-RB will be implemented on multiprocessor distributed memory architecture based on Parallel Virtual Machine (PVM) environment with Linux operating system.

This paper is outlined as follows. Section II recalls the formulation of the IADEMF4 scheme. Section III presents the development of the IADEMF4-RB parallel strategy. The computational complexity of the RB methods considered in this paper is given in Section IV. Section V shows the numerical experiment conducted in this study. The results and discussion on parallel performance of the methods under consideration are discussed in Section VI. At the end of this paper is the conclusion.

II. Formulation of the IADEMF4 (An Overview)

In this section, the development of the IADEMF4 algorithm [11] is briefly reviewed. Consider the one-dimensional heat equation (1) which models the flow of heat in a homogeneous unchanging medium of finite extent, in the absence of heat source.

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2} \tag{1}$$

subject to given initial and Dirichlet boundary conditions

$$\begin{aligned} &U(x,0) = f(x), \quad 0 \le x \le 1 \\ &U(0,t) = g(t), \quad 0 < t \le T \\ &U(1,t) = h(t), \quad 0 < t \le T \end{aligned} \tag{2}$$

Based on the finite difference approach, the time-space domain is discretized by using a set of lines parallel to the $t-$ axis given by $x_i = i\Delta x$, $i = 0,1, \ldots, m, m+1$ and a set of lines parallel to the $x-$axis given by $t_k = k\Delta t$, $k = 0,1, \ldots, n, n+1$. The grid spacings have uniform size, that is, $\Delta x = 1/(m+1)$ and $\Delta t = T/(n+1)$. At a grid-point $P(x_i, t_k)$ in the solution domain, the dependent variable $U(x,t)$ which represents the non-dimensional temperature at time t and at position x, is approximated by u_i^k.

The IADEMF4 is developed by firstly executing the unconditionally stable fourth-order Crank-Nicolson approximation (3) on the heat equation [12].

$$\frac{1}{\Delta t}(u_i^{k+1} - u_i^k) = \frac{1}{2(\Delta x)^2}(\delta_x^2 - \frac{1}{12}\delta_x^4)(u_i^{k+1} + u_i^k) \tag{3}$$

The discretization of (3) leads to the expression given in (4), with the constants defined as in (5).

$$\begin{aligned} &au_{i-2}^{k+1} + bu_{i-1}^{k+1} + cu_i^{k+1} + du_{i+1}^{k+1} + eu_{i+2}^{k+1} = -au_{i-2}^k - bu_{i-1}^k \\ &+\hat{c}u_i^k - du_{i+1}^k - eu_{i+2}^k, \quad i = 2,3, \ \ldots, \ m-1 \end{aligned} \tag{4}$$

$$a = \frac{\lambda}{24},\ b = -\frac{2\lambda}{3},\ c = \frac{4+5\lambda}{4},\ d = -\frac{2\lambda}{3},\ e = \frac{\lambda}{24},\ \hat{c} = \frac{4-5\lambda}{4} \tag{5}$$

In matrix form, the approximation in (4) can be represented by $A\mathbf{u} = \mathbf{f}$ (6), where A is a sparse penta-diagonal coefficient matrix, and the column vectors $\mathbf{u} = (u_2, u_3, ..., u_{m-2}, u_{m-1})^T$ contain the unknown values of u at the time level $k+1$ and $\mathbf{f} = (f_2, f_3, ..., f_{m-2}, f_{m-1})^T$ consists of boundary values and known u values at the previous time level k.

$$A\mathbf{u} = \mathbf{f}$$

$$\begin{bmatrix} c & d & e & & & & \\ b & c & d & e & & \mathrm{O} & \\ a & b & c & d & e & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & a & b & c & d. & e \\ & \mathrm{O} & & a & b & c & d \\ & & & & a & b & c \end{bmatrix}_{(m-2)x(m-2)} \begin{bmatrix} u_2 \\ u_3 \\ . \\ . \\ . \\ u_{m-2} \\ u_{m-1} \end{bmatrix}_{k+1} = \begin{bmatrix} f_2 \\ f_3 \\ . \\ . \\ . \\ f_{m-2} \\ f_{m-1} \end{bmatrix} \tag{6}$$

The entries in $\mathbf{f}$ are defined as

$$\begin{aligned} f_2 &= -b(u_1^k + u_1^{k+1}) + \hat{c}u_2^k - du_3^k - eu_4^k \\ f_3 &= -a(u_1^k + u_1^{k+1}) - bu_2^k + \hat{c}u_3^k - du_4^k - eu_5^k \\ f_i &= -au_{i-2}^k - bu_{i-1}^k + \hat{c}u_i^k - du_{i+1}^k - eu_{i+2}^k, \quad i = 4,5,...,m-3 \\ f_{m-2} &= -au_{m-4}^k - bu_{m-3}^k + \hat{c}u_{m-2}^k - du_{m-1}^k - e(u_m^k + u_m^{k+1}) \\ f_{m-1} &= -au_{m-3}^k - bu_{m-2}^k + \hat{c}u_{m-1}^k - d(u_m^k + u_m^{k+1}) \end{aligned} \tag{7}$$

The IADEMF4 scheme secondly employs the fractional splitting of the higher-order accuracy formula of the MF variant [13],

$$(rI + G_1)\mathbf{u}^{(p+1/2)} = (rI - gG_2)\mathbf{u}^{(p)} + \mathbf{f} \tag{8}$$

$$(rI + G_2)\mathbf{u}^{(p+1)} = (rI - gG_1)\mathbf{u}^{(p+1/2)} + g\mathbf{f} \tag{9}$$

where G_1 and G_2 are two constituent matrices and r, I and p represent an acceleration parameter, an identity matrix and the iteration index respectively. The value of g is defined as $g = \frac{6+r}{6}$, $r > 0$. The vectors $\mathbf{u}^{(p+1)}$ and $\mathbf{u}^{(p+1/2)}$ represent the approximate solution at the iteration level $(p+1)$ and at some intermediate level $(p+1/2)$, respectively.

After some algebraic manipulations for the equations in (8) and (9), the form, $\left[G_1 + G_2 - \frac{1}{6}G_1G_2\right]\mathbf{u} = \mathbf{f}$ is obtained, suggesting that matrix A in (6) can be decomposed into.

$$A = G_1 + G_2 - \frac{1}{6}G_1G_2 \tag{10}$$

To retain the penta-diagonal structure of A, the matrices G_1 and G_2 have to be in the form of lower and upper tri-diagonal matrices respectively, Thus,

$$G_1 = \begin{bmatrix} 1 & & & & \\ l_1 & 1 & & & O \\ \hat{m}_1 & l_2 & \ddots & & \\ & \hat{m}_2 & \ddots & \ddots & \\ & & \ddots & l_{m-4} & 1 \\ & O & & \hat{m}_{m-4} & l_{m-3} \quad 1 \end{bmatrix}_{(m-2)x(m-2)}$$

and

$$G_2 = \begin{bmatrix} \hat{e}_1 & \hat{u}_1 & \hat{v}_1 & & & \\ & \hat{e}_2 & \hat{u}_2 & \hat{v}_2 & O & \\ & & \ddots & \ddots & \ddots & \\ & O & & \hat{e}_{m-4} & \hat{u}_{m-4} & \hat{v}_{m-4} \\ & & & & \hat{e}_{m-3} & \hat{u}_{m-3} \\ & & & & & \hat{e}_{m-2} \end{bmatrix}_{(m-2)x(m-2)} \quad (11)$$

If each G_1 and G_2 in (11) is substituted into the matrix A in (10), then the new entries of the latter can be compared with those in (6) to yield the following constants.

$$\hat{e}_1 = \frac{6(c-1)}{5}, \quad \hat{u}_1 = \frac{6d}{5}, \quad l_1 = \frac{6b}{6-\hat{e}_1}, \quad \hat{e}_2 = \frac{6(c-1) + l_1\hat{u}_1}{5},$$

$$\hat{v}_i = \frac{6e}{5} \text{ where } i = 1,2, \ ..., m-4$$

for $i = 2,3, \ ..., m-3$

$$\hat{u}_i = \frac{6d + l_{i-1}\hat{v}_{i-1}}{5}, \quad \hat{m}_{i-1} = \frac{6a}{6-\hat{e}_{i-1}}, \quad l_i = \frac{6b + \hat{m}_{i-1}\hat{u}_{i-1}}{6-\hat{e}_i},$$

$$\hat{e}_{i+1} = \frac{6(c-1) + l_i\hat{u}_i + \hat{m}_{i-1}\hat{v}_{i-1}}{5} \quad (12)$$

Since G_1 and G_2 are three banded matrices, then it is easy to obtain the inverses of $(rI + G_1)$ and $(rI + G_2)$. By rearranging the equations in (8) and (9), the following expressions are obtained.

$$\mathbf{u}^{(p+1/2)} = (rI + G_1)^{-1}(rI - gG_2)\mathbf{u}^{(p)} + (rI + G_1)^{-1}\mathbf{f} \quad (13)$$

$$\mathbf{u}^{(p+1)} = (rI + G_2)^{-1}(rI - gG_1)\mathbf{u}^{(p+1/2)} + g(rI + G_2)^{-1}\mathbf{f} \quad (14)$$

Based on the above two equations, the computational formulae at each of the half iteration levels can be derived as given in (15) and (16).

1) At the (p+1/2) iteration level:

$$u_i^{(p+1/2)} = \frac{1}{R}(E_{i-1}u_i^{(p)} + W_{i-1}u_{i+1}^{(p)} + V_{i-1}u_{i+2}^{(p)} - \hat{m}_{i-3}u_{i-2}^{(p+1/2)} - l_{i-2}u_{i-1}^{(p+1/2)} + f_i), \quad i = 2,3,...,m-2,m-1 \quad (15)$$

2) At the (p+1) iteration level:

$$u_i^{(p+1)} = \frac{1}{Z_{i-1}}(S_{i-3}u_{i-2}^{(p+1/2)} + Q_{i-2}u_{i-1}^{(p+1/2)} + Pu_i^{(p+1/2)} - \hat{u}_{i-1}u_{i+1}^{(p+1)} - \hat{v}_{i-1}u_{i+2}^{(p+1)} + gf_i), \quad i = m-1,m-2,...,3,2 \quad (16)$$

with

$$\hat{m}_{-1} = \hat{m}_0 = l_0 = V_{m-2} = V_{m-3} = W_{m-2} = \hat{u}_{m-2} = \hat{v}_{m-2} = \hat{v}_{m-3} = Q_0 = S_{-1} = S_0 = 0$$

$$R = 1 + r, \quad P = r - g,$$

$$E_i = r - g\hat{e}_i, \quad Z_i = r + \hat{e}_i, \quad i = 1,2, \ ..., m-2$$

$$W_i = -g\hat{u}_i, \quad Q_i = -gl_i, \quad i = 1,2, \ ..., m-3$$

$$V_i = -g\hat{v}_i, \quad S_i = -g\hat{m}_i, \quad i = 1,2, \ ..., m-4 \quad (17)$$

The two-stage IADEMF4 algorithm is implemented by using the required equations at the two iteration levels in alternate sweeps along all the grid-points in the interval (0,1) until convergence is reached. The method is explicit, since at each level of iteration, the computational molecules involve two known grid-points at the new level and another three known ones at the old level (Fig. 1 and 2). The unknown $u_i^{(p+1/2)}$ in (15) is calculated by proceeding from the left boundary towards the right, whereas the unknown $u_i^{(p+1)}$ in (16) is calculated from the right boundary and moves to the left.

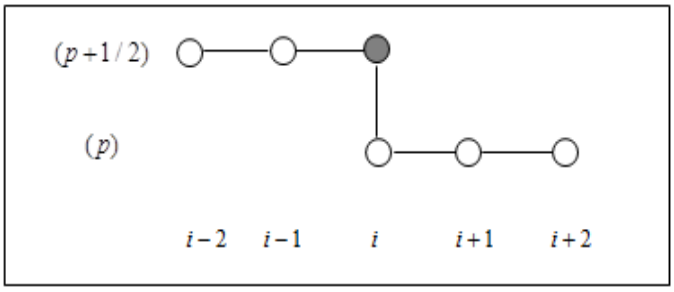

Fig. 1. Computational Molecule of the IADEMF4 at the $(p+1/2)$ Iteration Level.

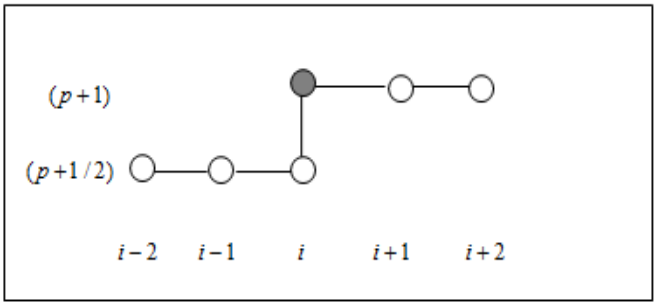

Fig. 2. Computational Molecule of the IADEMF4 at the $(p+1)$ Iteration Level.

III. PARALLELIZATION OF THE IADEMF4

It is observed that for $i = 2,3, \ ..., \ m-1$, the computation of the unknown grid-point, $u_i^{(p+1/2)}$, requires the values of the grid-points at $u_{i-2,}^{(p+1/2)}$ and $u_{i-1}^{(p+1/2)}$ (Fig. 1) and the computation of the unknown $u_{m+1-i}^{(p+1)}$ requires the values of $u_{m+2-i}^{(p+1)}$ and $u_{m+3-i}^{(p+1)}$ (Fig. 2). The unknown grid-points can only be determined after the values of their two previous neighbors at their respective current iteration levels have been calculated. In other words, all values at the $(p+1/2)$th level cannot be calculated independently and simultaneously, so as values at the $(p+1)$th level. These situations show that the IADEMF4 is not inherently parallel. Thus, to handle this problem, this study resorts to undertake a domain decomposition approach that firstly divides the physical domain into a number of subdomains, each being assigned to a processor; and secondly exchanges appropriate data across the boundaries of the subdomains. The Red-Black (RB) ordering is the domain decomposition strategy that is considered in this study. The approach focuses on minimizing the problem of data dependencies and it is highly parallel.

A. The IADEMF4-RB

The RB ordering has shown its competitiveness in terms of speedup and efficiency, as has been proven in studies made by Evans [14] in solving the parallel SOR iterative methods; Brill et al. [15] in using the block GS-RB on the Hermite collocation discretization of partial differential equations in two spatial dimensions; and Alias [5] in parallelizing the IADEMF2. Darwis et al. [16] proved that the GS-RB algorithm is more accurate and converges faster than the GS algorithm. Yavneh [17] showed that the SOR-RB is more efficient and smoother than the sequential SOR method for solving two-dimensional Poisson equations.

This section parallelizes the IADEMF4 by using the RB ordering technique. The algorithm used will be referred to as the IADEMF4-RB.

The strategy to develop the IADEMF4-RB algorithm begins by decomposing the domain Ω into two different independent subdomains, Ω^R and Ω^B. Each grid-point in the subdomains Ω^R and Ω^B is denoted red and black respectively. If i is even, the grid-point is marked red, and if i is odd, the grid-point is marked black. Assuming m is even, then, the computational formulae for the IADEMF4-RB are:

$$u_i^{(p+1/2)} = (1-\omega_y)u_i^{(p)} + \frac{\omega_y}{R}(E_{i-1}u_i^{(p)} + W_{i-1}u_{i+1}^{(p)} + V_{i-1}u_{i+2}^{(p)} - \hat{m}_{i-3}u_{i-2}^{(p+1/2)} - l_{i-2}u_{i-1}^{(p+1/2)} + f_i) \quad (18)$$

for $i = 2,4,..,m-2$ (red grid-points) and $i = 3,5,..,m-1$ (blackgrid-points)

$$u_i^{(p+1)} = (1-\omega_z)u_i^{(p+1/2)} + \frac{\omega_z}{Z_{i-1}}(S_{i-3}u_{i-2}^{(p+1/2)} + Q_{i-2}u_{i-1}^{(p+1/2)} + Pu_i^{(p+1/2)} - \hat{u}_{i-1}u_{i+1}^{(p+1)} - \hat{v}_{i-1}u_{i+2}^{(p+1)} + gf_i) \quad (19)$$

for $i = 2,4,..,m-2$ (red grid-points) and $i = 3,5,..,m-1$ (blackgrid-points)

The purpose of including the relaxation factors ω_y and ω_z in (18) and (19) is to accelerate the convergence rate of the scheme.

The IADEMF4-RB ordering, on say, three processors, P_1, P_2 and P_3, is illustrated in Fig. 3. P_1 and P_3 holds boundary values at $i = 0$ and $i = m+1$, respectively. The fourth-order methods require additional boundary values which are at positions $i = 1$ (a grid-point in P_1) and $i = m$ (a grid-point in P_3). As a strategy to obtain good load balancing, similar numbers of alternate red (R) and black (B) grid-points are assigned to each processor [18]. Depending on the color of the grid-point, the first two starting grid-points in a processor may be labelled as 'st_R' and followed by 'st_B', and the last two end grid-points may be labelled as 'en_B' followed by 'en_R'.

The following describes the implementation of the IADEMF4-RB based on Fig. 3. The algorithm is subjected to the given initial and boundary conditions. Before the beginning of the execution, the unknowns, $u_i^{(p+1/2)}$, for $i = 2,3,..,m-1$, are given 'guessed' values at the initial time. Then, the execution of the IADEMF4-RB algorithm is performed in two phases:

The first phase involves the computations of only the red grid-points at the iteration levels $(p+1/2)$ and $(p+1)$. This phase requires every processor to compute in parallel the red unknowns by making use of the initialized 'guessed' values. Example, the computation of $u_{st_R}^{(p+1/2)}$ in P_2 requires 'guessed' $u_{en_R}^{(p+1/2)}$ value from P_1 and $u_{st_B}^{(p+1/2)}$ value from P_2 itself, while the computation of $u_{en_R}^{(p+1)}$ in P_2 requires 'guessed' $u_{st_R}^{(p+1)}$ value from P_3 and $u_{en_B}^{(p+1)}$ value from P_2 itself.

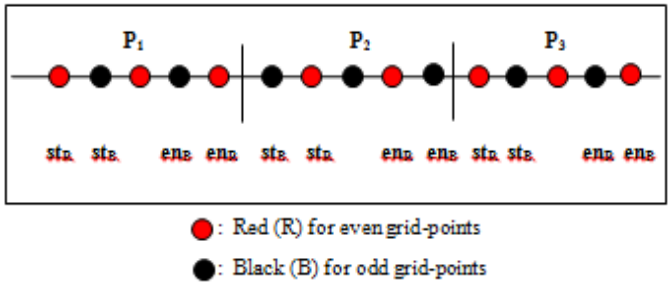

Fig. 3. One-Dimensional IADEMF4-RB Ordering

After the computations of the red grid-points for the two iteration levels have been completed, adjacent processors exchange their updated red values at the boundary grid-points to prepare for the calculation of the black grid-points in the second phase. Example,

Send updated $u_{\text{en}_\text{R}}^{(p+1/2)}$: from P_1 to P_2, and from P_2 to P_3.

Send updated $u_{\text{st}_\text{R}}^{(p+1)}$: from P_2 to P_1, and from P_3 to P_2.

The second phase continues by computing simultaneously the black unknowns at levels $(p+1/2)$ and $(p+1)$, using the most recent red values computed in the first phase. For example, the computation of $u_{\text{st}_\text{B}}^{(p+1/2)}$ in P_2 uses the updated red values $u_{\text{en}_\text{R}}^{(p+1/2)}$ and the 'guessed' black value $u_{\text{en}_\text{B}}^{(p+1/2)}$ from P_1, while the computation of $u_{\text{en}_\text{B}}^{(p+1)}$ in P_2 requires the updated red values $u_{\text{st}_\text{R}}^{(p+1)}$ from P_3 and the 'guessed' $u_{\text{st}_\text{B}}^{(p+1)}$ value from P_3. The updated black grid-points at the boundaries are then shared between adjacent processors. Example,

Send updated $u_{\text{en}_\text{B}}^{(p+1/2)}$: from P_1 to P_2, and from P_2 to P_3

Send updated $u_{\text{st}_\text{B}}^{(p+1)}$: from P_2 to P_1, and from P_3 to P_2.

The two phases are repeated until convergence is reached. Due to the dependencies on the updated values between adjacent processors, the IADEMF4-RB algorithm involves statements that take care of the communication between the processors. An example of a procedure for sending and receiving messages between processors in a PVM environment is as illustrated in Fig. 4. The IADEMF4-RB algorithm implemented by a slave processor can be described as in Fig. 5.

```
if (left!=0) /* If there is a processor on the left*/
    pvm_initsend( PvmDataDefault );
    pvm_pkdouble( &u [start], 1,1);
    pvm_send(left,50 );
end-if

if (right!= 0) /* If there is a processor on the right*/
    pvm_recv(right,50);
    pvm_upkdouble(&u [end+1],1, 1 );
    pvm_initsend( PvmDataDefault );
    pvm_pkdouble(&u [end], 1,1);
    pvm_send(right,60 );
 end-if
if (left!=0) /* If there is a processor on the left*/
    pvm_recv(left,60);
    pvm_upkdouble(&u [start-1],1, 1 );
end-if
```

Fig. 4. Communication Procedures for Sending and Receiving Messages between Adjacent Processors.

IADEMF4 –RB: Slave's Parallel Algorithm
begin
slaves receive data from master: tid , m, r, Δt, Δx, λ , ω
for $\forall i \in \Omega$
determine initial conditions $u_i^{(p)}$
initialize guessed values $u_i^{(p+1/2)}$
end-for
while (time level < T)
for $t = k$ and $t = k+1$
determine boundary conditions at u_0 , u_1 , u_m and u_{m+1}
end-for
for $\forall i \in \Omega^R$
compute f_i (refer to (7))
end-for
for $\forall i \in \Omega^B$
compute f_i (refer to (7))
end-for
set iteration $= 0$
while (convergence conditions are not satisfied)
for $\forall i \in \Omega^R$
compute $u_i^{(p+1/2)}$ (refer to (18))
end-for
for $\forall i \in \Omega^R$
compute $u_i^{(p+1)}$ (refer to (19))
end-for
send and receive updated red boundary values between adjacent slave processors (Fig. 4)
for $\forall i \in \Omega^B$
compute $u_i^{(p+1/2)}$ (refer to (18))
end-for
for $\forall i \in \Omega^B$
compute $u_i^{(p+1)}$ (refer to (19))
end-for
send and receive updated black boundary values between adjacent slave processors (Fig. 4)
test for convergence:
compute $e_i \leftarrow \left| u_i^{(p+1)} - u_i^{(p)} \right|$ for $\forall i \in \Omega^R$ and $\forall i \in \Omega^B$
if max $\left| e_i \right| < \varepsilon$
then $u_i^{(p)} \leftarrow u_i^{(p+1)}$
add 1 to iteration (if necessary)
end-while
end-while
Determine numerical errors for $\forall i \in \Omega^R$ and $\forall i \in \Omega^B$
slave sends data analysis to master
pvm_exit;

Fig. 5. IADEMF4-RB–Slave's Parallel Algorithm.

B. Parallel Algorithms for Benchmarking

The IADEMF2, the GS4 and the SOR4 algorithms [9] can also be parallelized using the RB ordering technique. They will serve as the benchmarks for the parallel IADEMF4-RB. The following are the schemes under consideration, assuming m is even.

1) the IADEMF2-RB algorithms:

$$u_i^{(p+1/2)} = (1-\omega_y)u_i^{(p)} + \frac{\omega_y}{d}(-l_{i-1}u_{i-1}^{(p+1/2)} + s_i u_i^{(p)} + w_i u_{i+1}^{(p)} + f_i) \quad (20)$$

for $i = 2,4,...,m$ (red grid-points) and $i = 1,3,5,...,m-1$ (black grid-points)

$$u_{m+1-i}^{(p+1)} = (1-\omega_z)u_i^{(p+1/2)} + \frac{\omega_z}{d_{m+1-i}}(v_{m-i}u_{m-i}^{(p+1/2)} + su_{m+1-i}^{(p+1/2)} + gf_{m+1-i} - \hat{u}_{m+1-i}u_{m+2-i}^{(p+1)}) \quad (21)$$

for $i = 2,4,...,m$ (red grid-points) and $i = 1,3,5,...,m-1$ (black grid-points)

2) the SOR4-RB algorithm (reduces to the GS4-RB algorithm when $\omega = 1$:

$$u_i^{(p+1)} = (1-\omega)u_i^{(p)} + \frac{\omega}{c}(f_i - au_{i-2}^{(p+1)} - bu_{i-1}^{(p+1)} - du_{i+1}^{(p)} - eu_{i+2}^{(p)}) \quad (22)$$

for $i = 2,4,...,m-2$ (red grid-points) and $i = 3,5,...,m-1$ (black grid-points)

IV. Computational Complexity

The computational complexity of the RB algorithms of interest is as given in Table I. It gives the number of parallel arithmetic operations that is required to evaluate the algorithms.

TABLE. I. Parallel Arithmetic Operations (m = Problem Size, n = Number of Iterations, P = Number of Processors)

Method	Number of additions	Number of multiplications	Total operation count
IADEMF4-RB	$10(m\text{-}2)n / P$	$13(m\text{-}2)n / P$	$23(m\text{-}2)n / P$
IADEMF2-RB	$6mn / P$	$9mn / P$	$15mn / P$
GS4-RB	$4(m-2)n / P$	$5(m-2)n / P$	$9(m-2)n / P$
SOR4-RB	$5(m-2)n / P$	$7(m-2)n / P$	$12(m-2)n / P$

V. Numerical Experiment

The IADEMF4-RB was implemented and tested on multiprocessor distributed memory architecture comprising of twelve interconnected processors with Linux operating system using the PVM communication library. In distributed memory, each processor has its own address space or local memory which is inaccessible to other processors. The processors operate independently in parallel, and they share their data by means of some form of inter-processor communication via an inter-connection network. The programmer is responsible for the details associated with message passing between processors. From the memory perspective, the size of memory increases in proportion to the increasing number of processors.

The parallel performances of the proposed algorithm was examined by solving a very large problem size on the experiment in (23), where m varied from 70,000 to 700,000. This problem was taken from Saul'yev (1964),

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2}, \quad 0 \le x \le 1 \quad (23)$$

subject to the initial condition $U(x,0) = 4x(1-x)$, $0 \le x \le 1$ and the boundary conditions $U(0,t) = U(1,t) = 0, \quad t \ge 0.$

The exact solution to the given problem is given by

$$U(x,t) = \frac{32}{\pi^3}\sum_{k=1,(2)}^{\infty}\frac{1}{k^3}\ e^{-\pi^2 k^2 t}\sin(k\pi x) \quad (24)$$

The other parameters considered for the experiment were λ = 0.5, Δt = 1.0204 x 10^{-12}, t = 5.1020 x 10^{-11}, and a stringent tolerance value of $\varepsilon = 10^{-15}$. The initial and Dirichlet boundary conditions at $i = 0$ and $i = m+1$ were applied based on the values given in the problem. For the fourth-order methods, the boundary values at positions $i = 1$ and $i = m$ were taken from the given exact solutions (24). The optimum values for r and the relaxation factors (ω_x , ω_y and ω) were determined by experiments.

VI. Results and Discussion

Table II compares the accuracy of the tested parallelized RB algorithms for a fixed problem size, $m = 700,000$. It is obvious that the IADEMF4-RB outperforms the IADEMF2-RB in terms of rate of convergence. The average absolute error, root mean square error and the maximum error of both algorithms seem identical up to four decimal places, due to the stringent tolerance value set in the experiment. The high computational complexity of the IADEMF4-RB is compensated by the high accuracy it achieves at every iteration and time level, causing its convergence to accelerate. The SOR4-RB speeds up the convergence of the GS4-RB, but they are both relatively not reliable in terms of accuracy.

Table III displays the number of iterations (n), execution time, speedup and efficiency of the IADEMF4-RB on using three different values of problem size, m. The execution time refers to the amount of time required to complete a parallel program on a number of P processors from the moment the execution starts till the moment the last processor finishes its execution [19]. Speedup expresses how much faster the parallel program executes relative to the sequential one. Amdahl's law states that there exists a bound on the speedup for a given problem with a fixed size [20], since some parts of the computations for solving a given problem are not parallelizable. Efficiency is a measure of the speedup achieved per processor. It estimates how well the processors are utilized during the execution of a parallel algorithm.

TABLE. II. PARALLEL RB ALGORITHMS – ERRORS AND NUMBER OF ITERATIONS

Method (m=700,000)	**Average absolute error**	**Root mean square error**	**Max. error**	**Number of iterations**
IADEMF4-RB $(r=0.8, \omega_y=1, \omega_z=1.1)$	1.5920e-09	7.3054e-09	1.9845e-07	288
IADEMF2-RB $(r=0.8, \omega_y=1, \omega_z=1.1)$	1.5920e-09	7.3054e-09	1.9845e-07	450
SOR4-RB (ω=1.06)	1.6150e-09	9.6395e-09	2.7422e-06	738
GS4-RB	1.6150e-09	9.6395e-09	2.7422e-06	794

$\lambda = 0.5$, $\Delta x = 2.60 \times 10^{-6}$, $\Delta t = 1.02 \times 10^{-12}$, $t = 5.10 \times 10^{-11}$, $\varepsilon = 1 \times 10^{-15}$

TABLE. III. IADEMF4-RB – PERFORMANCES USING SEVERAL VALUES OF m

m	**Δx**	P	**Execution time (s)**	**Speedup**	**Efficiency**
70,000 $n=359$	1.43×10^{-5}	1	4.869491	1	1
		2	2.507665	1.941843	0.970921
		4	1.518787	3.206171	0.801542
		6	1.261464	3.860190	0.643365
		8	1.102297	4.417585	0.552198
		10	1.039360	4.685086	0.468508
		12	1.008263	4.829584	0.402465
385,000 $n=312$	2.60×10^{-6}	1	20.039541	1	1
		2	10.062964	1.991415	0.995707
		4	5.300258	3.780842	0.945210
		6	3.828272	5.234617	0.872436
		8	2.993626	6.694069	0.836758
		10	2.447466	8.187873	0.818787
		12	2.101530	9.535691	0.794640
700,000 $n=288$	1.43×10^{-6}	1	35.682042	1	1
		2	17.896741	1.993773	0.996886
		4	8.962541	3.981241	0.995310
		6	6.202509	5.752840	0.958806
		8	4.900683	7.281034	0.910129
		10	3.992991	8.936168	0.893616
		12	3.456841	10.32215	0.860179

$\lambda = 0.5$, $\Delta t = 1.02 \times 10^{-12}$, $t = 5.10 \times 10^{-11}$, $\varepsilon = 1 \times 10^{-15}$

The results in Table III show that the execution time for a problem using any of the considered sizes is reduced and the speedup improves as the number of processors increases. For $m = 70,000$, the increase in speedup from $P = 1$ to $P = 12$ is about 80% and for $m = 700,000$, the increase is about 90%. This shows that parallel computation improves performance in terms of execution time and speedup over serial computation. Due to overheads, the overall efficiency for any m tends to decrease as the number of processors increases. Overheads have impacts on parallel performance. The two common types of overheads are the communication time and the idle time. The communication time is the time spent on communication and exchanging of data during the execution in all processors and the idle time is the time when processors stay idle, waiting for busy processors to send messages. Idling may be due to load imbalances amongst processors, or a bottleneck at the master processor when it has to interact with other worker processors [21].

For every number of processor ran in the experiment, the execution time for a problem size of 70,000 is comparatively smaller than a problem ten times its size. This is expected since fewer grid-points involve less mathematical operations and data sharing. The table, however, shows an improvement in convergence rate, speedup and efficiency as the size increases to $700,000$. The smaller size with higher number of iterations (n) seems to be less efficient due to the additional overhead imposed by having communications routed through the PVM daemon.

Fig. 6 shows that the execution time taken by every tested algorithm (listed in Table II) decreases with increasing P. However, the IADEMF4-RB executes in the least amount of time for every P. Despite the IADEMF4's greater computational complexity, its parallelization using the RB technique and the use of relaxation parameters have enabled it to execute in a shorter time on one and more processors in comparison to its counterpart of second-order.

Fig. 7 shows that every tested algorithm has a speedup of less than P, which implies that the parallel code is bounded by the sequential code (Amdahl's law). The parallel code runs slower due to overheads that outweigh the benefits of parallel computation. Amongst the four algorithms, the IADEMF4-RB proves to continue giving the best speedup as P increases. At $P = 12$, the speedup of the IADEMF4-RB is almost 14% closer to the linear speedup. As for the IADEMF2-RB, the SOR4-RB and the GS4-RB, there is an 18, 24 and 28 percent difference, respectively, between the method's speedup and the linear speedup.

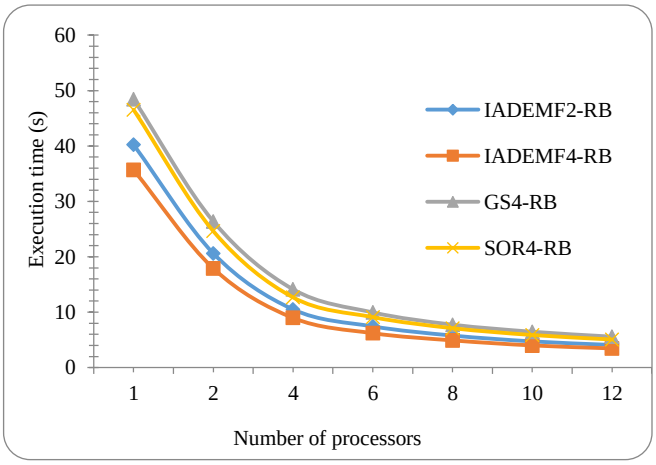

Fig. 6. Execution Time Versus Number of Processors.

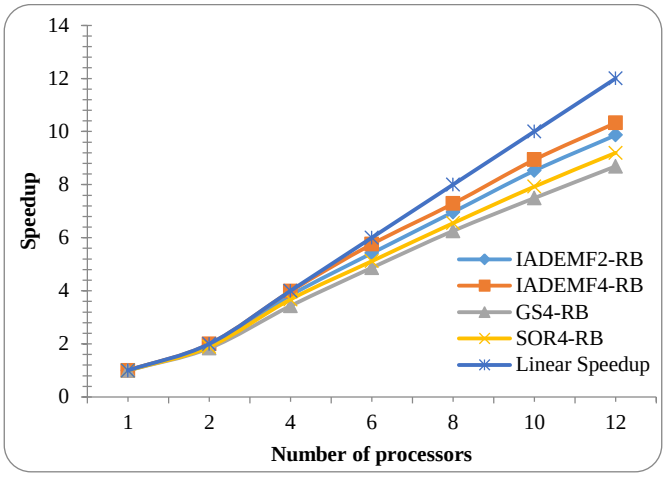

Fig. 7. Speedup Versus Number of Processors.

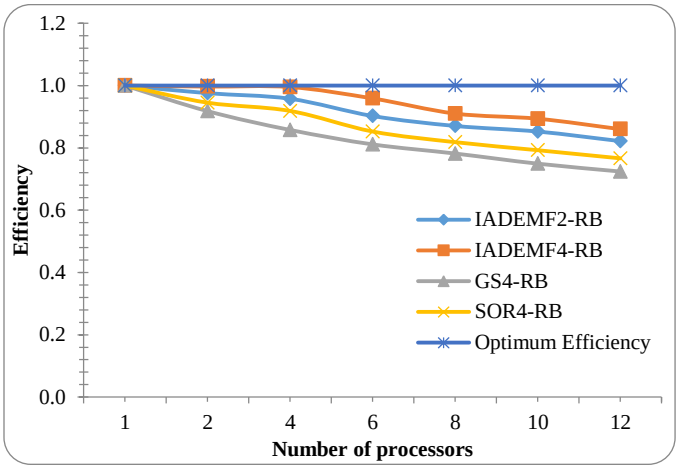

Fig. 8. Efficiency Versus Number of Processors.

Fig. 8 illustrates the reduction in efficiency as the number of processors increases. The overhead increases as P increases, leading to a declining performance in efficiency. The IADEMF4-RB, for example, performs efficiently for $P \leq 4$ and becomes less efficient for $P > 4$. The superior speedup performance by the IADEMF4-RB (Fig. 7), however, makes it the most efficient algorithm amongst the tested algorithms. With the number of processors equals to 12, the IADEMF4-RB achieves a speedup of 10.32 that equates to a higher efficiency of about 0.86 (Table III).

Temporal performance is a metric which is inversely proportional to the execution time. If there are several parallel algorithms solving the same problem with the same problem size implemented on the same number of processors, then the algorithm with the largest value for temporal performance will be considered as the best algorithm that can perform in the least amount of execution time. Fig. 9 shows that the IADEMF4-RB has proven itself as the algorithm with the best temporal performance amongst all the methods considered for comparison.

Granularity is an important performance metric since it gives a good indication of the feasibility of parallelization. It gives a qualitative measure of the ratio of the amount of computational time to the amount of communication time within a parallel algorithm [19]. The results of the granularity for the different tested parallel-RB methods are summarized in Table IV. Clearly, the granularity of all the methods decreases with increasing number of processors. This is due to the dependency of granularity on computational time and communication time. For any $P \leq 12$, the IADEMF4-RB has the largest granularity, indicating that the application spends more time in computation relative to communication. The large granularity of the IADEMF4-RB gives a good indication of the feasibility of its parallelization. The GS4-RB has the least granularity due to the idle time incurred by message latency, improper load balancing and time spent waiting for all processors to complete the process.

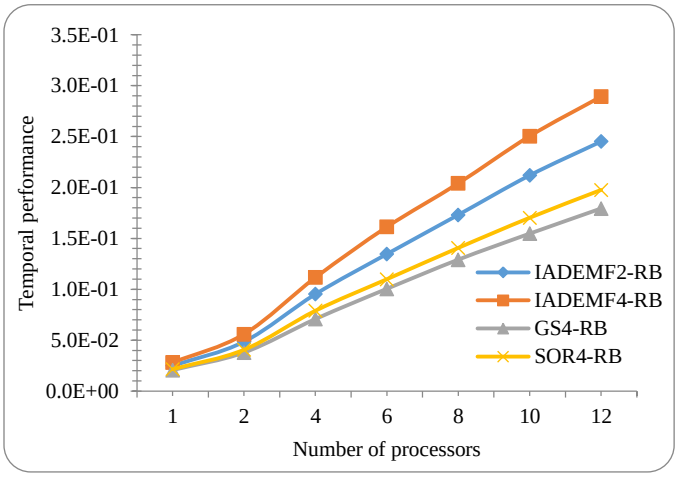

Fig. 9. Temporal Performance Versus Number of Processors.

TABLE. IV. SUMMARY OF THE GRANULARITY RESULTS FOR THE TESTED RB METHODS

P	IADEMF4-RB	IADEMF2-RB	SOR4-RB	GS4-RB
2	16.8	15.2	10.6	8.6
4	16.4	11.7	7.9	5.1
6	9.9	6.6	4.7	3.8
8	6.2	5.1	3.8	3.2
10	5.5	4.5	3.3	2.7
12	4.4	3.8	2.9	2.4

VII. CONCLUSION

This study strategizes to accelerate the convergence rate and the sequential execution time of the IADEMF4 by implementing it on a distributed computing based on PVM. The approach to parallelize the IADEMF4 is by implementing the RB parallel strategy.

The proposed IADEMF4-RB parallel algorithm significantly outperforms its counterparts of the second-order, as well as the benchmarked fourth-order classical methods. This is with regards to accuracy, convergence rate and parallel measures such as execution time, speedup, efficiency, temporal performance and granularity. Despite its higher computational complexity, its increasing number of correct digits at each iteration yields faster rate of convergence with higher level of accuracy for a large size matrix. The relatively coarse granularity delivered by the RB parallel implementation indicates the feasibility of parallelizing the proposed IADEMF4.

The efficient performance in parallel gives benefits, especially in solving problems involving larger sparse linear systems of equations that usually consumes huge amount of serial time. Future work is to consider applying the IADEMF4-RB in time-dependent PDEs that require higher-order accuracy with significant speedup and efficiency. Another possibility is to apply the proposed parallel method onto shared or hybrid memory architectures to reduce the problem of communication issues.

REFERENCES

[1] M. S. Sahimi, A. Ahmad, and A. A. Bakar, "The Iterative Alternating Decomposition Explicit (IADE) method to solve the heat conduction equation," International Journal of Computer Mathematics, vol. 47, pp. 219-229, 1993.

[2] D. J. Evans and M. S. Sahimi, "The Alternating Group Explicit Iterative Method to solve parabolic and hyperbolic partial differential equations," Ann. Rev. of Num. Fluid Mechanics and Heat Transfer, vol. 2, pp. 283-389, 1989.

[3] M. S. Sahimi, E. Sundararajan, M. Subramaniam, and N. A. A. Hamid, "The D'Yakonov fully explicit variant of the iterative decomposition method," Comp. Math. , vol. 42, pp. 1485-1496, 2001.

[4] M. S. Sahimi, N. A. Mansor, N. M. Nor, N. M. Nusi, and N. Alias, "A high accuracy variant of the Iterative Alternating Decomposition Explicit method for solving the heat equation," Int. J. Simulation and Process Modelling, vol. 2, Nos. 1/2, pp. 77-86, 2006.

[5] N. Alias, "Development and implementation of parallel algorithms in the IADE and AGE class of methods to solve parabolic equations on a distributed parallel computer systems," PhD Thesis, Universiti Kebangsaan Malaysia (2003).

[6] N. Alias and S. Kireev, "Fragmentation of IADE method using LuNA system," Malyshkin V. (eds) Parallel Computing Technologies, Lecture Notes in Computer Science, vol. 10421. Springer, Cham, 2017.

[7] J. Sulaiman, M. K. Hasan, and M. Othman, "The half-sweep Iterative Alternating Decomposition Explicit Method (HSIADE) for diffusion equations," Lecture Notes on Computer Science, vol. 3314, Berlin-Heidelberg, pp. 57-63, 2004.

[8] J. Sulaiman, M. K. Hasan, and M. Othman, "Quarter-sweep Iterative Alternating Decomposition Explicit algorithm applied to diffusion equations," International Journal of Computer Mathematics, vol. 81(12), pp. 1559-1565, 2004.

[9] N. Alias, M. S. Sahimi, and A. R. Abdullah, "Parallel strategies for the Iterative Alternating Decomposition Explicit Interpolation-conjugate Gradient method in solving heat conductor equation on a distributed parallel computer systems," Proceedings Third International Conference Numerical Analysis Eng., pp. 31-38, 2003.

[10] R. H. Shariffudin and S. U. Ewedafe, "Parallel domain decomposition for 1-D active thermal control problem with PVM," International Journal of Advanced Computer Science and Applications, vol. 6, No. 10, 2015.

[11] N. A. Mansor, A. K. Zulkifle, N. Alias, M. K. Hasan, and M. J. N. Boyce, "The higher accuracy fourth-order IADE algorithm," Journal of Applied Mathematics, vol. 2013 Article ID 236548, http://dx.doi.org/10.1155/2013/236548, 2013.

[12] G. D. Smith, "Numerical solution of partial differential equations: Finite difference methods," second ed., Oxford University Press, 1978.

[13] A. R. Mitchell and G. Fairweather, "Improved forms of the alternating direction methods of Douglas,Peaceman,and Rachford for solving parabolic and elliptic equations," Numerische Mathematik, vol. 6 (1), pp. 285–292, 1964.

[14] D. J. Evans, "Parallel S.O.R iterative methods," Parallel Computing, vol. 1, pp. 3-18, 1984.

[15] S. H. Brill and G. F. Pinder, "Parallel implementation of the Bi-CGSTAB method with Block Red-Black Gauss-Seidel preconditioner applied to the Hermite Collocation discretization of partial differential equations," Parallel Computing, vol. 28:3, pp. 399-414, 2002.

[16] R. Darwis, N. Alias, N. Yaacob, M. Othman, N. Abdullah, and T. Y. Ying, "Temperature behavior visualization on rubber material involving phase change simulation," Journal of Fundamental Sciences, vol. 5, pp. 55-62, 2009.

[17] I. R. Yavneh, "On Red-Black SOR smoothing in multigrid," SIAM J. Sci. Comput. , vol. 17(1), pp. 180-192, 1995.

[18] B. Körfgen and I. Gutheil, "Parallel linear algebra methods, computational nanoscience: do it yourself!," John von Neumann Institute for Computing. Jülich, NIC Series, vol. 31, pp. 507-522, 2006.

[19] J. Kwiatkowski, "Evaluation of parallel programs by measurement of its granularity," Proceeding PPAM '01 International Conference on Parallel Processing and Applied Mathematics–Revised Papers, Springer-Verlag, London, 2002.

[20] G. M. Amdahl, "Validity of the single-processor approach to achieving large scale computing capabilities," AFIPS Conference Proceedings, vol. 30. AFIPS Press, Reston, Va., pp. 483-485, 1967.

[21] J. Lemeire, "Leaning causal models of multivariate systems and the value of it for the performance modeling of computer programs," PhD Thesis, Vrije Univesiteit, Brussel, Brussels University Press, 2007.