CONCURRENCY: PRACTICE AND EXPERIENCE, VOL. 7(8), 795–822 (DECEMBER 1995)

Parallel solution of large-scale differential-algebraic systems

R.S. MAIER

University of Minnesota
Army High Performance Computing Research Center
1100 Washington Avenue So.
Minneapolis, MN 55415, USA

L.R. PETZOLD

Department of Computer Science
University of Minnesota
Minneapolis, MN 55455, USA

W. RATH

Fachbereich Mathematik
Postfach 964
09009 Chemnitz, Germany

SUMMARY

DASPK solves large-scale systems of differential-algebraic equations. It is based on the integration method in DASSL, but instead of a direct method for the associated linear systems which arise at each time step, the preconditioned GMRES iteration is applied in combination with an inexact Newton method. Two parallel versions of DASPK have been developed: DASPKF90, a Fortran 90 data parallel implementation, and DASPKMP, a message-passing implementation written in Fortran 77 with extended BLAS. The parallel versions have been implemented for the Thinking Machines Corporation (TMC) CM-5, a massively parallel multiprocessor, keeping the user interface relatively simple while allowing for portability to other massively parallel architectures. The codes have been demonstrated on several large-scale test problems, including three-dimensional formulations of the heat equation, the Cahn–Hilliard equation and a multi-species reaction–diffusion problem. The formulations are described, including detail on preconditioning the Krylov iteration, timing results and performance analysis.

1. INTRODUCTION

Large systems of differential-algebraic equations (DAEs) arise in many scientific applications. A common source of such problems are partial differential equations discretized by the method of lines in two or three spatial dimensions. This effort is aimed at providing a basis of software experience and solution methods for solving very large systems on a massively parallel architecture. It is assumed that the DAE system is of the general form

$$\mathbf{F}\left(t,\mathbf{y},\mathbf{y}'\right) = 0 \tag{1}$$

where $\mathbf{F}$, $\mathbf{y}$ and $\mathbf{y}'$ are N–dimensional vectors, and a consistent set of initial conditions $\mathbf{y}(t_0) = \mathbf{y}_0$, $\mathbf{y}'(t_0) = y_0'$ is given, i.e. $\mathbf{F}(t_0,\mathbf{y}_0,\mathbf{y}_0') = 0$. The focus of this work is on DASPK (differential-algebraic solver preconditioned Krylov)[1,2] which is a member of the DASSL (differential-algebraic systems solver)[3,4] family. In DASSL, the linear systems which arise at each time step are solved with dense or banded direct linear system solvers. Parallelization of the direct methods in DASSL has been considered for message-passing architectures in [5].

For sparse large–scale systems an iterative solver like the preconditioned GMRES (generalized minimum residual)[6] method can be quite efficient. In contrast to the solution of large-scale systems of ODEs $\mathbf{y}' = \mathbf{f}(t,\mathbf{y})$ with codes like LSODPK[7], a preconditioner is always needed for DAEs which are not ODEs.

CCC 1040–3108/95/080795–28
©1995 by John Wiley & Sons, Ltd.
Received 4 April 1994
Accepted 21 November 1994

The purpose of our paper is twofold: firstly to introduce two new parallel solvers for large-scale DAEs on the CM5, and secondly to report our experiences. The paper is organized as follows. Section 2 provides a short description of DASPK. Section 3 describes the two implementations of DASPK for the Thinking Machines Corporation (TMC) CM-5. The first version is DASPKF90, a data parallel implementation for the CM Fortran compiler. DASPKMP, the second version, is a message-passing implementation written in Fortran 77 with extended BLAS. Section 4 presents some numerical experiments applying the two DASPK versions and different preconditioners to the solution of several large-scale problems.

2. THE DASPK ALGORITHM

DASPK solves the system 1 of N differential–algebraic equations for $\mathbf{y}$ and $\mathbf{y}'$, for a specified range of the independent variable t. Values $\mathbf{y}(t_0) = y_0$ and $\mathbf{y}'(t_0) = y_0'$ at the initial time t_0 must be given as input. These initial values should be consistent, that is, $\mathbf{y}_0$, $\mathbf{y}_0'$, t_0 should satisfy equation 1. Here we summarize the main steps of the algorithm; for further detail, see [1].

Integration over the specified range of t is usually accomplished in a series of steps. The algorithm for computing an integration step involves replacing the derivatives with difference approximations and using a predictor-corrector method. In the predictor-corrector method an initial guess for the new solution is developed by evaluating the predictor polynomial, which interpolates solution values at previous time steps. The new solution is then computed by solving a non-linear system of equations in the corrector step. The predictor and corrector polynomials are specified using the backward differentiation formulas (BDF) of orders one through five. A complete description of the integration method is given in [3]. In each corrector step a sequence of nonlinear systems are solved by a Newton-type iteration. Each of these iterations requires the solution of a linear system. These linear systems are approximately solved by the preconditioned generalized minimum residual (GMRES) method[6].

In the corrector step, the Newton iteration solves the discrete non-linear system

$$\mathbf{F}\,(t,\mathbf{y},\alpha\mathbf{y}+\beta) = 0 \tag{2}$$

for the value $\mathbf{y}$, where α and β are constant during the iterations. The approximation $\mathbf{y}' \simeq \alpha\mathbf{y} + \beta$ arises from the BDF formula. Differentiating 2 with respect to $\mathbf{y}$ gives the Newton iteration matrix (of order N)

$$\mathbf{A} = \frac{\partial\mathbf{F}}{\partial\mathbf{y}} + \alpha\frac{\partial\mathbf{F}}{\partial\mathbf{y}'} \tag{3}$$

This iteration matrix appears in the linearized Newton system to be solved by DASSL or DASPK. In DASSL the iteration matrix $\mathbf{A}$ is explicitly computed and factored. The cost and storage of this is often prohibitive in large-scale computation, hence the need for iterative methods for solving the Newton system. One of the powerful features of the iterative approach is that it does not need to compute and store the iteration matrix $\mathbf{A}$ explicitly. Instead, it requires only the action of $\mathbf{A}$ times a vector $\mathbf{v}$. In DASPK, this matrix–vector product is approximated via a difference quotient on the function $\mathbf{F}$ in Equation 2, leading to

$$\mathbf{A}\mathbf{v} \simeq \left(\mathbf{F}\left(t,\mathbf{y}+\sigma\mathbf{v},\alpha(\mathbf{y}+\sigma\mathbf{v})+\beta\right) - \mathbf{F}\,(t,\mathbf{y},\alpha+\beta)\right)/\sigma \tag{4}$$

Each Newton, or non-linear, iteration requires the solution of a linear system

$$\mathbf{Ax} = \mathbf{b}, \tag{5}$$

where $\mathbf{A}$ is the $N \times N$ iteration matrix in 3 and $\mathbf{x}$, $\mathbf{b}$ are N-vectors. Up to four Newton iterations are allowed per time step. The termination criteria and other details for the Newton iteration are given in [1].

In the case of iterative methods, the linear system 5 is solved by the preconditioned GMRES iterative method. We refer to this as the linear iteration. Depending on the options chosen, DASPK uses the complete or the incomplete GMRES method and it may be restarted.

GMRES is one of a class of Krylov subspace projection methods[8]. The method starts with a given initial guess $\mathbf{x}_0$ for the system solution. Letting $\mathbf{x} = \mathbf{x}_0 + \mathbf{z}$, we obtain the system $\mathbf{Az} = r_0$, where $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$ is the initial residual. $\mathbf{z} = \mathbf{z}_l$ is then chosen in the Krylov subspace $\mathbf{K}_l = \text{span}\left\{\mathbf{r}_0, \mathbf{Ar}_0, \ldots, \mathbf{A}^{l-1}\mathbf{r}_0\right\}$. For the GMRES algorithm $\mathbf{z}_l$, hence $\mathbf{x}_l = \mathbf{x}_0 + \mathbf{z}_l$, is specified uniquely by the condition

$$||\mathbf{b} - \mathbf{Ax}_l||_2 = \min_{\mathbf{x} \in \mathbf{x}_o + \mathbf{K}_l} ||\mathbf{b} - \mathbf{Ax}||_2 = \min_{\mathbf{z} \in \mathbf{K}_l} ||\mathbf{r}_0 - \mathbf{Az}||_2 \tag{6}$$

Here, $||.||_2$ denotes the Euclidean norm.

GMRES uses the Arnoldi process [9] to construct an orthonormal basis of the Krylov subspace $\mathbf{K}_l$. This results in an $N \times l$ matrix $\mathbf{V}_l = [\mathbf{v}_1, \ldots, \mathbf{v}_l]$ and an $l \times l$ upper Hessenberg matrix $\mathbf{H}_l$ such that

$$\mathbf{H}_l = \mathbf{V}_l^T \mathbf{A} \mathbf{V}_l \quad \text{and} \quad \mathbf{V}_l^T \mathbf{V}_l = \mathbf{I}_l \tag{7}$$

where $\mathbf{I}_l$ is the $l \times l$ identity matrix.

The resulting GMRES algorithm, in which $\mathbf{l}_{max}$ and δ are given parameters, is described as follows.

Algorithm (GMRES)

1. Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$ and set $\mathbf{v}_1 = \mathbf{r}_0/||\mathbf{r}_0||_2$.
2. For $l = 1, \ldots, k$ do:
   - Form $\mathbf{Av}_l$ and orthogonalize it against $\mathbf{v}_1, \ldots, \mathbf{v}_l$ via:

$$\begin{aligned} \mathbf{w}_{l+1} &= \mathbf{Av}_l - \sum_{i=1}^{l} \mathbf{h}_{i,l}\mathbf{v}_i, \ \mathbf{h}_{i,l} = (\mathbf{Av}_l)^T \mathbf{v}_i \\ \mathbf{h}_{l+1,l} &= ||\mathbf{w}_{l+1}||_2 \\ \mathbf{v}_{l+1} &= \mathbf{w}_{l+1}/\mathbf{h}_{l+1,l} \end{aligned}$$

   - Compute $\rho_l = ||\mathbf{b} - \mathbf{Ax}_l||_2$.
   - If $\rho_l \leq \delta$, go to Step 3.
3. Compute $\mathbf{x}_l = \mathbf{x}_0 + \mathbf{V}_l \mathbf{y}_l$, and stop.

One can compute the residual norm $||\mathbf{b} - \mathbf{A}\mathbf{x}_l||_2$ without computing $\mathbf{x}_l$ at each step. Details of the GMRES algorithm and its implementation in DASPK are given in [1,6].

Preconditioning the linear system is necessary for most problems to achieve an adequate rate of convergence. Preconditioning in an iterative method for solving **Ax=b** means applying the method instead to the equivalent system

$$(\mathbf{P}^{-1}\mathbf{A})\mathbf{x} = \mathbf{P}^{-1}\mathbf{b} \tag{8}$$

where the left preconditioner **P** is chosen in advance. To accelerate convergence, **P** should be chosen to approximate the matrix A while keeping the system **P x=c** easy to solve for **x**. In some preconditioning methods, such as ILU, one must compute and store a preconditioner matrix explicitly. However, this matrix is constructed in such a way that it is much cheaper to generate and store than the actual iteration matrix. DASPK actually solves a scaled, preconditioned system, where the scaling depends on the user error tolerances. For further details, see [1].

3. PARALLEL IMPLEMENTATION

3.1. CM-5 system architecture

The Thinking Machines Corporation CM-5 is the successor of the CM-2/200 as a massively parallel multicomputer. The CM-5 is designed to support two programming models. On the one side is a data-parallel model, the HPF/CM-2/200 model, a model of a single instruction multiple data (SIMD) machine. On the other side the CM-5 supports a message-passing model, a single-program multiple-data (SPMD) model.

The CM-5 is equipped with three networks connecting the processing nodes: a data network, a control network and a diagnostic network. The data network is an asynchronous network with a point-to-point 5–20 Mbyte/s bandwidth. The control network provides coordination and synchronization, broadcast and reduce operations, e.g. global min and max, and parallel prefix operations.

Each node (PN) of the CM-5 consists of one RISC processor, currently a 33 MHz Cypress Sparc chip set and four vector units (VUs) on two data path (DP) chips. A node has 32 Mbytes of DRAM memory broken into four banks. A network interface (NI) connects the node to the control and data network. It can inject data at 20 Mbyte/s, and read data at 40 Mbyte/s off the MBus. The MBus is 64 bit wide and capable of supporting 256 Mbyte/s.

The Sparc chip, the heart of a node, has a separate integer and FPU registers and supports 22 MIPS and 5 Mflops. It controls the vector units and network interface through memory mapped registers. The Sparc uses the vector units as memory controllers and can access the entire 32 Mbytes of DRAM memory. Neither the network interface, nor the vector units can fetch its own instructions. The Sparc is the sole instruction scheduler for the entire node.

Each vector unit has a peak performance of 32 Mflops, which results in a 128 Mflops peak performance of the whole node. A vector unit has 8 Mbyte of private memory and can do one load or store per clock cycle. The vector units allow vector lengths of 1,4,8 and 16, where 8 is the optimal length for the vector units of a CM-5 node.

The design of the nodes is maybe the weakest part in the CM-5 system architecture. The vector units are slaves to the Sparc, which does the instruction scheduling through memory mapped registers. Each vector unit can access its local 8 Mbyte of memory with a bandwidth of 128 Mbyte/s. The Sparc is a bottle neck and limits the intervector unit

transfer. All memory transfers are made by the Sparc reading or writing its own registers at 20 Mbyte/s. The nodes exhibit memory locality since moving data between vector units is slow compared with local access time, or the nodes MBus speeds. If a message-passing program should use the vector units, the node is treated as a small SIMD machine. The Sparc fulfils the role of a control processor, and the vector units act as nodes. The node level program must then be written in a data-parallel language as CM Fortran or C*.

The nodes of a CM-5 are grouped into independent subsets called partitions. A partition must consist of 2^n nodes and each partition has one control processor, called the partition manager. The partition managers are conventional workstations and provide a conventional UNIX environment. In the data-parallel programming model, a partition manager performs all scalar operations and acts as instruction scheduler for the nodes. In the message-passing model, it initiates the CM-5 processes.

We performed all development and testing on the Army High Performance Computing Research Center (AHPCRC) CM-5 installed at the Minnesota Supercomputer Center. The 544-processor CM-5 is typically operated with 32-PE and 512-PE partitions or 32-PE, 128-PE, 128-PE and 256-PE partitions. It has a peak performance of 70 Gflops (512-PE), and current applications can sustain 10–20 Gflops for large problems.

3.2. Implementation of DASPK on the CM-5

There are two versions of DASPK available for the CM-5. The first one is DASPKF90, a data parallel version for the CM Fortran compiler. This compiler is based on Fortran 90. Data sets, as arrays or matrices, are laid out in a global address space. The user can specify if these data sets are located on the front end (partition manager) or as parallel arrays across all processors. Communication between the nodes and the partition manager is hidden in Fortran 90 or parallel statements. A CM Fortran program is a single program that runs synchronously on all nodes of the CM-5. The second version of DASPK for the CM-5 is DASPKMP, a message-passing implementation. This version is written in standard Fortran 77. Using a communication library (CMMD), the user handles load balancing and data layout and specifies all communications among the nodes via message-passing. DASPKMP uses Fortran level 1 BLAS and Fortran BLAS-like routines for all vectorizable constructs. However, there are currently no optimized BLAS routines available for the CM-5. As a consequence, the current version of DASPKMP does not support the vector units. Message-passing subroutine calls are isolated in a small number of routines called only by BLAS and extended BLAS routines. The message–passing subroutine calls are currently made to the TMC CMMD Library, and not to a general protocol such as PVM. Thus DASPKMP is not strictly portable to other MPP machines, although the difficulty of replacing the CMMD calls with PVM calls is not very great.

DASPKF90 and DASPKMP are both Fortran-callable subroutines with some special properties which make it easier to develop complex application programs. DASPKF90 and DASPKMP employ *dynamic memory allocation* to set up initial work arrays. The calling program need not provide either code with workspace. Instead, DASPKF90 and DASPKMP use a malloc procedure to allocate internal storage the first time they are called.

DASPKF90 and DASPKMP also use a *reverse communication* interface. DASSL and DASPK use a forward communication interface in which the user-defined routine RES which evaluates the residual of the DAE is callled by the DAE solver. There is no predefined RES routine as in DASSL and serial DASPK versions. Instead, DASPKF90 and DASPKMP

return to the calling program each time they need to evaluate the residual. After evaluating the residual the calling program simply calls DASPKF90 or DASPKMP again, passing it the residual. The RES calling sequence is no longer strictly defined. This type of interface is illustrated in the following example:

```
do while (usrtask.eq.'delta' .or. usrtask.eq.'psol')
  call daspk(neq,t,y,yprime,tout,info,rtol,atol,idid,
 *           rwork,iwork,delta,wt,psol,usrtask,usrflag)
  if (usrtask.eq.'delta') then
    call myresidual(m,neq,coeff,y,yprime,delta)
  elseif (usrtask.eq.'psol') then
    call preconditioner(neq,t,y,yprime,wt,psol,usrflag)
  endif
enddo
```

where `myresidual` is the RES subroutine, and `preconditioner` is the preconditioner routine. Note that this code segment defines a simple loop around the call to DASPK. A detailed description of the DASPKF90 and DASPKMP subroutine arguments can be found in [2].

3.3. Analysis of DASPK computational cost

The computational cost of using DASPK is divided between the cost of the user's residual and preconditioner subroutines and the DASPK subroutine. This Section describes the primary costs of the DASPK subroutine. As indicated in preceding Sections, DASPK employs a variable-step, variable-order predictor–corrector algorithm which requires the solution of one or more non-linear systems of equations at each integration time step. The solution of each non-linear system is accomplished with a Newton-like method, in which a Newton step involves the solution of a linear system of equations. The solution of these linear systems (via GMRES) represents the dominant cost in DASPK, and it is reasonable to analyze the cost of DASPK in terms of the GMRES kernel.

To further simplify the analysis, all scalar work in the GMRES algorithm is ignored, including the solution of the least-squares subproblem (although the subproblem can be solved in parallel, it does not contribute significantly to the total operation count for large problems).

By counting the number of operations in the GMRES algorithm shown in Section 2 we can see that the approximate cost is $N(3 + k(5 + 2k))$ flops, where N is the number of unknowns and k is the maximum Krylov subspace dimension (i.e. the cost of k Krylov iterations). Note that the flop count grows as $O(Nk^2)$, so there is good reason to prefer low-dimensional subspaces.

The performance of the GMRES kernel is illustrated in Figure 1 on a 512-PE CM-5 partition. The gigaflop rate describes an S-shaped function of problem size, reflecting the transition from a communication-dominated size regime to a near-peak rate for the algorithm. We note that the performance is problem-independent given a matrix of the appropriate size and subspace dimension $k = 10$.

The GMRES iteration is a combination of local and global data-parallel computation and a significant number of scalar operations. Step 2(a) of the GMRES algorithm in

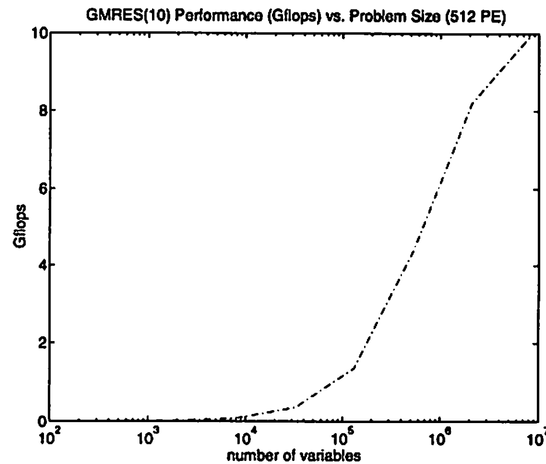

Figure 1. Performance of GMRES algorithm on the CM-5 (Fortran 90 compiler). Curve shows floating point operations per second (Gflops) sustained in the GMRES kernel for subspace dimension $k = 10$ for problem sizes ranging from $N = 2^9$ to 2^{23} variables

Section 2 illustrates a number of data-parallel steps involving $O(N)$ flops. These parallel steps include purely local computation (linked triads, or DAXPY) and global computations (inner products). The GMRES algorithm also includes a large number of sequential steps. Step 2(b) of the GMRES algorithm in Section 2 illustrates that a small least-squares problem (order k) must be solved. Given a DASPK application with, say, $N/p \leq 50$ variables per processor, the time required for scalar work can easily dominate the time required for execution of data parallel statements. Note in Figure 2 that execution time is approximately constant for $N \leq 10^5$ variables ($N/p \leq 200$). In our implementation, we have not attempted to parallelize the solution of the small least-squares problem. The potential for parallelism has been considered in [10] but we are unaware of any actual implementations.

Ideally, the DASPK/GMRES iteration should involve a low-dimensional Krylov subspace, e.g. $k \leq 10$. Although the subspace dimension is modified automatically in DASPK, the user can also influence the dimension by specifying a maximum subspace size and by supplying or improving the preconditioner subroutine. Section 4.3.2. describes the effect of preconditioner quality in reducing the subspace dimension.

4. NUMERICAL EXPERIMENTS

This Section describes the mathematical formulation and computational solution of several problems arising from method-of-lines discretization of partial differential equations. Section 4.1. discusses the three-dimensional heat equation. The heat equation forms the basis for an analysis of parallel performance because it displays most of the significant scaling behaviors associated with parallel DASPK while being relatively simple to understand and analyze. Section 4.3. discusses a multispecies reaction–diffusion problem in two and three spatial dimensions. The two-dimensional formulation is used for compari-

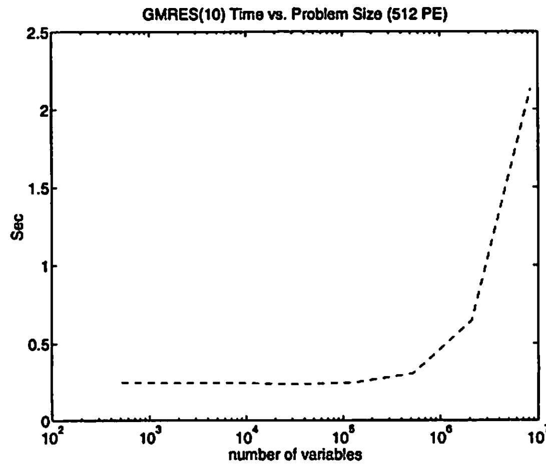

Figure 2. Performance of GMRES algorithm on CM-5. Curve shows total time (s) spent in GMRES kernel for subspace dimension $k = 10$ for problem sizes ranging from $N = 2^9$ to 2^{23} variables

son with a sequential implementation of DASPK and for computational experiments with preconditioners. Section 4.4. discusses the three-dimensional Cahn–Hilliard equation. The Cahn–Hilliard equation demonstrates an especially stiff problem, illustrating the need for powerful preconditioners and the need for additional research into parallel preconditioners for sparse, non-symmetric operators.

4.1. Three-dimensional heat equation

The three-dimensional heat equation is formulated as

$$\begin{array}{rclll} u_t & = & \Delta u & (x,y,z) \in \Omega & \\ u & = & 0 & (x,y,z) \in \Gamma & (9) \\ u & = & g(x,y,z) & t = 0,(x,y,z) \in \Omega & \end{array}$$

where $\Omega \subset \mathrm{R}^3$ is the unit cube in the first orthant $(0 \leq x,y,z \leq 1)$, with boundary Γ and $0 \leq t \leq 10.24$. The initial value function $g(x,y,z)$ is defined as

$$g(x,y,z) = 64\left(x(1-x) + y(1-y) + z(1-z)\right) \tag{10}$$

which describes a temperature gradient radiating from the 'hot' center of Ω. The domain Ω is discretized with a uniform Cartesian product mesh with M internal points in each direction, with spacing $h_x = h_y = h_z = 1/(M+1)$. Spatial derivatives are discretized via central finite differences. For

$$y_{i,j,k} = u(ih_x,jh_y,kh_z) \qquad (0 \leq i,j,k \leq M+1) \tag{11}$$

the discrete problem is

$$
\begin{aligned}
y'_{i,j,k} \;=\; & (h_x)^{-2}(y_{i+1,j,k} + y_{i-1,j,k} + y_{i,j+1,k} \\
& + y_{i,j-1,k} + y_{i,j,k+1} + y_{i,j,k-1} - 6y_{i,j,k})
\end{aligned}
\tag{12}
$$

and the total size of the system is $N = M^3$.

4.1.1. Computational cost of solving the heat equation

The heat equation on the unit cube illustrates most of the important scaling behaviors of DASPK and at the same time may be coded in a very concise form. To illustrate the computational and communication cost, the residual is expressed in Fortran 90 as:

```
 double precision, array(m,m,m) :: y, yprime, delta
 ...
 delta = yprime - (eoshift(y,1,1) + eoshift(y,1,-1) +
&                  eoshift(y,2,1) + eoshift(y,2,-1) +
&                  eoshift(y,3,1) + eoshift(y,3,-1) -
&                  6 * y ) * coeff
```

where `y(m,m,m)` corresponds to an $m \times m \times m$ mesh. The resulting operation count is $9m^3$ flops. The communication cost is $6m^3$ send/gets (the `eoshift` function returns a copy of the array argument shifted one or more indices in a single dimension, necessitating communication of the shifted elements). Some commonly used preconditioners for the heat equation require $9m^3k$ flops, where k is the order (in the case of polynomial preconditioners) or number of iterations (in the case of an iterative method such as Jacobi). These types of preconditioners involve repeated application of the discrete Laplacian operator and so generate operation counts which are multiples of the residual.

4.1.2. Using DASPKMP for the heat equation

When using DASPKMP, the user must initialize and control all communications in computing the residual of the heat equation. Approximately 200 lines of Fortran 77 code were needed to control the communication between the processors via calls to the CMMD library and compute the 7-point stencil used in the residual and preconditioner subroutines. A polynomial preconditioner of order three provided in [2] was used, along with error tolerances `rtol` $= 0$ and `atol` $= 10^{-3}$. The heat equation was solved for the cases $M = 32,64,128$ on partitions with 32,128,256 and 512 processors.

Table 1 shows the results for the message-passing version of DASPK. Columns 3–5 show the times for the complete program, the residual subroutine and the third-order preconditioner subroutine. In the sixth column, the speed-up in execution time is given, using the 32-PE partition as a baseline. As might be expected, the speed-ups improve for larger problems. The improvement is due to the fact that message length increases with problem size. As message length increases, the effect of message start-up cost (latency) on speed-up is diminished.

Table 1. Heat equation solution time using DASPKMP. RES and PRE represent time spent in residual and preconditioner routines; remainder is spent in DASPKMP. Speed-up is ratio of execution time to 32-PE total. See text for discussion of observed superlinear speed-up

No. of Proc.	M	Execution Time (sec) Total	RES	PRE	Speedup
32	32	210	37	122	–
	64	3587	654	1990	–
	128	> 12 hours	–	–	–
128	32	68	12	39	3.1
	64	891	157	520	4.0
	128	18374	3392	10245	–
256	32	37	7	21	5.6
	64	438	80	253	8.2
	128	8251	1522	4555	–
512	32	23	4	13	9.0
	64	242	44	143	14.8
	128	3588	618	2015	–

4.1.3. Using DASPKF90 With Data Reshaping

In the current implementation of DASPKF90 the subroutine argument arrays are one-dimensional arrays of size $N = M^3$, the number of equations. The discrete problem 12 suggests that the residual subroutines and the preconditioner should work with three-dimensional arrays. If three-dimensional arrays are used in these routines, it is necessary to map the one-dimensional arrays used in DASPKF90 into three dimensions and vice versa. The CM Fortran function `reshape` can be used for this purpose, giving the following code for the residual subroutine `res`:

```
      subroutine res (m,neq,coeff,y,yprime,delta)
      integer :: neq,m
      double precision :: coeff
      double precision, array(neq) :: y, yprime, delta
      double precision, array(m,m,m) :: yt

      yt = reshape([m,m,m],source=y)

      yt = ( eoshift(yt,1,1) + eoshift(yt,1,-1) +
     &       eoshift(yt,2,1) + eoshift(yt,2,-1) +
     &       eoshift(yt,3,1) + eoshift(yt,3,-1) -
     &       6 * yt ) * coeff

      delta = yprime - reshape([neq],source=yt)
      return
      end
```

In this subroutine, `m` is the number of meshpoints in one dimension, `neq` the number of equations, `y` is the solution vector, `yprime` the derivative vector, `delta` the vector

of residuals, `yt` is a three-dimensional work array and `coeff` $= 1/h_x^2$. In summary, the residual computation using reshaping needs the following four steps:

- reshape the unknowns into a 3D mesh
- compute 7-point stencil
- reshape the mesh into a vector
- compute `delta`, the residual of the discrete heat equation 12.

Note that four lines of code with six calls of the CM Fortran function `eoshift` were used to compute the seven-point stencil, while in the message-passing version about 200 lines of Fortran 77 code were used.

As in the preceding Section, the polynomial preconditioner of order three was used. The heat equation was solved for the cases $M = 32,\ M = 64,\ M = 128$ on partitions with 32,126,256 and 512 processors.

Table 2. Solution time using DASPKF90 with reshaping on 3D heat equation. RES and PRE represent time spent in residual (reshaping) and preconditioner (reshaping) routines. Times in parentheses are seconds used in reshaping data. Speed-up is ratio of execution time to 32-PE total

No. of Proc.	M	Total	Execution Time (seconds) RES	(Reshape)	PRE	(Reshape)	Speedup
32	32	108	41	(32)	60	(33)	–
	64	1111	446	(363)	615	(361)	–
	128	26181	11563	(10363)	13853	(10219)	–
128	32	48	17	(12)	26	(11)	2.2
	64	357	143	(117)	197	(121)	3.1
	128	4997	2210	(1920)	2606	(1740)	5.2
256	32	38	12	(8)	21	(8)	2.9
	64	232	92	(75)	127	(77)	4.8
	128	2540	1063	(906)	1381	(906)	10.3
512	32	31	10	(6)	17	(6)	3.5
	64	145	55	(42)	80	(42)	7.7
	128	1806	763	(670)	988	(704)	14.5

Table 2 shows the time (in seconds) needed for the complete program, the residual subroutine and preconditioner routine for different problem sizes and on different partitions. The third column shows the complete time which the program used to solve the discrete heat equation. Columns 4 and 6 show the time spent in the user-supplied RES subroutine and the preconditioner, including the reshaping and all computations in these routines. Columns 5 and 7 show the time for reshaping alone in the user supplied subroutines. Column 8 gives speed-up in total execution time compared to the 32-PE partition.

Reshaping is a global communication process. The data are laid out in a new way and the result is an extensive communication exchange between the processors of the partition. As expected, Table 2 shows that most of the time is spent in reshaping from one to three dimensions and vise versa. The time for reshaping increased faster than the problem size, and finally reached 90% of the time needed for the residual and polynomial preconditioner

routines. Note that the time required by DASPKF90 is the remainder of total time after subtracting the RES and PRE times, typically less than 10%.

The speed-ups in Table 2 indicate that DASPKMP obtains a larger speed-up for $M = 32$ and $M = 64$ than DASPKF90. The primary reason for the difference is that DASPKF90 utilizes the vector hardware of the CM-5. Thus, while DASPKF90 is faster than DASPKMP, the efficiency of vector utilization depends on vector length and problem size. Spreading the same problem over more processors tends to damp vector processor efficiency.

4.1.4. *Using DASPKF90 without reshaping*

In the preceding subsection it was shown that reshaping from one to three dimensions and vice versa cost most of the time for solving the heat equation. There are two possibilities to avoid the reshaping. The first one is to develop an interface which would allow DASPKF90 to accept data structures of arbitrary shape and layout, similar to the current TMC scientific library (CMSSL) routines. With the current TMC software, there does not seem a way to do this with complete generality or elegance. For purposes of experimentation we have coded custom versions of DASPKF90 which accept data structures of various kinds. Again, this is not a general or elegant approach to building library-quality software.

The second possibility to avoid the reshaping is to use the new 'Array aliasing utilities' for CM Fortran version 2.1 Beta 1[11]. A set of CM Fortran Utility Library procedures enable the user to create an array alias, which addresses memory already allocated for another array but treats it as being of a different type, shape or layout. This procedure was used to create three-dimensional aliases for the one–dimensional DASPKF90 subroutine arguments `y`, `yprime`, `rtol`, `atol`, `rwork`, `delta`, `wt` and `psol`. These aliases were then used as subroutine arguments for the residual and preconditioner subroutine. The result is the following code:

```
     call res(m,neq,coeff,yalias,yprimealias,deltaalias)
     ...
     subroutine res (m,neq,coeff,y,yprime,delta)
     integer :: neq,m
     double precision :: coeff
     double precision, array(m,m,m) ::  y, yprime, delta

     delta = yprime - (eoshift(y,1,1) + eoshift(y,1,-1) +
    &                  eoshift(y,2,1) + eoshift(y,2,-1) +
    &                  eoshift(y,3,1) + eoshift(y,3,-1) -
    &                  6 * y ) * coeff
     return
     end
```

This residual subroutine computes the 7-point stencil and then accumulates the residual `delta` of the discrete heat equation. Reshaping is avoided because alias arrays are passed as subroutine arguments to the residual routine.

Table 3 shows the results for this approach. Columns 3–5 show the timings for the complete program, the residual subroutine and the third-order polynomial preconditioner routine. Columns 6–8 show the performance of the residual, preconditioner and DASPKF90 subroutines in Gflops. The Gflop rates are calculated on the basis of operation counts

Table 3. Heat equation solution time and Gflop rate using DASPKF90 with equivalence library. RES and PRE are user-defined residual and preconditioner routines. DASPK is DASPKF90 subroutine. Speed-up is ratio of execution time to 32-PE total

No. of proc.	M	Execution time (sec) Total	RES	PRE	Gflops RES	PRE	DASPK	Speed-up
32	32	45	9	28	0.04	0.05	0.05	–
	64	381	82	248	0.06	0.09	0.19	–
	128	6152	1332	4000	0.08	0.10	0.38	–
128	32	23	5	14	0.07	0.09	0.09	1.9
	64	120	26	79	0.20	0.27	0.62	3.2
	128	1709	375	1110	0.28	0.38	1.41	3.6
256	32	21	4	12	0.08	0.11	0.09	2.1
	64	78	17	49	0.31	0.43	0.83	4.9
	128	751	163	490	0.53	0.70	2.46	8.2
512	32	21	4	12	0.08	0.11	0.08	2.1
	64	65	13	40	0.40	0.53	0.84	5.9
	128	469	101	310	0.93	1.21	4.72	13.1
	256	7991	1745	5214	1.17	1.57	7.87	–

discussed in Sections 3.3. and 4.1.1.. The Gflop rates for DASPKF90 are based on the time spent in that routine against the number of flops in the GMRES algorithm. The number of flops in the GMRES algorithm is approximated as $fl = M^3L(3 + \bar{k}(5 + \bar{k}))$, where M is the mesh parameter, L is the number of non-linear iterations and $\bar{k}$ is the average Krylov dimension (number of linear iterations divided by number of non-linear iterations). The sustained, or average, Gflop rate is not shown in Table 3 but can be calculated as the time-weighted average of the individual Gflop rates. Column 9 reports the speed-up in execution time compared to the 32-PE partition.

Avoiding the reshaping and using the array aliases results in better performance in all cases, relative to Table 2. With increasing problem size the performance improves more than proportionately and for large problems the user-supplied subroutines using the alias approach are seven times faster than the subroutines which use the `reshape` function.

The Gflop rates increase with problem size and for $M = 256$ a sustained rate of 3.2 Gflops was observed on the partition with 512 processors, $\simeq 5\%$ of the peak performance. For $M = 256$ the problem has about 16.8 million equations and required 4.84 Gbyte of main memory and 2.22 h of CPU time.

Finally, we note that the current Equivalence Library does not allow aliasing arrays of arbitrary dimensions. The Library is currently a beta release. Furthermore, it was found that, for power of two array dimensions, the Library can apparently be bypassed entirely with no change in results for the applications described here.

4.2. Analysis of performance

4.2.1. Comparison of programming models

The DASPKMP version is in most most cases slower than the two DASPKF90 versions. As described in Section 3.2., DASPKMP does not use the vector units and therefore this

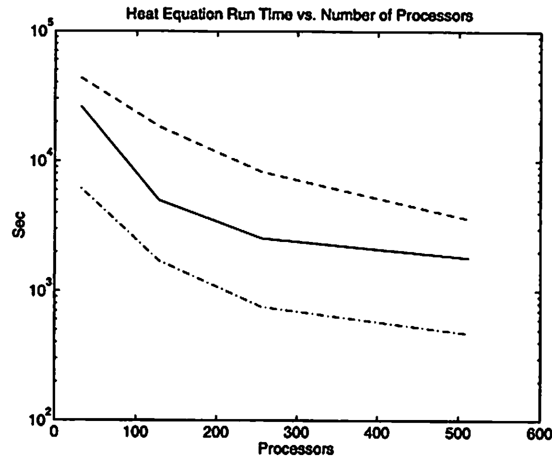

Figure 3. Performance comparison for three programming models. Curves represent execution time for heat equation with 128^3 *mesh on 32, 128, 256 and 512 processors (- - -) DASPKMP, (—) DASPKF90 with reshaping, (- · · ·) DASPKF90 with array aliasing*

result is expected. However, for $M = 32$ with either 256 processors or 512 processors, the message-passing version lies between the two approaches for DASPKF90. These problems are too small for their partition sizes and the vector units cannot be used efficiently. Figure 3 summarizes the relative performance of DASPKMP and DASPKF90 for an $M = 128$ instance of the heat equation. Note that DASPKF90 with the Equivalence Library is almost an order of magnitude faster than DASPKMP.

At the current time the data parallel version DASPKF90 is the method of choice for most problems. As long as there are no efficient BLAS routines which use the vector units of the CM-5, the message-passing version DASPKMP cannot get a high Gflop rate on the CM-5. The situation for other massively parallel computers is different, since most of these machines use a scalar processor on the node.

4.2.2. Speed-up

Fixed-size speed-up is defined here as T_{32}/T_{PE}, where T_{32} is execution time on a 32-PE partition (the smallest available) and $T_{PE} \in \{T_{32}, T_{128}, T_{256}, T_{512}\}$. A graph of fixed-size speed-up against number of processors is presented in Figure 4, based on an $M = 128$ instance of the heat equation using DASPKF90. Note that full (16-fold) speed-up is not obtained for the 512-PE partition. Also, note that slightly superlinear speed-up is obtained for the 256-PE partition. Both observations are related to the fact that the number of integration steps differs on the partitions even though the same problem is being solved. The 128-PE partition required the fewest integration steps while the 512-PE partition required the most. This phenomenon is described in Section 4.2.3.. For the present discussion note that, while the number of integration steps is sufficient to explain speed-up variations for T_{128} and T_{256}, it is not sufficient to explain the subideal speed-up of T_{512}, which reflects decreased efficiency of processor and vector utilization.

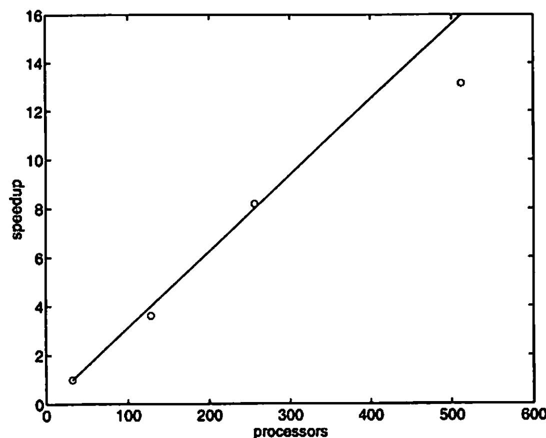

Figure 4. Fixed problem speed-up. (—) ideal processor speed-up for M = 128 instance of 3D heat equation relative to 32-PE partition time, (○) speed-up for 128, 256 and 512 processors

4.2.3. *Effect of rounding errors on performance*

In the preceding Section it was noted that the same problem may require a slightly different number of integration steps on different-size partitions. This results from different rounding errors on different-size processor partitions. Different rounding errors arise due to different orders of summation in large global sums. Thus, this phenomenon affects very large and/or poorly scaled problems.

For most data structures, including one-dimensional arrays, the mapping of array elements to processors depends on the number of processors. As a result, the order of global summation differs, because each processor computes partial sums before contributing a partial sum to the global sum. Global sums (e.g. inner product) are sensitive to the order of summation. For sums involving very many terms (e.g., $> 10^5$), differences in the 13th decimal digit can frequently be observed between two processor partitions of different sizes (using double precision).

Global sums are a common operation in DASPK, appearing in the calculation of all inner products and norms. For example, the calculation of matrix entries h_{ij} appearing in step 2(a) of the GMRES Algorithm involve global sums. Since the matrix **H** is used to compute the direction vector for the non-linear iteration, small variations in **H** lead to small variations in trajectory. For very large problems, small variations in trajectory can accumulate and lead to differences in the number of linear iterations needed to solve the current non-linear iteration, and eventually to differences in the number of time steps and other behaviors of the code. This is due to the fact that DASPK makes its decisions on the basis of these norms. We have experimented with scaling in computing norms and observed that, while scaling does improve consistency, it does not eliminate these differences.

4.2.4. *Effect of preconditioning on performance*

Least-squares polynomial preconditioning approximates the action of $\mathbf{A}^{-1}$ on a vector by

$$\mathbf{A}^{-1}\mathbf{v} \simeq s_k(\mathbf{A})\mathbf{v} = \sum_{i=0}^{k} \alpha_i \mathbf{A}^i \mathbf{v}$$

where $s(\lambda)_{k-1}$ minimizes $\|1 - s(\lambda)\|_w$ over all polynomials s of degree $\leq k - 1$ and $w(\lambda)$ is a non-negative weight function. The polynomial $s(\lambda)_{k-1}$ is computed as a linear combination of the polynomials $t_i(\lambda) = (q_i(0) - q_i(\lambda))/\lambda$,

$$s(\lambda)_{k-1} = \left[\sum_{i=0}^{k} q_i(0)t_i(\lambda)\right] / \left[\sum_{i=0}^{k} q_i(0)^2\right]$$

where $q_i(\lambda), i = 0,\ldots,n$, are the orthogonal polynomials with respect to $w(\lambda)$. For details of the derivation and a listing of some polynomial coefficients, see [12].

Table 4 shows the effect of polynomial preconditioning on execution time, Krylov subspace size, and integration steps required in solving the heat equation. Each term in the polynomial requires a calculation similar to the heat equation residual, using the 7-point stencil. Note that while preconditioning substantially reduces the number of integration steps and the average subspace dimension, the total time is not reduced proportionately.

Table 4. Effect of preconditioning on performance. Results for solving 3D heat equation with 128^3 mesh on 128 processors, using least-squares polynomial preconditioning

Preconditioner order (k)	Integration steps	Average Krylov dimension	Time (s)	% time in DASPK
0	692	16.4	1904	37
3	278	11.3	1718	13
6	209	8.7	1776	8
9	180	6.8	1779	6

It is expected that the preconditioner may represent the dominant cost in a DASPK calculation. A typical preconditioner is more expensive than the residual subroutine. Table 5 illustrates that an order-3 polynomial preconditioner is three times more expensive than the residual. Overall, DASPK requires about 13% of the total time, the remainder being in the residual and preconditioner routines. Increasing the order of the preconditioner would reduce the fraction of time in DASPK and vice versa.

4.2.5. *Effect of stiffness*

If the 3D heat equation grid is refined, we expect that the discrete Laplacian will become increasingly ill-conditioned, i.e. the problem becomes increasingly stiff. Refinement has the following effects on the DASPK code:

Table 5. Execution time profile of heat equation solution. Results for solving 3D heat equation with 128^3 mesh on 128 processors, using $k = 3$ polynomial preconditioning

Routine	Time (s)	% of total time
Residual	378	22
preconditioner	1114	65
DASPK	226	13
Total	1718	100

- Increased stiffness causes difficulty for the linear solver, automatically increasing the average Krylov dimension.
- Attempts to increase the Krylov dimension above the default maximum cause a non-linear iteration 'failure', resulting in an automatic reduction in the integration step size, which increases the number of integration steps.

Thus, mesh refinement will cause total execution time to increase due to the increased number of linear iterations, as well as the increased number of unknowns.

Figure 5 illustrates the effect of mesh refinement on DASPK performance. Solution of the 3D heat equation on a 256^3 mesh required 17 times more time than the 128^3 mesh (8073 as against 468 s). However, the 256^3 problem used 8 times more meshpoints and 2.8 times more linear iterations than the 128^3 problem. Thus, the increased difficulty of the problem ($\simeq$ 22 times) was slightly offset in terms of run time by increased efficiency of processor utilization.

4.3. Multispecies food web problem

The multispecies food web [13] simulates competition and/or predator–prey relationships in a spatial domain, and is a special case of a general reaction–diffusion problem. The following DAE formulation represents s species, where $c_i, i = 1, \ldots, p$, represent $p = s/2$ prey species concentrations, and $c_i, i = p+1, \ldots, s$, are the $p = s/2$ predator species (with infinitely fast reaction rates). The $c_i(x,t)$ are distributed in space and time such that

$$\begin{cases} \frac{\partial c_i}{\partial t} = g_i(c) + d_i \Delta c_i & (i = 1, 2, \ldots, s/2+1) \\ 0 = g_i(c) + d_i \Delta c_i & (i = s/2+1, \ldots, s) \end{cases} \tag{13}$$

The spatial domain Ω is taken to be the unit square or cube in the first orthant, i.e $\Omega \subset \mathbf{R}^{nd}, nd = 2,3$. The boundary conditions are of Neumann type (zero normal derivatives) everywhere. The reaction terms

$$g_i(c) = c_i(b_i + \sum_{j=1}^{s} a_{ij} c_j) \tag{14}$$

have coefficients $A \in \mathbf{R}^{s \times s}$, $b \in \mathbf{R}^s$, defined as [1]:

$$\begin{cases} a_{ii} = -1 \quad (\text{all } i) \\ a_{ij} = -0.5 \cdot 10^{-6} \quad (i \leq p, j > p) \\ a_{ji} = 10^4 \quad (i > p, j \leq p) \\ a_{ij} = 0 \quad (otherwise), \end{cases} \tag{15}$$

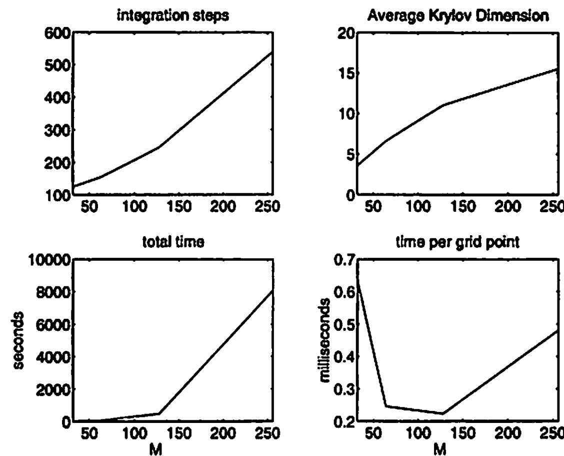

Figure 5. Effect of mesh refinement on performance; solution statistics for solving heat equation with DASPKF90 (equivalence) using polynomial preconditioning (k = 3), 512 processors and M = 32,64,128,256. Upper left: Number of integration steps against M. Upper right: Average Krylov dimension against M. Lower left: Total time (s) against M. Lower right: Total time per grid point (ms) against M

$$\left\{ \begin{array}{lcl} b_i & = & (1 + \alpha \prod_{j=1}^{nd} x_j + \beta \prod_{j=1}^{nd} \sin(4\pi x_j)) \quad (i \le p) \\ b_i & = & -b_{i-p} \quad (i > p) \end{array} \right. \tag{16}$$

In this problem, $\alpha = 50$ and various values of β are used. The diffusion coefficients are defined

$$\left\{ \begin{array}{lcl} d_i & = & 1 \quad (i \le p) \\ d_i & = & 0.05 \quad (i > p). \end{array} \right. \tag{17}$$

The initial conditions used for this problem are taken to be simple peaked functions that satisfy the boundary conditions and very nearly satisfy the constraints, given by

$$\begin{array}{lcl} c_i & = & 10 + i \left(16 \prod_{j=1}^{nd} x_j (1 - x_j) \right)^2 \quad (i = 1, \ldots, p) \\ c_i & = & - \left(b_i + \sum_{j=1}^{p} a_{ij} c_j \right) a_{ii}^{-1} \quad (i = p+1, \ldots, s). \end{array}$$

We discretized the PDE system 13 and boundary conditions with central differences on an $M \times M$ mesh for $nd = 2$ and correspondingly for $nd = 3$. The resulting DAE systems have $N = sM^2$ and $N = sM^3$ equations and unknowns.

The equations of the discrete system are ordered by species and then by mesh point. Thus if $\mathbf{c}_i = (c_{i,1}, \ldots, c_{i,M^2})^T$ is the vector of species i variables at all mesh points, then $y = (\mathbf{c}_1, \ldots, \mathbf{c}_p, \mathbf{c}_{p+1}, \ldots, \mathbf{c}_s)^T$ is the vector of all variables. The DAE system is then of the form

$$\begin{aligned} \mathbf{c}_i' &= \mathbf{f}_i = \mathbf{R}_i(t,\mathbf{y}) + \mathbf{S}_i(t,\mathbf{c}_i) \quad (i \le p) \\ 0 &= \mathbf{f}_i = \mathbf{R}_i(t,\mathbf{y}) + \mathbf{S}_i(t,\mathbf{c}_i) \quad (i > p) \end{aligned}$$

where the reaction part $\mathbf{R}_i$ depends only on the variables $c_{j,m}, j = 1,\ldots,s$ at mesh point m, and the diffusion part $\mathbf{S}_i$ depends on $\mathbf{c}_i$ but no other $\mathbf{c}_j$. Writing the DAE system as

$$0 = \mathbf{F}(t,\mathbf{y},\mathbf{y}') = (\mathbf{c}_1' - \mathbf{f}_1, \ldots, \mathbf{c}_p' - \mathbf{f}_p, -\mathbf{f}_{p+1}, \ldots, -\mathbf{f}_s)^T \tag{18}$$

we define the DASPK Jacobian in the form

$$\mathbf{J} = \alpha \mathbf{F}_{y'} + \mathbf{F}_y = \alpha \begin{pmatrix} \mathbf{I}_1 & 0 \\ 0 & 0 \end{pmatrix} - \frac{\partial \mathbf{f}}{\partial \mathbf{y}} = \alpha \bar{\mathbf{I}}_1 - \frac{\partial \mathbf{R}}{\partial \mathbf{y}} - \frac{\partial \mathbf{S}}{\partial \mathbf{y}} \tag{19}$$

where $\mathbf{f} = (f_1, \ldots, f_s)^T = \mathbf{R} + \mathbf{S}$, and $\bar{\mathbf{I}}_1 \equiv \begin{pmatrix} \mathbf{I}_1 & 0 \\ 0 & 0 \end{pmatrix}$, where $\mathbf{I}_1$ is the identity matrix of order pN.

4.3.1. Preconditioning based on operator-splitting

Preconditioners for matrices of the form (19) have been studied in [1]. Here we summarize some of the results and present the results of some experiments with parallel versions of these preconditioners. Let

$$\mathbf{A} = \frac{\partial \mathbf{S}}{\partial \mathbf{y}} = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} \qquad \mathbf{B} = \frac{\partial \mathbf{R}}{\partial \mathbf{y}} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

For problems which are dominated by the reaction, the matrix $\mathbf{J}$ can be approximated by $\mathbf{P}_R = \alpha \bar{\mathbf{I}}_1 - \frac{\partial \mathbf{R}}{\partial \mathbf{y}}$. The effectiveness of this preconditioner depends on the size of $B_{22}^{-1} A_{22}$. In particular, it is shown in [1] that

$$\mathbf{P}_R^{-1}\mathbf{J} - \mathbf{I} = \begin{pmatrix} 0 & 0 \\ 0 & B_{22}^{-1} A_{22} \end{pmatrix} + O(\epsilon)$$

where $\epsilon = \alpha^{-1}$. $\mathbf{P}_R$ is a block-diagonal matrix with blocks of sizes $s \times s$. The solution of linear systems involving $\mathbf{P}_R$ can be parallelized via a multiple-instance dense LU solver, as illustrated in the following Section.

For problems which are dominated by diffusion, the following matrix approximates $\mathbf{J}$:

$$\mathbf{P}_S = \alpha \bar{\mathbf{I}}_1 - \frac{\partial \mathbf{S}}{\partial \mathbf{y}}$$

It is shown in [1] that

$$\mathbf{P}_S^{-1}\mathbf{J} - \mathbf{I} = \begin{pmatrix} 0 & 0 \\ A_{22}^{-1} B_{21} & A_{22}^{-1} B_{22} \end{pmatrix} + O(\epsilon)$$

For most reaction–diffusion problems, the reaction terms dominate in one part of the problem and the diffusion terms dominate in the other part. In general, one can construct the operator-split preconditioners:

$$\mathbf{P}_{SR} = \left(\mathbf{I} - \alpha^{-1}\frac{\partial \mathbf{S}}{\partial \mathbf{y}}\right)\left(\alpha\bar{\mathbf{I}}_1 - \frac{\partial \mathbf{R}}{\partial \mathbf{y}}\right)$$

$$\mathbf{P}_{RS} = \left(\mathbf{I} - \alpha^{-1}\frac{\partial \mathbf{R}}{\partial \mathbf{y}}\right)\left(\alpha\mathbf{I}_1 - \frac{\partial \mathbf{S}}{\partial \mathbf{y}}\right)$$

based on $\mathbf{P}_R$ and $\mathbf{P}_S$. It is shown in [1] that:

$$\mathbf{P}_{SR}^{-1}\mathbf{J} - \mathbf{I} = \begin{pmatrix} 0 & 0 \\ 0 & B_{22}^{-1}A_{22} \end{pmatrix} + O(\epsilon)$$

$$\mathbf{P}_{RS}^{-1}\mathbf{J} - \mathbf{I} = \begin{pmatrix} 0 & 0 \\ A_{22}^{-1}B_{21} & A_{22}^{-1}B_{22} \end{pmatrix} + O(\epsilon)$$

and, in addition, if $\epsilon \min\{||\mathbf{A}_{11}||, ||\mathbf{B}||\} \leq 1$, then

$$\mathbf{P}_{RS}^{-1}\mathbf{J} - \mathbf{I} = \begin{pmatrix} 0 & 0 \\ 0 & A_{22}^{-1}B_{22} \end{pmatrix} + O(\epsilon)$$

4.3.2. Parallel preconditioners

There are several considerations which guide the choice of preconditioners in DASPK applications. These considerations include:

- DASPK employs a Newton method to compute an integration step, which is accomplished with GMRES.
- The GMRES method is considered most efficient when the subspace dimension is kept small.
- Often, the solution of the Newton system is not required to a very high accuracy.
- The best preconditioners are sometimes expensive and not necessarily the most parallel.

The multispecies food web problem was solved using the split operator approach outlined in the preceding Section, involving a reaction ($\mathbf{P}_R$) preconditioner and a diffusion ($\mathbf{P}_S$) preconditioner. The $\mathbf{P}_R$ preconditioner has a block-diagonal structure, with $\mathbf{M}^{nd}$ diagonal blocks of order s, where M is the mesh parameter, nd is the number of spatial dimensions and s is the number of species. The $\mathbf{P}_R$ preconditioner is applied by factoring the block-diagonal matrix using the TMC CMSSL multiple-instance routine, `gen_lu_factor` and solving with `gen_lu_solve`. The computational cost of the factorization is $O(M^{nd}s^3)$ floating-point operations.

The $\mathbf{P}_S$ preconditioner is also block-diagonal, with s blocks, where each block is the discrete Laplacian of order M^{nd}. A $\mathbf{P}_S$ preconditioner which is effective at holding down the size of the Krylov subspace and at the same time highly parallelizable is hard to find. Several iterative methods for applying the $\mathbf{P}_S$ preconditioner were evaluated, including Jacobi, Gauss–Seidel, polynomial (least-squares) and ADI. It was found that several Jacobi iterations usually give the best compromise between parallel efficiency and effectiveness as a preconditioner. Least-squares polynomial preconditioning is described briefly in Section 4.2.4.. Gauss–Seidel is a more effective preconditioner than Jacobi, but requires a back-substitution algorithm, which is not particularly efficient on

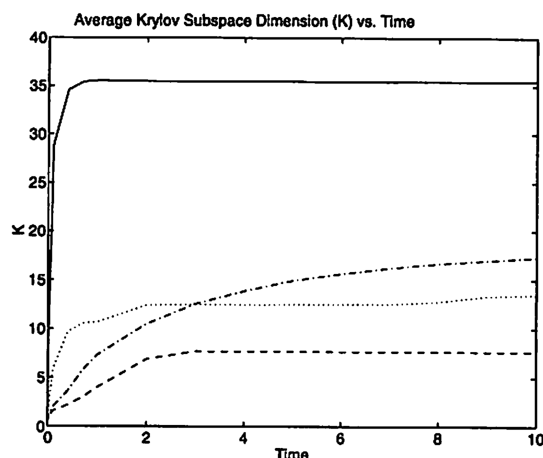

Figure 6. Effect of preconditioning on average Krylov subspace dimension (k); DAE 2-species reaction-diffusion problem, 64 × 64 *mesh, 32 PE. (—) no preconditioner, (-·-·) $\mathbf{P}_R$, (····) $\mathbf{P}_S$, (- - - -) $\mathbf{P}_{RS}$*

a massively parallel architecture. The same consideration affects the ADI method. The ADI method was implemented using the CMSSL routines `gen_tridiag_factor` and `gen_tridiag_solve_factored`. Although the ADI method converges more quickly than the other methods, it requires the solution of multiple-instance tridiagonal linear systems of order M. While the CMSSL tridiagonal solution algorithm is itself adapted for parallel implementation, it nonetheless involves a significant number of sequential steps. The performance of ADI does not appear competitive in the context of parallel preconditioning for linear systems.

The choice of preconditioners can dramatically affect the convergence of the iterative method. The problem instance involves two species on a 64 × 64 mesh, using $\beta = 100$ and `rtol` $= 10^{-5}$ and `atol` $= 10^{-7}$. Figure 6 illustrates the relative effectiveness of the $\mathbf{P}_R$ and $\mathbf{P}_S$ preconditioners in reducing the average Krylov subspace dimension. Neither the $\mathbf{P}_R$ nor the $\mathbf{P}_S$ preconditioner alone are sufficient to obtain fast convergence for the problem. However, the combination of the preconditioners is quite effective. For simpler preconditioners P_R and P_S, P_R is most effective early in the problem, and P_S is most effective later, when the problem is nearing a steady state and the time step lengthens. Overall, and not unexpectedly, the operator-split preconditioner is most effective in reducing the size of the Krylov subspace. However, it is also the most expensive.

The quality of the preconditioner can also influence the time step size. Figure 7 shows the step size obtained with different preconditioners. The problem instance involves two species on a 64 × 64 mesh, using $\beta = 1000$ and `rtol` $= 10^{-5}$ and `atol` $= 10^{-7}$. P_{RS} is a far more effective preconditioner for this problem than $\mathbf{P}_R$, $\mathbf{P}_S$ or no preconditioner. $\mathbf{P}_R$ and $\mathbf{P}_S$ preconditioners alone tend to restrict the step size, while the combination increases the step substantially.

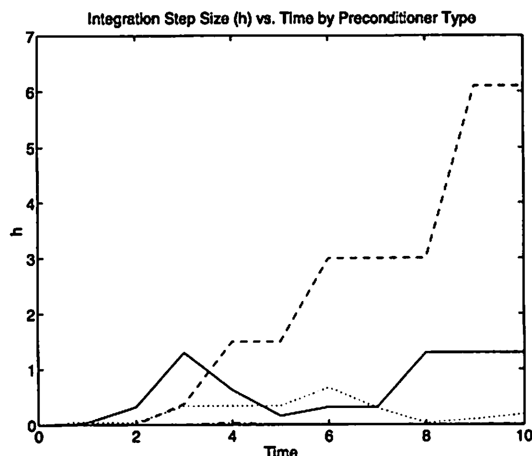

Figure 7. *Effect of preconditioning on integration step size; DAE 2-species reaction-diffusion problem, 64 × 64 mesh, 32 PE. (—) no preconditioner, (- · - ·) $\mathbf{P}_R$, (· · · ·) $\mathbf{P}_S$, (- - - -) $\mathbf{P}_{RS}$*

Table 6 provides additional detail on preconditioner performance, including the number of residual evaluations, the total time and the number of times the $\mathbf{P}_R$ matrix was factorized. The number of function evaluations is approximately the same as the number of times the preconditioner is actually applied to the solution of a linear system. Note that the $\mathbf{P}_R$ preconditioner alone requires the longest execution time, due to the expense of factorization. The combination $\mathbf{P}_{RS}$ preconditioner requires about one-third the time needed without a preconditioner. It is apparent that $\mathbf{P}_{RS}$ is the clear winner, and the reason is due to many fewer time steps, and hence fewer function and preconditioner evaluations, in the latter part of the problem.

Table 6. Summary of preconditioner performance. DAE 2-species reaction–diffusion problem, 64 × 64 mesh, 32 PE

Preconditioner	P_R factorizations	RES evaluations	Time (s)
RS	349	9174	477
N	0	49256	1431
S	0	19495	737
R	2080	47415	2462

4.3.3. *Parallel performance for nd = 2 with reshaping*

In this Section, results are presented for an $nd = 2$ dimensional implementation of the CM-5 multispecies code, using reshaping to convert from the residual routine data structure to the DASPKF90 vector.

Performance of the code using 32 processors was compared to a sequential implementation on a single-processor Cray Y/MP. The performance comparison involved a 14-species problem on a 60×60 mesh reported in [1]. Details of the comparison are summarized in Table 7 and show that the two implementations are comparable in performance. Both implementations used the same $\mathbf{R}_S$ preconditioner but used slightly different $\mathbf{P}_S$ preconditioners (Gauss–Seidel as against Jacobi). The Equivalence Library was not used for the $nd = 2$ multispecies problems. Instead, the array reshaping routines described in Section 4.1.3. were used. This imposes a severe cost on the code, in terms of communication load. In general, the Mflop rates for the problems reported in this Section are quite low, owing to the use of reshaping and the relatively small number of equations per processor.

Table 7. Comparison of DASPK on Cray Y/MP and CM-5. DAE 14-species reaction–diffusion problem, 60×60 mesh, 32 PE

Cray Y/MP	CM-5
DASPK (sequential)	DASPKF90 with reshaping
5 iterations, Gauss–Seidel	5 iterations, Jacobi
Single CPU	32 PE
471 s	515 s
215 integration steps	355 integration steps
Av. Krylov dim. = 2.8	Av. Krylov dim. = 1.9

The performance of the $nd = 2$ CM-5 multispecies code was also evaluated in terms of scaling behavior. Table 8 summarizes an experiment where the problem size M and number of processors were increased simultaneously so that the number equations *per processor* was held constant at 128. It was observed that:

- the number of linear iterations and integration steps increases with problem size, because mesh refinement increases the stiffness of the problem
- although the number of equations per processor is constant, total solution time increases slowly due to increasing stiffness.

In summary, the primary factors causing an increase in execution time are the number of linear iterations and preconditioner evaluations. A secondary factor involves the mapping of data structures onto processors. In general, the power-of-two array dimensions are most favorable for load-balancing. Note in Table 8 that the 181^2 mesh required 1245 seconds compared to 915 seconds for the 128^2 mesh, even though these two problem instances both used about the same number of linear iterations. The reason for the increase is that the mapping of the 181^2 mesh to processors is not ideal, compared to the mapping of the 128^2 mesh. It has been acknowledged by TMC that the CM-5 is most efficient when arrays are allocated with dimensions which are powers of two.

4.3.4. *Parallel performance for $nd = 3$*

An $nd = 3$ version of the multispecies code was developed, along with a customized version of DASPKF90, in which all data structures in DASPKF90 were rewritten to conform to

Table 8. Scaling of processors with problem size. DAE 10-species reaction–diffusion problem, $\beta = 1000$, P_{RS} preconditioner, problem size and number of processors scaled for 128 variables per processor

M	Total eqns.	PEs	Eqns. per PE	Integ. steps	Linear itns.	Prec. eval.	Time (s)
64	40960	32	128	345	1728	50	426
128	163840	128	128	596	3721	46	915
181	327610	256	128	701	3837	83	1245
256	655360	512	128	987	5141	137	1440

the four-dimensional array used in computing the residual and preconditioner (three spatial dimensions and a fourth dimension over the number of species). This implementation was designed to illustrate the performance of the code in the absence of data reshaping costs.

The main result is that the $nd = 3$ code has much better performance than the $nd = 2$ code with reshaping. For example, the $nd = 2$ code with reshaping used 341 s to solve a 64×64 2-species problem on 32 processors, while the $nd = 3$ code used 2256 s to solve a $64 \times 64 \times 64$ 2-species problem. Thus the $nd = 3$ version solved a $64\times$ larger problem in about $7\times$ the amount of time.

Fixed-size speed-up for the $nd = 3$ version was calculated for a problem of size $M = 32, s = 2$ with the $\mathbf{P}_{RS}$ preconditioner. Using the 32-PE partition as the baseline, speed-ups of 1.6, 1.9 and 2.0 were obtained on the 128, 256 and 512-processor partitions. The results show that the problem size $N = 65536$ is best suited to the 32-PE partition, since it requires only about twice the time used on the 512-PE partition. The loading factor of this problem is 2048 variables per processor on the 32-PE partition as against 128 variables per processor on the 512-PE partition.

The capacity of the code to solve large-scale reaction–diffusion equations was demonstrated by a problem of size $M = 128, s = 4$, with $N = 8388608$ solved on a 512-PE partition, using the $\mathbf{P}_{RS}$ preconditioner with three Jacobi iterations. The problem required 5.3 Gbytes of memory and 1.0 CPU hours. The processor loading was 16,384 variables per processor. Larger values of s can also be solved, although the memory requirements are large enough to create significant resource conflicts with other users on our time-sharing system.

4.4. Three-dimensional Cahn–Hilliard equation

The Cahn-Hilliard equation is of fourth order and parabolic type. It is used in modeling phase separation of alloys. The following formulation models phase separation dynamics in a binary alloy:

$$u_t = -\epsilon^2 \Delta^2 u + \Delta f(u), \qquad x \in \Omega \qquad t > 0 \tag{20}$$

where the variable $-1 \leq u(x,t) \leq 1$ represents the relative proportion of two pure materials, with $u = 1$ and $u = -1$ being the pure states and $u = 0$ a 50–50 mixture of the two materials. The material is assumed to be in a closed vessel represented by $\Omega \subset \mathbf{R}^3$ (the unit cube), with boundary Γ such that

$$\frac{\partial u}{\partial n} = \frac{\partial \Delta u}{\partial n} = 0 \qquad x \in \Gamma, \qquad t \geq 0. \tag{21}$$

The non-linear function $f(u)$ is a smooth potential defined in this case as $f(u) = u^3 - u$. The *interaction* parameter $0 < \epsilon \ll 1$ is related to the characteristic wavelength $O(\epsilon^{-1})$ of the solution. It can be shown [14] that this formulation is mass conservative and that the average concentration of the alloy is preserved.

Eyre [14] describes three distinct time scales associated with phase separation. The first is spinodal decomposition, with time scale $t = O(\epsilon^2)$, where a fine-grained structure appears. The second time scale is $t = O(1)$, where the material interfaces coarsen and separation becomes very slow. The third time scale is $t = O(e^{1/\epsilon})$, characterized by extremely slow coarsening of interfaces. The first two time scales pose the greatest challenge to a numerical method because of the rapid formation of numerous steep concentration gradients.

The spatial scale of the Cahn–Hilliard equation also introduces complications for numerical methods. The solution displays significant variation on an $O(\epsilon)$ length scale. To accurately capture the solution dynamics as ϵ is decreased, the spatial discretization must be reduced proportionately.

To solve equation (20), the domain Ω is discretized on a uniform Cartesian product mesh with M internal points in each direction, with spacing $h_x = h_y = h_z = 1/(M+1)$. The total number of unknowns in the formulation is $N = M^3$. The discrete Laplacian is approximated with central finite differences. The initial value function $g(x)$ is defined as

$$g(x) = \ rand(x)10^{-6} \qquad x \in \Omega \qquad t = 0, \tag{22}$$

where the function *rand* is a random number from the uniform distribution in [−0.5,0.5]. This initial value function describes a 50–50 alloy.

Initial computational experiments with DASPKF90 showed that it was not possible to solve the problem without a preconditioner for $M > 10$. Experiments with parallel preconditioners such as Jacobi and least-squares polynomials required an unacceptable number of linear and non-linear iterations during rapid phase separation.

An effective preconditioner was obtained by the operator splitting approach, using the FFT. Note that the Jacobian matrix corresponding to the DASPK formulation of equation (20) is given by

$$\mathbf{J}(u) = \alpha\mathbf{I} + \epsilon^2\Delta^2 - \Delta(f'(u)) \tag{23}$$

which may be approximated by the product

$$\alpha^{-1}\mathbf{J}(u) \simeq \mathbf{P}_S\mathbf{P}_R = (\mathbf{I} + \alpha^{-1}\epsilon^2\Delta^2)(\mathbf{I} - \alpha^{-1}\Delta f'(u)) \tag{24}$$

where $\mathbf{P}_S$ and $\mathbf{P}_R$ represent partial preconditioners for the linear and non-linear terms. The preconditioner $\mathbf{P}_S$ can be computed exactly by means of the FFT

$$\mathbf{P}_S = (\mathbf{I} + \alpha^{-1}\epsilon^2\Delta^2) = \mathbf{Q}(\mathbf{I} + \alpha^{-1}\epsilon^2\Lambda^2)\mathbf{Q}^T \tag{25}$$

where $\mathbf{Q}$,Λ are the eigenvectors and eigenvalues of the discrete Laplacian, respectively, and $\mathbf{Q}$,$\mathbf{Q}^T$ are applied by the FFT method. An effective and efficient preconditioner $\mathbf{P}_R$ has not yet been identified. Fortunately, the method works well using $\mathbf{P}_R = \mathbf{I}$, unless ϵ is very small, as illustrated in the following numerical results.

The Fortran 90 implementation of the Cahn–Hilliard equation used a programming 'trick' to avoid reshaping. For certain array dimensions (e.g. powers of two), it is possible to pass a three-dimensional array declared as `y(m,m,m)` into DASPKF90, which internally declares the array as `y(neq)`, where `neq = m*m*m`. The same result could be achieved by using the Equivalence Library, described earlier.

The performance of DASPKF90 on large-scale problems is illustrated in Table 9 for $M = 128$ ($N = 2{,}097{,}152$), and `rtol = atol` $= 10^{-4}$ and $\epsilon \in \{0.05, 0.025, 0.01\}$. Note that the number of linear iterations, average Krylov dimension k and time are higher for smaller values of ϵ, due to the increasingly oscillatory behavior of the solution and the need for a better preconditioner $\mathbf{P}_R$. For even smaller values of ϵ, the number of time steps grew very large as the solution became highly oscillatory. The total execution time on a 512-PE CM-5 partition is divided among the residual subroutine (RES), the preconditioner routine (FFT) and the integration routine (DASPK). The preconditioner routine dominates the execution time. The solution was evaluated by monitoring the mean value $\bar{\mathbf{u}}$ and the maximum norm, $\|\mathbf{u}\|_\infty$. The mean value of $\mathbf{u}$ was conserved to at least two decimal digits of precision in each simulation.

Table 9. Solution of Cahn–Hilliard equation with DASPKF90 for three values of ϵ. Problem size is $m = 128$ ($N = 2097152$), with tolerances `rtol = atol` $= 10^{-4}$, maximum time $t_{max} = 1$, solved on a 512-PE CM-5 partition

ϵ	Integ. steps	Linear itn.	Avg. k	Execution time (s) Total	RES	FFT	DASPK
0.050	30	19	0.4	16.5	1.3	13.8	1.0
0.025	30	60	1.3	25.8	2.0	21.7	1.6
0.010	47	546	7.5	144.4	9.1	124.6	10.2

5. CONCLUSIONS

Two parallel versions of DASPK have been introduced. DASPKF90 and DASPKMP are currently installed on the CM-5 and are routinely used for solving large-scale DAEs. The codes are available by contacting the authors. Currently, DASPKF90 runs about ten times faster than DASPKMP due to the fact that, with the current TMC software, DASPKMP is not able to utilize the vector processors. It is expected that future versions of the machine and/or software will address this problem and the performance of the codes will be more comparable. Also, if message-passing BLAS were available for CM-5 which could effectively make use of the vector processors, this would result in considerable speed-up for the message-passing code.

Reshaping is a problem in the development of data-parallel libraries for the CM-5. The equivalence library which TMC is now providing with its F90 compiler is a useful step, but too restrictive for many situations. We believe that the reshaping problem is a serious impediment to the development on the CM-5 of the kind of general purpose mathematical libraries which users have become accustomed to over the years. In other respects, the data-parallel mode was fast and relatively easy to use. In solving the three-dimensional

heat equation, the DASPKF90 subroutine ran at about 8 Gflops on the AHPCRC CM-5 512-PE partition (most of the flops were used in GMRES). However, the residual and preconditioner subroutines for the heat equation on a 256^3 mesh require a virtually one-to-one ratio of send/gets relative to arithmetic, and hence the sustained performance of DASPKF90, residual and preconditioner was only about 1–2 Gflops. This number is higher for more complicated systems of PDEs requiring more arithmetic per node, for example for systems of reaction–diffusion equations in three dimensions.

The 32-processor CM-5 was about comparable in runtime to a single-processor Cray Y/MP, for a 14-species reaction–diffusion problem with $\beta = 1000$ on a 60×60 mesh. DASPKF90 (with reshaping) and diffusion preconditioner using five iterations of Jacobi took 515 s and 355 integration steps, with average Krylov dimension 1.9. In contrast, DASPK on the Cray Y/MP with diffusion preconditioner using five iterations of Gauss-Seidel took 471 s, with 215 integration steps and an average Krylov dimension of 2.8.

For the multispecies reaction–diffusion problem, it was found that a split-operator preconditioner was very effective, and that the combination of diffusion and reaction preconditioners permitted the use of a fairly weak diffusion preconditioner (i.e. Jacobi). For the Cahn–Hilliard equation it was found that a powerful diffusion preconditioner (FFT) was necessary to overcome stiffness. A satisfactory preconditioner for the non-linear term in that equation was not identified and suggests the need for parallel preconditioners for sparse non-symmetric operators having the structure of the discrete Laplacian.

In general, the performance of DASPKF90 and DASPKMP will not scale linearly with problem size when the problem size is increased via mesh refinement. For many PDEs, where problem size is taken here to be the number of dependent variables in the DAE vector, the increasing stiffness of the problem will tend to increase the number of linear iterations and integration steps. Thus, although the speed of a linear iteration tends to scale with N, the overall time requirement will increase. The result is that scalability depends upon the problem instance as well as the architecture.

ACKNOWLEDGEMENTS

The work of the second author was partially supported by ARO contract number DAAL03-89-C-0038 with the University of Minnesota Army High Performance Computing Research Center, and by ARO contract number DAAL03-92-G-0247 and NIST contract number 60NANB2D1272, and by the Minnesota Supercomputer Institute. The work of the third author was partially supported by ARO contract number DAAL03-92-G-0247.

REFERENCES

1. P. N. Brown, A. C. Hindmarsh and L. R. Petzold, 'Using Krylov methods in the solution of large-scale differential-algebraic systems', *SIAM J. Sci. Comput.* **15**, 1467–1487 (1994).
2. R. S. Maier and L. R. Petzold, *User's Guide to DASPKMP and DASPKF90*, University of Minnesota AHPCRC report, 1993.
3. K. Brenan, S. Campbell and L. R. Petzold, *Numerical Solution of Initial–Value Problems in Differential-Algebraic Equations*, Elsevier, 1989.
4. C .W. Gear, 'Simultaneous numerical solution of differential/algebraic equations', *IEEE Trans.*, **CT–18**, (1), 89–95 (1971).
5. A. Skjellum, S. Mattisson, M. Morari and L. Peterson, 'Concurrent DASSL: Structure, application, and performance', *Proc. Fourth Conf. on Hypercube Concurrent Computers and Applications*, Monterrey, California, 1989.

6. Y. Saad and M. H. Schultz, 'GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems', *SIAM J. Sci. Stat. Comput.* **7**, 856–869 (1986).
7. P. N. Brown and A. C. Hindmarsh, 'Reduced storage matrix methods in stiff ODE systems', *J. Appl. Math. Comput.*, **31**, 40–91 (1989).
8. Y. Saad, 'Krylov subspace methods for solving large unsymmetric linear systems', *Math. Comput.* **37**, 105–126 (1981).
9. W. E. Arnoldi, '*The principle of minimized iterations in the solution of the matrix eigenvalue problem*', Q. *J. Appl. Math.*, **9** 17–29 (1951).
10. Homer Walker, 'Implementation of the GMRES method using Householder transformations', *SISC*, **9**, 152–163 (1988).
11. Thinking Machines Corporation, *CM Fortran Release Notes, Preliminary Documentation for Version 2.1 Beta 1*, Cambridge, Massachusetts, April 1993.
12. Y. Saad, 'Practical use of polynomial preconditionings for the Conjugate Gradient method', *SIAM J. Sci. Stat. Comput.*, **6**, 865–881 (1985).
13. P. N. Brown, 'Decay to uniform states in food webs', *SIAM J. Appl. Math.*, **46**, 376–392 (1986).
14. David J. Eyre, 'The dynamics of patterns for two phase separation equations', Ph.D. thesis, The University of Utah, 1992.