Parallel Solution of Initial Value Problems

B. Neta
Naval Postgraduate School
Department of Mathematics
Code MA/Nd
Monterey, CA 93943

Keywords: Initial Value Problems, Boundary Value Problems, Parallel Processing, Hypercube, Box Scheme, Recursive Doubling, Extrapolation, Euler's Scheme, Gragg's Method, Method of Lines.

MOS Subject Classification: 65L, 65W

Abstract

This talk will be divided to three parts. In the first part we discuss the solution of linear ordinary differential systems of equations. Initial value as well as boundary value problems will be considered. The second part will cover nonlinear ordinary differential systems. In this case the algorithm is based on extrapolation and as such is efficient only on small parallel computers. We close with the method of lines and its use to solve partial differential equations on parallel computers.

1 Introduction

In the paper we discuss the parallel solution of ordinary initial and boundary value problems and the use of method of lines to solve partial differential equations on parallel computers. We open with a brief survey of parallel computing. The algorithm for linear ordinary differential equations will be discussed in section 3. The next section will detail the parallel solution of nonlinear ordinary differential equations. We close with a section on the method of lines and its use to solve partial differential equations on parallel computers.

2 Parallel Computing

The complexity of scientific computing today demands faster computers. Greater detailed models require a substantial amount of computation. Faster computers are needed to provide the result of the computation in a timely manner. Two approaches are possible:

i. increase the speed of circuitry. This is bounded by speed of light and cost.

ii. parallel computers.

Parallel processing is an efficient form of information processing which emphasizes the exploitation of concurrent events in the computing process. Concurrency implies parallellism, simultaneity and pipelining. Parallel events may occur in multiple resources during the same time interval; simultaneous events may occur at the same time instant; and pipelining events may occur in overlapped time spans. These concurrent events are attainable in a computer system at various processing levels, (see Hwang (1984, p. 6))

i. program level (executing multiple programs by time sharing),

ii. task level,

iii. inter-instruction level,

iv. intra-instruction level.

2.1 Classification of Parallel Computers by Type

Parallelism can be achieved in several ways:

1. temporal parallelism - pipeline computer (Cray - 1) computations are divided into a number of stages or segments with the output of one being the input of the next.

2. spatial parallelism - array processor (CM). Use multiple synchronized processing elements.

3. Asynchronous parallelism - multiprocessor (Cm^*). These processors are capable of performing independent operations, but share resources such as memory.

4. Multicomputer (INTEL hypercube) - a refinement of a multiprocessor. The processors are as in 3 but have their own local memory. These offer an added degree of freedom in programming, however interaction between processors may require synchronization.

Remark: many array processors, multicomputers and multiprocessors employ pipeline processors.

2.2 Classification by Architecture

1. Single instruction, single data (SISD) - serial computers include pipeline.
2. SIMD - array processors.
3. MISD - No current computer.
4. MIMD - multi processors or multicomputers. See Flynn (1966). The INTEL iPSC is a MIMD machine.

2.3 Classification by Topology

By the interprocessor connections (which are the means of communication).

Mesh

Pyramid

Butterfly

Hypercube

See Quinn (1987, pp. 25-30)
The following figures show a variety of parallel computers.

2.4 Measures of Performance

Two common measures of performance are: speedup & efficiency.

Speedup is defined by

$$S_p = \frac{T_s}{T_p}$$

where

T_s - time to complete task on a serial computer

T_p - time to complete task on a parallel computer with p processors.

Parallel programs often contain additional operations to accomodate parallelism. Many suggest that T_s should be measured on the fastest serial computer. However, the variation in the technical specifications of both computers may cloud the issue whether parallel processing is beneficial. As in many other articles we take T_1 for T_s.

The other measure, E, efficiency, accounts for the relative cost of achieving a specific speedup.

$$E_p = \frac{S_p}{p}$$

where p, the number of processors, is also the theoretical speedup.

Many factors could limit the speedup and efficiency:

1. number of sequential operations that cannot be parallelized,
2. communication time,
3. idle time to get synchronization.

This is stated as Amdahl's (1967) law:

$$S_p \leq \frac{1}{f + \frac{1-f}{p}}$$

where f is a fraction of operations to be performed sequentially.

2.5 INTEL iPSC/2 Hypercube

The iPSC/2 is a MIMD multicomputer with a hypercube topology. It consists of a system resource manager (host) and processors (nodes). The host, a 386-based computer, provides the interface between the user and the nodes.

The nodes are self-contained INTEL 80386 microprocessors, each containing 80387 numeric coprocessor, its own local memory and a direct-connect communication module (DCM). Each computing node may be augmented by a vector extension module for pipelined vector operations.

Communications are completed through message passing. The DCM allows messages to be sent directly to the receiving node without disturbing the other node processors. The iPSC/2 uses a UNIX operating system and may be programmed in Fortran or C languages.

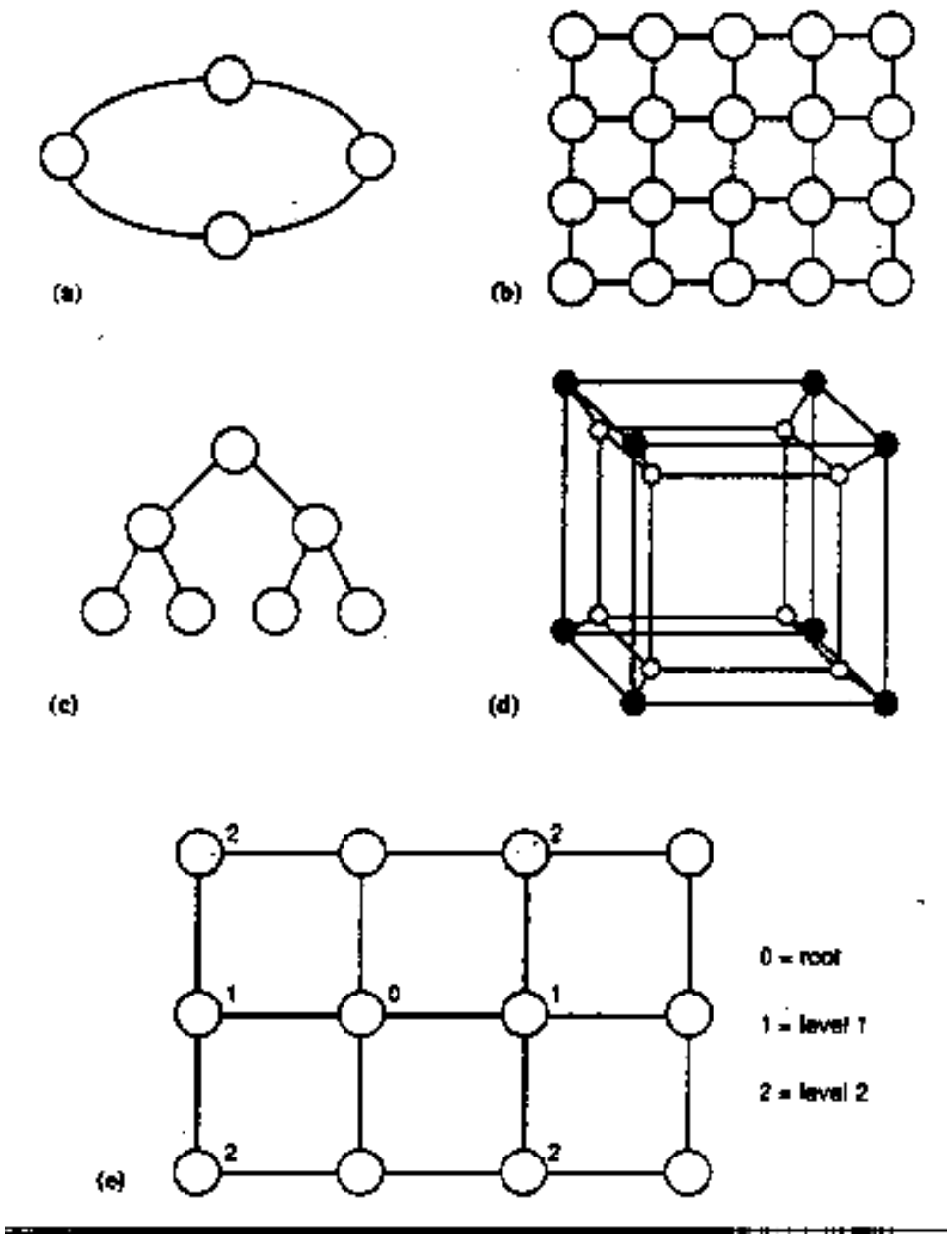

Figure 1: MIMD interconnection network topologies (a) ring; (b) mesh; (c) tree; (d) hypercube; (e) tree mapped to a reconfigurable mesh. See Duncan 1990

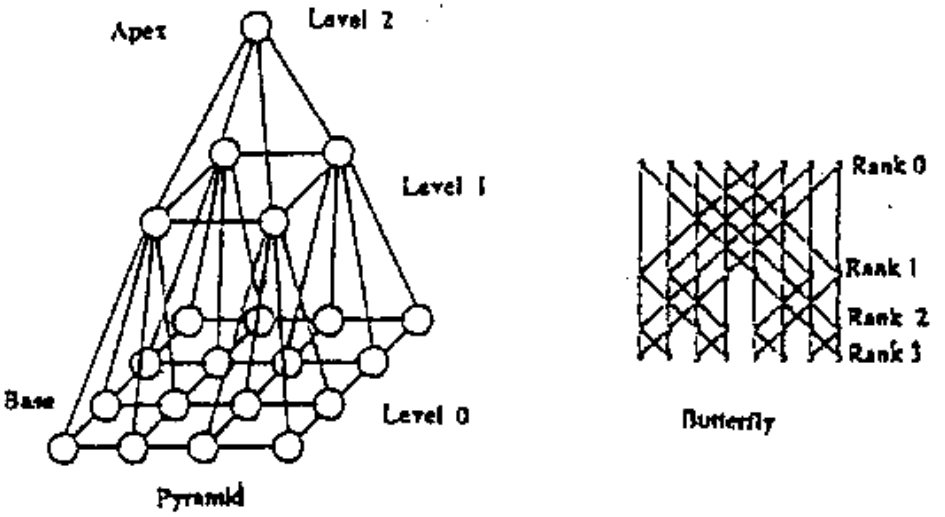

Figure 2: Pyramid on left and butterfly on the right. See Duncan 1990

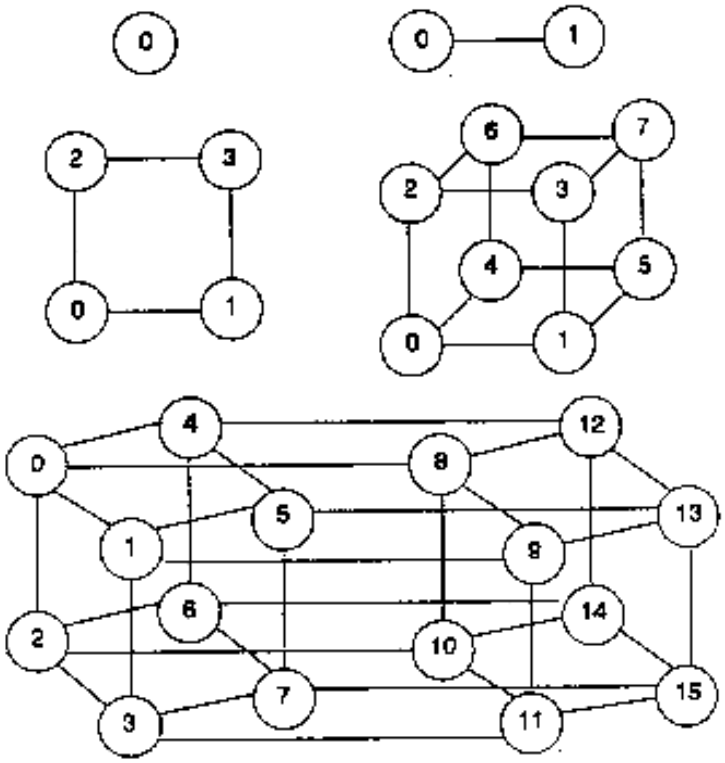

Figure 3: Hypercubes of dimension zero through four. See Duncan 1990

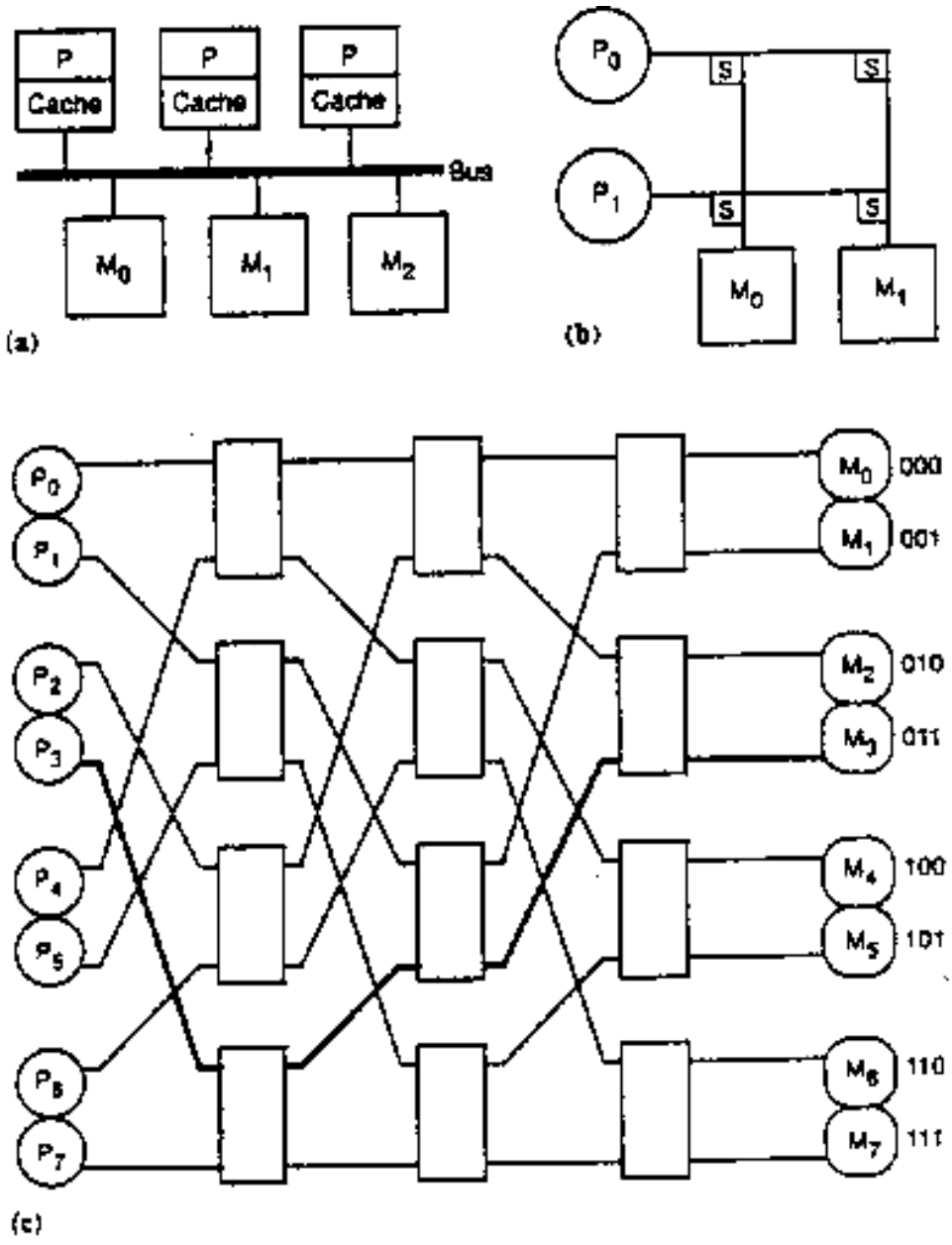

Figure 4: MIMD shared memory interconnection schemes: (a) bus interconnection; (b) 2x2 crossbar; (c) 8x8 omega MIN routing a P3– request to M_3. See Duncan 1990

2.6 Methods of Parallelization

We mention here two methods of parallelization.

1. Vectorizatoin

The process of converting blocks of sequential operations into vector instructions that may be pipelined.

2. Distributing Computations

In order to partition a program into parallel tasks to distribute among the processors of a multicomputer, a different strategy is needed. The strategy depends on the parallel computer used. Two decomposition strategies are suggested:

Control decomposition and domain decomposition.

i. Control Decomposition

This is the strategy of dividing tasks or processes among the nodes. This incorporates a divide and conquer approach.

One method of control decomposition is for the parallel program to self-schedule tasks. For this method one node assumes the role of a manager with the rest assuming roles of workers. The manager maintains a list of processes to be accomplished and assigns processes to the working nods. The working nodes request jobs, receive processes and perform the indicated tasks.

A second method is to pre-schedule the processes. The exact tasks required of each node are explicitly stated in the parallel program. This method saves the cost of the manager, but the progammer has to ensure that processes are evenly distributed among the nodes (load balancing).

ii. Domain Decomposition

In this strategy the input data or domain is divided among the nodes. The major difference between control and domain decomposition is that domain decomposition strategy requires each node to perform essentially the same tasks but with different input data.

Domain decompoisiton is recommended if the calculations are based on a large data structure and the amount of work is the same for each node. An example is a multiplication of two large matrices by using block multiplication. The user should be aware of load balancing since not every input data requires the same amount of work.

2.7 Performance

The following three factors should be considered:

i. Load balance,
ii. Communication to computation ratio,
iii. Sequential bottlenecks.

Load balance refers to the degree to which all nodes are active. If the work is not evenly distributed among the processors, the parallel algorithm will show constrained speedup. Load balancing maybe achieved by reducing the grain size of the parallel tasks, self-scheduling tasks or redistributing the domain.

Communication time is inherent in parallel algorithms. A large communication to computation ratio constrains efficiency. Reduction of this ratio maybe achieved by increasing the grain size, grouping messages, or recalculating values instead of receiving them from another node.

Sometimes tasks cannot begin until completion of a previous task. A sequential bottleneck happens when other processors are waiting for another node to complete a task before they may continue. The portion of operations that are not completed in parallel restricts speedup as can be seen by Amdahl's law. The only method to remove such bottlenecks is to modify or reorder the algorithm in order to overlap sequential code with other computations (if possible).

3 Ordinary Initial Value Problems

The numerical solution of ordinary differential systems of equations is an intrinsically sequential procedure: given the data at a point x (or at several points $x, x-h, \cdots, x-mh$), one advances to the following point $x+h$.

In order ot parallelize this procedure, we first consider linear systems, i.e.

$$\underline{y}'(x) = A(x)\underline{y}(x) + \underline{f}(x) \quad a < x < b$$

where y and f are n dimensional vectors and A is a nxn matrix. The idea here is based on the following:

For homogeneous systems

$$\underline{y}'(x) = A(x)\underline{y}(x) \quad a < x < b \tag{1}$$

the solution at the right endpoint is a linear function of the values at the left endpoint, i.e.

$$\underline{y}(b) = Y_{[a,b]}\underline{y}(a)\,.$$

Here $Y_{[a,b]}$ is the value at $x = b$ of the fundamental solution matrix Y, which is defined as

$$\begin{aligned} Y' &= A(x)Y \\ Y(a) &= I \end{aligned}$$

where Y is nxn and I is the identity matrix.

Theorem:

Let

$$\underline{e}^{(i)} = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \longleftarrow i^{th}$$

Let $\underline{y}^{(i)}$ be solutions of

$$\underline{y}' = A\,\underline{y}$$

satisfying the initial conditions

$$\underline{y}^{(i)}(a) = \underline{e}^{(i)}$$

then $\underline{y}^{(i)}$ form a fundamental set of solutions of the system.

Remark:

$$Y_{[a,b]}Y_{[b,c]} = Y_{[a,c]} \qquad \text{for } a < b < c\,.$$

To solve the problem on the interval $[a,b]$, we propose to assign N contiguous subintervals

$$[a,x_1],\ [x_1,x_2],\cdots,\ [x_{N-1},b]$$

to the N processors, and let each of them compute in parallel, the corresponding fundamental solution. This is a task requiring a possibly large number of sequential steps, for the numerical evaluation of $Y[x_i,x_{i+1}]$.

One way of doing this is described here.

Let

$$\begin{aligned} x_j &= a + jh \qquad\qquad j = 0,1,2,\cdots,m \\ h &= \frac{b-a}{m} \end{aligned}$$

be a uniform mesh.

The box scheme (see e.g. Keller, 1976) applied to the system (1) yields

$$\underline{y}_{j+1} = \underline{y}_j + h\{A_{j+1/2}(\underline{y}_{j+1} + \underline{y}_j)/2 + \underline{f}_{j+1/2}\}\,,$$

where

$$\begin{aligned} A_{j+1/2} &= A(a + (j+1/2)h) \\ \underline{f}_{j+1/2} &= \underline{f}(a + (j+1/2)h) \end{aligned}$$

and $\underline{y}_j$ is the approximation to $\underline{y}(x_j)$. Let $\{j_i, i = 1,\cdots,s\}$ be a strictly increasing sequence such that $j_1 > 0$ and $j_s = m$. Let Φ_i be nxn matrices defined by

$$\Phi_i = \prod_{j=j_{i-1}}^{j_i-1}(I - \tfrac{h}{2}A_{j+1/2})^{-1}(I + \tfrac{h}{2}A_{j+1/2}), \quad i = 1,2,\cdots,s$$

where $j_0 = 0$ and h is sufficiently small so that $I - \frac{h}{2}A_{j+1/2}$ are nonsingular.

Similarly, let

$$\begin{aligned} \underline{\phi}_i = {} & (I - \tfrac{h}{2}A_{j_i-1/2})^{-1}(I + \tfrac{h}{2}A_{j_i-1/2})\underline{\tilde{y}}_{j_i-1-j_{i-1}} \\ & + h(I - \tfrac{h}{2}A_{j_i-1/2})^{-1}\underline{f}_{j_i-1/2},\ i = 1,2,\cdots,s \end{aligned}$$

where

$$\underline{\tilde{y}}_0 = 0$$

and

$$\underline{\tilde{y}}_{j+1} = (I - \frac{h}{2}A_{j+1/2+j_{i-1}})^{-1}\left[(I + \frac{h}{2}A_{j+1/2+j_{i-1}})\underline{\tilde{y}}_j + h\underline{f}_{j+1/2+j_{i-1}}\right]$$
$$j = 0, \cdots, j_i - j_{i-1} - 2.$$

Then it was shown by Keller and Nelson (1987)

$$\underline{y}_{j_i} = \Phi_i \underline{y}_{j_{i-1}} + \underline{\phi}_i \qquad i = 1, 2, \cdots, s$$

Notes:

1. The last factor in Φ_i is the matrix required for $\underline{\phi}_i$.
2. The vector $\underline{\tilde{y}}_{j_i - 1 - j_{i-1}}$ can be computed in the same loop Φ_i is computed since they require the same matrices.

As can be seen in the above description, this procedure may be extended to inhomogeneous equations, with initial data $\underline{y}(a) =$ given or two point boundary data

$$B_1\underline{y}(a) + B_2\underline{y}(b) = \text{ given}\,.$$

The algorithm for the initial value problem

Step 1. Using N processors to solve the linear inhomogeneous system with initial conditions:

$$\underline{y}'(x) = A(x)\underline{y}(x) + \underline{f}(x)$$
$$\underline{y}(x_{min}) = \underline{g}$$

divide the required interval into N subintervals:

$$[x_{\min}, x_1], [x_1, x_2], \cdots, [x_{N-1}, x_{\max}] .$$

The algorithm will produce numerical approximations for $y(x_j),\ j = 1, 2, \cdots, N$.

Step 2. Do in parallel:

Processor j, working on the interval $[x_{j-1}, x_j]$ solves numerically the following two systems:

$$Y_j' = A(x)Y_j$$
$$Y_j(x_{j-1}) = I \text{ the identity matrix}$$

$$\underline{\phi}_j'(x) = A(x)\underline{\phi}_j(x) + \underline{f}(x)$$
$$\underline{\phi}_j(x_{j-1}) = \underline{0}$$

The matrix Y_j is the fundamental solution on the subinterval, while $\underline{\phi}_j$ incorporates the inhomogeneous effect of the forcing function $\underline{f}$. When this step is completed, one may recursively compute $\underline{y}(x_j)$ from:

$$\begin{aligned}
&\underline{y}(x_1) = Y_1(x_1)\underline{g} && +\underline{\phi}_1(x_1)\\
&\underline{y}(x_2) = Y_2(x_2)\underline{y}(x_1) && +\underline{\phi}_2(x_2)\\
&\cdots\\
&\underline{y}(x_N) = Y_N(x_N)\underline{y}(x_{N-1}) && +\underline{\phi}_N(x_N)
\end{aligned}$$

Step 3. The last step of the algorithm is an efficient performance of the recursion above, assuming that $N = 2^m$. This is a generalization of the recursive doubling algorithm due to Stone (1973)

a) For $1 \le j \le N$, initialize:

$$\begin{aligned}
\underline{y}_j &= \underline{\phi}_j(x_j),\\
M_j &= Y_j(x_j)\\
\text{also :}\quad \underline{y}_1 &= \underline{g} + M_1\underline{y}_1,\\
k &= 1
\end{aligned}$$

b) For all $j > k$

$$\begin{aligned}
\underline{y}_j^* &= \underline{y}_j + M_j\underline{y}_{j-k}\\
M_j^* &= M_j M_{j-k}
\end{aligned}$$

c) For all $j > k$ replace $M, \underline{y}$ by $M^*, \underline{y}^*$

$$\begin{aligned} \underline{y}_j &= \underline{y}_j^*, \\ M_j &= M_j^* \end{aligned}$$

d) Set $k = 2k$. If $k < N$, repeat steps b-c. Otherwise the algorithm ends with $\underline{y}_j$ the numerical approximation to the solution at x_j.

3.1 Experimenting with IVPs

We have solved several systems of a variety of order. We will only give results for 3x3 and 10x10. The system is

$$\begin{aligned} y_i' &= y_i + x y_{i+1} + f_i \qquad 0 \le i \le n-1 \\ y_n' &= y_n + f_n \end{aligned}$$

where f_i are chosen so the solution is

$$\begin{aligned} y &= (1, e^x, e^{-x}, e^{2x}, e^{-2x}, e^{3x}, e^{-3x}, x, \sin x, \cos x) \qquad n = 10 \\ y &= (1, e^x, e^{-x}) \qquad n = 3 \end{aligned}$$

The number of steps per processor was varied from 10 to 80. The following table gives the ratio of communication to total time.

# of steps per processor	3x3 message type	3x3 send on demand	10x10 send on demand
10	64%	73%	14%
20	47%	60%	6%
40	31%	43%	4%
80	21%	26%	2%

Bianchini (1993) implemented this solver on T805 transpoter system with an 8 processor ring topology. Each processor computes 20 steps. The communication time is 11 msec, the computation time for the solution is 1158 msec and for the recursive doubling is 48 msec. Therefore 0.9% of time is spent on communication. We can estimate the total running time as follows: a single processor to solve a system nxn using s steps will require

$$ksn$$

(evaluating n right hand sides s times, ignoring matrix-vector multiplications). Using N processors, our algorithm requires

$$\frac{s}{N} n^2$$

because we compute the nxn fundamental matrix. Heuristically, there is gain in parallelism only if the order of the system is smaller than the number of processors. Even if there is **no** obvious gain, the algorithm may become efficient when used as the first step of an inverse problem, or distributed parameter problem.

3.2 Algorithm for BVPs

Step 1. Using N processors to solve

$$\begin{aligned} \underline{y}' &= A(x)\underline{y} + \underline{f}(x) \\ B_1\underline{y}(a) &+ B_2\underline{y}(b) = \underline{g} \end{aligned}$$

divide the interval into N subintervals.

Step 2. Do in parallel:

Processor j, working on the interval $[x_{j-1}, x_j]$ solves numerically

$$\begin{aligned} Y_j' &= A(x)Y_j \\ Y_j(x_{j-1}) &= I \\ \underline{\phi}_j' &= A(x)\underline{\phi}_j + \underline{f}(x) \\ \underline{\phi}_j(x_{j-1}) &= \underline{0} \end{aligned}$$

Step 3.

3a) For $1 \leq j \leq N$

$$\begin{aligned} \underline{y}_j &= \underline{\phi}_j(x_j), \\ M_j &= Y_j(x_j) \\ k &= 1 \end{aligned}$$

3b) For $j > k$

$$\begin{aligned} \underline{y}_j^* &= \underline{y}_j + M_j\underline{y}_{j-k} \\ M_j^* &= M_j M_{j-k} \end{aligned}$$

3c) For all $j > k$ replace $M, \underline{y}$ by $M^*, \underline{y}^*$

3d) set $k = 2k$. If $k < N$, repeat 3b-3c above.

Step 4. Processor N solves the following for η

4a) $B_1\underline{\eta} + B_2(M_N\underline{\eta} + \underline{y}_N) = \underline{g}$ then broadcasts $\underline{\eta}$ to all processors. This is the initial value consistent with the boundary condition.

4b) For all j compute in parallel

$$\underline{y}_j = \underline{y}_j + M_j\underline{\eta}$$

4 Nonlinear Systems

The idea here is to use extrapolation. One can solve the system using Euler's method or Gragg's method and then extrapolate to obtain a high accuracy solution.

4.1 Solver

Euler's method

$$y_{n+1} = y_n + hf(x_n, y_n)$$

with truncation

$$y_n - y(nh) = A_1 h + A_2 h^2 + A_3 h^3 + \cdots$$

Gragg's method

$$\begin{aligned}
&z_{1/2} = y_0 + \tfrac{h}{2} f(x_0, y_0) \\
&y_1 = y_0 + hf(x_{1/2}, z_{1/2}) \\
&\begin{cases} z_{n+1/2} = z_{n-1/2} + hf(x_n, y_n) & \\ & n = 1, 2, \cdots \\ y_{n+1} = y_n + hf(x_{n+1/2}, z_{n+1/2}) & \end{cases}
\end{aligned}$$

with truncation

$$y_n - y(nh) = B_2 h^2 + B_4 h^4 + \cdots$$

Each processor uses an ODE solver with

$$h_r = \frac{N}{r} H$$

The common points are $x_j = a + (j-1)NH$.

4.2 Extrapolation

Given

$$\{h_r, y(x_i, h_r) | r = 0, 1, \cdots, N-1; \quad i = 1, 2, \cdots, M\}$$

The solution at those M points in (a, b) is computed by the same scheme by all N possible $h's$).

Find a polynomial of degree $N-1$

$$\Pi_{N-1}(h_r) = y(x_i, h_r)$$

or a rational function

$$R_{\mu,\nu}(h_r) = y(x_i, h_r).$$

For polynomial extrapolation we construct a table of values T_{rs} as follows

$$\begin{aligned}
T_{r0} &= y(x_i, h_r) \\
T_{rs} &= T_{r+1s-1} + \frac{T_{r+1s-1} - T_{rs-1}}{(\frac{h_r}{h_{r+s}})^\gamma - 1}
\end{aligned}$$

$$s = 0, 1, \cdots, N-1$$
$$r = 0, 1, \cdots, N-s$$

$$\begin{aligned} \gamma &= 1 \text{ for Euler} \\ &= 2 \text{ for Gragg} \end{aligned}$$

For rational functions we construct a table of values as follows

$$\begin{aligned} T_{r-1} &= 0 \\ T_{r0} &= y(x_i, h_r) \\ T_{rs} &= T_{r+1\,s-1} + \frac{T_{r+1\,s-1} - T_{r\,s-1}}{(\frac{h_r}{h_{r+s}})^\gamma [1 - \frac{T_{r+1\,s-1} - T_{r\,s-1}}{T_{r+1\,s-1} - T_{r+1\,s-2}}] - 1} \end{aligned}$$

$$s = 1, 2, \cdots, N-1$$
$$r = 0, 1, \cdots, N-s-1$$

assume

$$\begin{aligned} \mu &= [\frac{N-1}{2}] \\ \nu &= N - 1 - \mu \end{aligned}$$

Extrapolation will yield $O(h^N), O(h^{2N})$ accuracy for Euler, Gragg schemes respectively.

In the following table we show which processor computes which part of the solution.

4.3 Experimenting with nonlinear systems

Example 1

$$\begin{aligned} &y' = y \sin x, \qquad 0 < x < 5, \\ &y(0) = e^{-1}. \end{aligned}$$

The exact solution is

$$y = e^{-\cos x}.$$

In the next table we summarize the results of the experiment with Euler method and extrapolation using N processors. It is clear that the results using polynomial extrapolation are much better. The accuracy ($||y_h - y_e||_0$, where y_h is the result of extrapolation) is increasing with the number of processors. The results using rational extrapolation are not as good and not improving after 5 processors.

To measure the order of the method, we have computed y_h and $y_{h/2}$ (the solution after extrapolation with step $h/2$ instead of h). The columns entitled 'error reduction' in each table report the error quotient 'coarse to fine'.

	processor	step	proc 1	proc 2	proc 3	proc 4	proc 5	proc 6	proc 7
T_{00}	1	$8H$							
			T_{01}						
T_{10}	2	$4H$		T_{02}					
			T_{11}		T_{03}				
T_{20}	3	$\frac{8}{3}H$		T_{12}		T_{04}			
			T_{21}		T_{13}		T_{05}		
T_{30}	4	$2H$		T_{22}		T_{14}		T_{06}	
			T_{31}		T_{23}		T_{15}		T_{07}
T_{40}	5	$\frac{8}{5}H$		T_{32}		T_{24}		T_{16}	
			T_{41}		T_{33}		T_{25}		
T_{50}	6	$\frac{4}{3}H$		T_{42}		T_{34}			
			T_{51}		T_{43}				
T_{60}	7	$\frac{8}{7}H$		T_{52}					
			T_{61}						
T_{70}	8	H							

Table 1: Extrapolation assigned to each processor

The accuracy using Gragg's Method in higher machine accuracy (double precision) has been reached with 5 processors if polynomial extrapolation is used.

The error quotient close to (4^p) until machine accuracy is reached.

Example 2

$$y'_j \quad = \quad \frac{j y_j y_{j+1}}{x^{j+2}} \qquad\qquad j = 1, 2, \cdots, n-1 \qquad 6 < x < 10$$

	POLYNOMIAL			RATIONAL		
Processor	coarse(1/4)	fine(1/8)	error reduction	coarse (1/4)	fine (1/8)	error reduction
2	1.55-02	4.54-03	3	5.15-03	1.26-03	4
3	1.17-03	1.78-04	7	3.52-04	4.03-05	9
4	6.71-05	5.23-06	13	9.04-04	4.23-05	21
5	3.05-06	1.22-07	25	1.11-04	2.11-06	53
6	1.19-07	2.42-09	49	6.38-06	1.05-06	6
7	3.92-09	4.06-11	97	1.26-05	5.67-05	0
8	1.14-10	5.01-13	228	2.11-04	5.29-05	4

Table 2: Euler scheme

	POLYNOMIAL			RATIONAL		
Processor	coarse(1/4)	fine(1/8)	error reduction	coarse (1/4)	fine (1/8)	error reduction
2	3.06-06	1.88-07	16	6.72-06	4.16-07	16
3	7.72-09	1.17-10	66	9.99-09	1.54-10	65
4	1.66-11	6.20-14	268	2.06-10	2.87-12	72
5	3.01-14	3.99-15	8	1.59-10	2.18-13	729
6	4.73-15	2.92-15	2	6.04-12	1.08-13	56
7	4.81-15	1.20-14	0	4.37-13	1.44-14	30
8	3.36-14	2.98-14	0	1.22-08	7.60-10	16

Table 3: Gragg method

$$\begin{aligned} y'_n &= \frac{n y_n y_1}{x^2} \\ y_j \quad (6) &= 6^j \end{aligned}$$

The exact solution is $y_j = x^j$.

	POLYNOMIAL			RATIONAL		
Processor	coarse(1/4)	fine(1/8)	error reduction	coarse (1/4)	fine (1/8)	error reduction
2	1.33-01	5.70-02	2	6.95-02	2.56-02	3
3	3.89-02	1.07-02	4	1.03-01	2.97-03	35
4	1.06-02	1.88-03	6	2.69-03	3.54-04	8
5	2.78-03	3.02-04	9	1.39-03	4.61-04	3
6	6.82-04	4.40-05	16	3.99-03	1.69-05	236
7	1.55-04	5.79-06	27	8.73-05	1.34-05	7
8	3.24-05	6.94-07	47	5.35-03	1.54-03	3

Table 4: Euler scheme

4.4 Speedup and Efficiency

Gragg's method for example 2 (n=4) and polynomial extrapolation

	POLYNOMIAL			RATIONAL		
Processor	coarse(1/4)	fine(1/8)	error reduction	coarse (1/4)	fine (1/8)	error reduction
2	7.51-03	7.18-04	10	5.00-03	4.79-04	10
3	4.56-04	1.27-05	36	1.39-04	1.12-06	124
4	2.06-05	1.59-07	130	8.22-06	1.40-07	59
5	6.89-07	1.45-09	475	7.29-07	1.25-08	58
6	1.80-08	1.03-11	1748	3.61-06	1.25-08	289
7	3.85-10	5.38-14	7156	8.62-09	1.60-10	54
8	7.10-12	2.55-14	278	1.15-05	7.35-07	16

Table 5: Gragg scheme

H	T(1) in msec	T(8) in msec	Speedup	Efficiency
.1	759	152	4.96	.62
.05	1509	247	6.08	.76

Table 6: Speedup and efficiency for example 2

5 Method of Lines

There is an extensive literature, primarily of Russian origin, for the method lines which has been summarized to 1965 by Liskovets (1965). See also Mikhlin and Smolitskiy (1967). The method is simple in concept - for a given system of partial differential equations discretize all but one of the independent variable (see Ames, 1977). This semidiscretization yields a coupled system of ordinary differential equations which are then numerically solved by a parallel scheme described above. For example, consider the nonlinear diffusion equation

$$u_t = [D(x,t,u)u_x]_x + f(x,t,u,u_x), \quad 0 < x < 1,\ 0 < t \le T$$

subject to the initial and boundary conditions

$$\begin{aligned} &u(x,0) = F(x), \quad 0 \le x \le 1, \\ &\alpha_1(t)u(0,t) + \beta_1(t)u_x(0,t) = \gamma_1(t), \quad 0 < t \le T, \\ &\alpha_2(t)u(1,t) + \beta_2(t)u_x(1,t) = \gamma_2(t), \quad 0 < t \le T, \end{aligned}$$

where D is a bounded function. The semidiscretization is given by the following system $i = 0$

$$\frac{du_0}{dt} = \begin{cases} 0 & \text{if } \beta_1 = 0 \\ \frac{2}{h}[D_{1/2}\frac{u_1 - u_0}{h} - D_0\frac{\gamma_1 - \alpha_1 u_0}{\beta_1}] + f(x_0, t, u_0, \frac{\gamma_1 - \alpha_1 u_0}{\beta_1}) & \text{if } \beta_1 \neq 0 \end{cases}$$

$u_0 = \frac{\gamma_1}{\alpha_1}$ if $\beta_1 = 0$

$1 \le i \le N-1$

$$\frac{du_i}{dt} = \frac{D_{i+1/2}(u_{i+1} - u_i) - D_{i-1/2}(u_i - u_{i-1})}{h^2} + f(x, t, u_i, \frac{u_{i+1} - u_{i-1}}{2h})$$

$i = N$

$$\frac{du_N}{dt} = \begin{cases} 0 & \text{if } \beta_2 = 0 \\ \frac{2}{h}[D_N \frac{\gamma_2 - \alpha_2 u_N}{\beta_2} - D_{N-1/2} \frac{u_N - u_{N-1}}{h}] + f(x_N, t, u_N, \frac{\gamma_2 - \alpha_2 u_N}{\beta_2}) & \text{if } \beta_2 \neq 0 \end{cases}$$

$u_N = \frac{\gamma_2}{\alpha_2}$ if $\beta_2 = 0$

The initial data

$$u_i(0) = F(x_i) \quad i = 0, 1, \cdots, N\,.$$

Chang and Madsen (1973) used the method of lines to solve a two dimensional chemical kinetics transport problem.

We conclude with a linear system

$$y' = Ay \quad 0 < x < 4$$

$$\underline{y}(0) = \begin{pmatrix} 1 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

where $A = (a_{ij})$ is a symmetric tridiagonal matrix whose elements are

$$a_{ii} = -2 \quad i = 1, 2, \ldots, N$$

$$a_{i,i+1} = a_{i+1,i} = 1, \quad i = 1, 2, \ldots, N-1.$$

This system resulting from approximating the one dimensional heat equation

$$u_t = u_{xx}.$$

The results are summarized in Table 7. The error quotient is much better than 2^p when using polynomial extrapolation. The accuracy, though, is not very high, and in linear systems it is cheaper and more accurate to use the idea developed by Lustman *et al* (1991).

	POLYNOMIAL			RATIONAL		
Processor	coarse(1/4)	fine(1/8)	error reduction	coarse (1/4)	fine (1/8)	error reduction
2	7.88-01	2.16-02	36	8.21-01	5.78-02	14
3	3.75-01	2.98-03	126	3.02-02	4.98-03	6
4	1.27-01	6.49-04	196	5.86-02	3.40-03	17
5	3.40-02	1.18-04	288	2.02-02	3.01-04	67
6	7.82-03	1.69-05	463	5.36-03	5.89-04	9
7	1.62-03	2.01-06	806	1.64-03	2.27-05	72
8	3.11-04	2.07-07	1502	5.06-03	1.10-03	5

Table 7: solution by method of lines

References

G. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, AFIPS Conf. Proc., **30** (1967), 483-485.

W. F. Ames, Numerical Methods for Partial Differential Equations, second edition, Academic Press, New York, 1973.

M. Bianchini, Implementation of parallel ODE software on T805 transputer, personal communication, 1993.

J. S. Chang and N. K. Madsen, Global transport and kinetics models, UCRL-75062, Lawrence Livermore Laboratory, 1973.

R. Duncan, A survey of parallel computer architectures, Computer, Feb. 1990, pp. 5-16.

M. J. Flynn, Very high-speed computing systems, Proc. IEEE, **54** (1966), 1901-1909.

K. Hwang, F. A. Briggs, Computer architecture and parallel processing, McGraw Hill, New York , 1984.

H. B. Keller, Numerical Solution of two point boundary value problems, SIAM, Philadelphia, 1976.

H. B. Keller, P. Nelson, Hypercube implementations of parallel shooting, Appl. Math. Comp., 1987.

O. A. Liskovets, The method of lines, (English Translation) J. Diff. Eqs., **1**, (1965), 1308.

L. Lustman, B. Neta, C. P. Katti, Solution of linear systems of ordinary differential equations on an Intel hypercube, SIAM J. Sci. Stat. Comput., **12** (1991), 1480-1485.

L. Lustman, B. Neta, W. Gragg, Solution of ordinary differential initial value problems on an Intel hypercube, Computers Math Applic., **23** (1992), 65-72.

L. Lustman, B. Neta, Software for the parallel solution of systems of ordinary differential equations, Naval Postgraduate School Technical Report NPS-MA-91-009, Monterey, CA,

1991.

S. G. Mikhlin and K. L. Smolitskiy, Approximate Methods for Solution of Differential and Integral Equations, Elsevier, New York, 1967.

B. Neta, C. P. Katti, Solution of initial value problems on a hypercube, Naval Postgraduate School Technical Report NPS-53-89-001, Monterey, CA, 1989.

M. J. Quinn, Designing efficient algorithms for parallel computers, Mc Graw Hill, New York, 1987.

H. S. Stone, An efficient parallel algorithm for the solution of a tridiagonal linear system of equations, J. ACM, **20**, (1973), 27-38.