A parallel exponential integrator for large-scale discretizations of advection-diffusion models*

L. Bergamaschi[1], M. Caliari[2], A. Martínez[3], and M. Vianello[3]

[1] Department of Mathematical Methods and Models for Scientific Applications
University of Padua, Italy
[2] Department of Computer Science, University of Verona, Italy
[3] Department of Pure and Applied Mathematics, University of Padua, Italy

Abstract. We propose a parallel implementation of the ReLPM (Real Leja Points Method) for the exponential integration of large sparse systems of ODEs, generated by Finite Element discretizations of 3D advection-diffusion models. The performance of our parallel exponential integrator is compared with that of a parallelized Crank-Nicolson (CN) integrator, where the local linear solver is a parallel BiCGstab accelerated with the approximate inverse preconditioner FSAI. We developed message passing codes written in Fortran 90 and using the MPI standard. Results on SP5 and CLX machines show that the parallel efficiency raised by the two algorithms is comparable. ReLPM turns out to be from 3 to 5 times faster than CN in solving realistic advection-diffusion problems, depending on the number of processors employed.

1 Finite Element Discretization of the Advection-Diffusion Model

We consider the classical evolutionary advection-diffusion problem

$$\begin{cases} \dfrac{\partial c}{\partial t} = \operatorname{div}(D\nabla c) - \operatorname{div}(cv) + \phi \qquad x \in \Omega,\ t > 0 \\ c(x,0) = c_0(x),\ x \in \Omega; \\ c(x,t) = g_{\mathrm{D}}(x,t),\ x \in \Gamma_{\mathrm{D}};\ \langle D\nabla c(x,t), \nu\rangle = g_{\mathrm{N}}(x,t),\ x \in \Gamma_{\mathrm{N}}; \quad t > 0 \end{cases} \tag{1}$$

with mixed Dirichlet and Neumann boundary conditions on $\Gamma_{\mathrm{D}} \cup \Gamma_{\mathrm{N}} = \partial\Omega$, $\Omega \subset \mathbb{R}^3$. Equation (1) represents, e.g., a simplified model for solute transport in groundwater flow (advection-dispersion), where c is the solute concentration, D the hydrodynamic dispersion tensor, $D_{ij} = \alpha_{\mathrm{T}}|v|\delta_{ij} + (\alpha_{\mathrm{L}} - \alpha_{\mathrm{T}})v_i v_j/|v|$, $1 \le i,j \le d$, v the average linear velocity of groundwater flow and ϕ the source. The standard Galerkin Finite Element (FE) discretization of (1) with nodes $\{x_i\}_{i=1}^N$ and linear basis functions gives a large scale linear system of ODEs like

$$\begin{cases} P\dot{\mathbf{c}} = H\mathbf{c} + \mathbf{b}, \quad t > 0 \\ \mathbf{c}(0) = \mathbf{c}_0 \end{cases} \tag{2}$$

* Work supported by the research fellowship "Parallel implementations of exponential integrators for ODEs/PDEs" (co-ordinator M. Vianello, University of Padova).

where $\mathbf{c} = [c_1(t), \ldots, c_N(t)]^{\mathrm{T}}$, $\mathbf{c}_0 = [c_0(x_1), \ldots, c_0(x_N)]^{\mathrm{T}}$, P is the symmetric positive-definite mass matrix and H the (nonsymmetric) stiffness matrix. Boundary conditions are incorporated in the matrix formulation (2) in the standard ways.

2 Exponential Integration via Polynomial Approximation

In the sequel we consider stationary velocity, source and boundary conditions in (1), which give constant H and $\mathbf{b}$ in system (2), which is the discrete approximation of the PDE (1). As known, the solution can be written explicitly in the exponential form

$$\mathbf{c}(t) = \mathbf{c}_0 + t\varphi(tP^{-1}H)\left[P^{-1}H\mathbf{c}_0 + P^{-1}\mathbf{b}\right] \quad , \tag{3}$$

where $\varphi(z)$ is the entire function $\varphi(z) = (e^z - 1)/z$ if $z \neq 0,\ \varphi(0) = 1$.

Applying the well known mass-lumping technique (sum on the diagonal of all the row elements) to P, we obtain a diagonal mass matrix P_{L}. Now system (3) (with P_{L} replacing P) can be solved by the exact and explicit exponential time-marching scheme.

$$\begin{aligned} \mathbf{c}_{k+1} &= \mathbf{c}_k + \Delta t_k \varphi(\Delta t_k H_{\mathrm{L}})\mathbf{v}_k, \quad k = 0, 1, \ldots, \\ H_{\mathrm{L}} &= P_{\mathrm{L}}^{-1}H, \qquad \mathbf{v}_k = H_{\mathrm{L}}\mathbf{c}_k + P_{\mathrm{L}}^{-1}\mathbf{b}. \end{aligned} \tag{4}$$

Exactness of the exponential integrator (4) entails that the time-steps Δt_k can be chosen, at least in principle, arbitrarily large with no loss of accuracy, making it an appealing alternative to classical time-differencing integrators (cf. [6, 5]).

However, the practical application of (4) rests on the possibility of approximating efficiently the exponential propagator $\varphi(\Delta t H_{\mathrm{L}})\mathbf{v}$, where $\mathbf{v} \in \mathbb{R}^N$. To this aim, we adopt the Real Leja Points Method (shortly ReLPM), recently proposed in the framework of FD spatial discretization of advection-diffusion equations [5], and extended to FE in [2]. Given a matrix A and a vector $\mathbf{v}$, the ReLPM approximates the exponential propagator as $\varphi(A)\mathbf{v} \approx p_m(A)\mathbf{v}$, with $p_m(z)$ Newton interpolating polynomial of $\varphi(z)$

$$p_m(A) = \sum_{j=0}^{m} d_j \prod_{k=0}^{j-1} (A - \xi_k I), \quad m = 1, 2, \ldots \tag{5}$$

at a sequence of Leja points $\{\xi_k\}$ in a compact subset of the complex plane containing the spectrum (or the field of values) of the matrix A. Following [5, 2], an algorithm for the approximation of the advection-diffusion FE propagator $\varphi(\Delta t H_{\mathrm{L}})\mathbf{v}$ can be easily developed, by means of Newton interpolation at "spectral" Leja points. In the sequel, the compact subset used for estimating the spectrum of H_{L} in (4) will be an ellipse.

Algorithm ReLPM (Real Leja Points Method)

1. INPUT: H_{L}, $\mathbf{v}$, Δt, tol
2. Estimate the spectral focal interval $[\alpha, \beta]$ for $\Delta t H_{\mathrm{L}}$, by Gershgorin's theorem
3. Compute a sequence of Fast Leja Points $\{\xi_j\}$ in $[\alpha, \beta]$ as in [1]
4. $d_0 := \varphi(\xi_0)$, $\mathbf{w}_0 := \mathbf{v}$, $\mathbf{p}_0 := d_0 \mathbf{w}_0$, $m := 0$
5. WHILE $e_m^{\mathrm{Leja}} := |d_m| \cdot \|\mathbf{w}_m\|_2 > \mathrm{tol}$
   (a) $\mathbf{z} := H_{\mathrm{L}} \mathbf{w}_m$
   (b) $\mathbf{w}_{m+1} := \Delta t\, \mathbf{z} - \xi_m \mathbf{w}_m$
   (c) $m := m + 1$
   (d) compute the next divided difference d_m
   (e) $\mathbf{p}_m := \mathbf{p}_{m-1} + d_m \mathbf{w}_m$
6. OUTPUT: the vector $\mathbf{p}_m$: $\|\mathbf{p}_m - \varphi(\Delta t H_{\mathrm{L}})\mathbf{v}\|_2 \approx e_m^{\mathrm{Leja}} \leq \mathrm{tol}$

The ReLPM algorithm turns out to be quite simple and efficient. Indeed, being based on two-term vector recurrences in real arithmetic, its storage occupancy and computational cost are very small, already with one processor. For implementation details not reported here, we refer to [5].

2.1. Parallel ReLPM (Real Leja Points Method). A standard data-parallel implementation of ReLPM has been performed. The cost of computing the Leja points is negligible with respect to the rest of the algorithm ([1]) and hence we decided that every processor performs step 3 separately, without exchanging data. To perform an efficient parallel implementation of the ReLPM we choose to partition the matrix H_L by rows and the vectors involved in algorithm ReLPM consequently. In this way the `daxpy` operations in 4, 5b and 5e are performed without any communication among processors. Moreover, estimation of the focal interval (step 2) and computation of the 2-norm of a vector (to check the exit test) needs that the processors exchange only a scalar (the result of their local computation). The matrix vector product of step 5a requires the processors to communicate a number of elements of vector $\mathbf{w}_m$. We employed the parallel sparse matrix vector routine, successfully experimented in [4], which will be described in 3.3.

3 Parallel implementation of Crank-Nicolson

3.1. Crank-Nicolson (CN) Method. Crank-Nicolson (CN) is a robust method, widely used in engineering applications, and a sound baseline benchmark for any advection-diffusion solver. In the case of the relevant ODEs system (2) (with stationary $\mathbf{b}$), its variable step-size version writes as

$$\left(P - \frac{\Delta t_k}{2} H\right) \mathbf{u}_{k+1} = \left(P + \frac{\Delta t_k}{2} H\right) \mathbf{u}_k + \Delta t_k\, \mathbf{b}, \quad k = 0, 1, \ldots, \quad \mathbf{u}_0 = \mathbf{c}_0 \ , \tag{6}$$

where, for estimation of the local truncation error and step-size control, we have used standard finite-difference approximation of the third derivatives in $\|\dddot{\mathbf{c}}(t_k)\|_2 \Delta t_k^3 < 12\,\mathrm{tol}$.

The large and sparse linear system in (6) is solved the BiCGstab iterative method [9], preconditioned at each step, since the system matrix depends on Δt_k and hence varies from step to step. To accelerate the iterative solver, we consider the "approximate inverse preconditioners". They explicitly compute an approximation to A^{-1} and their application needs only matrix vector products, which are more effectively parallelized than solving two triangular systems, as in the ILU preconditioner. We selected the FSAI (Factorized Sparse Approximate Inverse) preconditioner proposed in [7], whose construction is more suited to parallelization than other approaches [3].

3.2. FSAI Preconditioning. Let A be a symmetric positive definite matrix (SPD) and $A = L_A L_A^T$ be its Cholesky factorization. The FSAI method gives an approximate inverse of A in the factorized form $H = G_L^T G_L$, where G_L is a sparse nonsingular lower triangular matrix that approximates L_A^{-1}. To construct G_L one must first prescribe a selected sparsity pattern $S_L \subseteq \{(i,j) : 1 \leq i \neq j \leq n\}$, such that $\{(i,j) : i < j\} \subseteq S_L$, then a lower triangular matrix $\hat{G}_L$ is computed by solving the equations $(\hat{G}_L A)_{ij} = \delta_{ij}$, $(i,j) \notin S_L$. The diagonal entries of $\hat{G}_L$ are all positive. Defining $D = [\mathrm{diag}(\hat{G}_L)]^{-1/2}$ and setting $G_L = D\hat{G}_L$, the preconditioned matrix $G_L A G_L^T$ is SPD and has diagonal entries all equal to 1.

The extension to the nonsymmetric case is straightforward; however the solvability of the local linear systems, and the nonsingularity of the approximate inverse, are only guaranteed if all the principal submatrix of A are non singular (which is the case, for instance, if $A + A^T$ is SPD). In the nonsymmetric case two preconditioner factors, G_L and G_U, must be computed. We limit ourselves to nonsymmetric matrices with a symmetric nonzero pattern (which is the common situation in matrices arising from FE discretization of PDEs), and set the sparsity patterns for G_U factor as $S_L = S_U^T$. The preconditioned matrix reads $D = G_L A G_U D^{-1}$, with $D = diag(G_L) = diag(G_U)$.

We set the sparsity patterns of the lower and upper triangular factors to allow nonzeros corresponding to nonzeros in the lower and upper triangular part of A^2, respectively. Next we perform a postfiltration step of the already constructed factors by using a small drop–tolerance parameter ϵ. The aim is to reduce the number of nonzero elements of the preconditioner, in order to decrease the arithmetic complexity of the iteration phase together with the communication complexity of multiplying the preconditioner by a vector.

For deeper implementation and performance details of parallel FSAI the author is referred to [3].

3.3. Efficient matrix-vector product. Following [4], we now briefly describe our implementation of the matrix-vector product, which is tailored for application to sparse matrices and minimizes data communication between processors. Within the ReLPM or CN algorithms, the vector $\mathbf{y} = B\mathbf{v}$ has to be calculated for $B = A, G_L, G_U$. Assume that the $N \times N$ matrix B is uniformly partitioned by rows among the p processors, so that $n \approx N/p$ rows are assigned to each processor. The same is done for the vector $\mathbf{v}$. The subset P^r containing the

nonzero elements belonging to processor r can be subdivided into two disjoint subsets $P_1^r = \{b_{ij} \in P^r,\ (i-1)n+1 \le j \le in\}$ and $P_2^r = P^r \backslash P_1^r$. Define the sets C_k^r, R_k^r of indices as: $C_k^r = \{j : b_{ij} \in P_2^r, k = ((j-1) \text{ div } n)+1\}$; $R_k^r = \{i : b_{ij} \in P_2^r, k = ((j-1) \text{ div } n)+1\}$. Processor r has in its local memory the elements of the vector $\mathbf{v}$ whose indices lie in the interval $[(r-1)n+1, rn]$. Before computing the matrix-vector product processor r: for every k such that $R_k^r \neq \emptyset$ sends to processor k the components of vector $\mathbf{v}$ whose indices belongs to R_k^r; gets from every processor k such that $C_k^r \neq \emptyset$, the elements of $\mathbf{v}$ whose indices are in C_k^r. At this point every processor is able to complete locally its part of the matrix-vector product.

4 Parallel experiments and results

4.1. Description of the test cases. We now discuss in detail two examples (cf. [2]), concerning FE discretizations of 3D advection-dispersion models like (1).

Example 1. The domain is $\Omega = [0,1] \times [0,0.5] \times [0,0.1]$, with a regular grid of $N = 161 \times 81 \times 41 = 534\,681$ nodes and $3\,072\,000$ tetrahedral elements. Here, $\phi \equiv 0$ and $c_0 \equiv 1$. Dirichlet boundary conditions are $c = 0$ on $\Gamma_{\mathrm{D}} = \{0\} \times [0.2, 0.3] \times [0,1]$, while the Neumann condition $\partial c/\partial \nu = 0$ is prescribed on $\Gamma_{\mathrm{N}} = \partial\Omega \setminus \Gamma_{\mathrm{D}}$. The velocity is $v = (v_1, v_2, v_3) = (1,0,0)$, the transmissivity coefficients are piecewise constant and vary by an order of magnitude depending on the elevation of the domain, $\alpha_{\mathrm{L}}(z) = \alpha_{\mathrm{T}}(z) \in \{0.0025, 0.025\}$.

Example 2. Same problem as of Example 1. However, the domain is discretized with a regular grid of $N = 161 \times 81 \times 161 \approx 2.1 \times 10^6$ nodes and about 12 millions of tetrahedral elements. Matrix of discretization H_L has roughly 2.1×10^6 rows and 3.1×10^7 nonzero elements.

In these examples the boundary conditions and vanishing sources lead to a zero steady state. The two integrators are employed on a time interval which produces a decrease of two orders of magnitude of the initial solution norm. While for CN the local time-step is selected adaptively, in order to guarantee a local error below the given tolerance, for the exponential integrator there is no restriction on the choice of Δt_k, since it is exact for autonomous linear systems of ODEs. To follow with some accuracy the evolution of the solution, we propose as in [5] to select the local time-step in (4) in such a way that the relative variation of the solution be smaller than a given percentage η, that is

$$\|\mathbf{c}_{k+1} - \mathbf{c}_k\|_2 \le \eta \cdot \|\mathbf{c}_k\|_2, \quad 0 < \eta < 1 \ . \tag{7}$$

If condition (7) is not satisfied, the time step Δt_k is halved and $\mathbf{c}_{k+1}$ recomputed; if it is satisfied with $\eta/2$ instead of η, the next time-step Δt_{k+1} is doubled. Clearly, smaller values of η allow better tracking of the solution.

4.2. Parallel programs and System's architecture The parallel programs are fortran 90 message passing codes, written using the MPI standard [8]. The

message passing programming model is a distributed memory model with explicit control parallelism. Message passing codes written in MPI are obviously portable and should transfer easily to clustered SMP systems, which are gradually becoming more prominent in the HPC market. We run the codes on two supercomputers located at the CINECA Supercomputer center of Bologna, Italy (`http://www.cineca.it`).

IBM SP5 supercomputer, an IBM SP cluster 1600, made of 64 nodes p5-575 interconnected with a pairs of connections to the Federation HPS (High Peformance Switch). Globally the machine has 512 IBM Power5 processors, capable of 4 double precision floating point operations per clock cycle, and 1.2 TBs of memory. Each microprocessor is supported by 36 MB of Level 3 cache. The peak performance of SP5 is 3.89 Tflops. Each p5-575 node contains 8 SMP processors POWER5 at 1.9 GHz, with 16GB of memory. The HPS switch is capable of a bandwidth of up to 2GB/s unidirectional.

IBM Linux Cluster (CLX), made of 512 2way IBM X335 nodes. Each computing node contains 2 Xeon Pentium IV processors. All the compute nodes have 2GB of memory (1GB per processor). Most processors of CLX are Xeon Pentium IV at 3.06 GHz with 512KB of L2 cache and the remaining ones, bought at the beginning of 2005, are Xeon Pentium IV EM64T at 3.00GHz with 1024KB of L2 cache. All the CLX processors are capable of 2 double precision floating point operations per cycle, using the INTEL SSE2 extensions. All the nodes are interconnected to each other through a Myrinet network (http://www.myricom.com), capable of a maximum bandwidth of 256MB/s between each pair of nodes. The global peak performance of CLX is of 6.1 TFlops. Parallel programming on the CLX is mainly based on the MPICH-GM version of MPI (myrinet enabled MPI).

4.3. Results concerning Example 1. We show in this section the timings of the two MPI codes when solving the problem described in Example 1. In the SP5 machine the running times were obtained by using the nodes in dedicated mode, hence reserving to our own use the entire node (8 processors) even to measure CPU times with 1,2, and 4 processors. We did not take any advantage of shared memory inside the node. In the CLX cluster only one of the two processors in each node was used, to optimize memory accesses performance.

CN has been run with variable stepsize, leading to 479 time steps to complete the simulation. To avoid the cost of constructing the FSAI preconditioner at each time-step, we chose to compute it selectively, depending on the variation of Δt_k. Besides, an improved preconditioning strategy (`mixed`) is proposed which consists in using Jacobi for the (well-conditioned) first steps, and FSAI for the remaining. The switch between the two accelerators takes place the next timestep after the solver first employs a number of iterations larger than a fixed value (40). BiCGstab iterations are stopped when the residual r_k satisfies $\|r_k\| \leq 10^{-4} \|b\|$. We report in Tables 1 and 2 the results of the codes running on the SP5 with a number o processors $p = 1, \cdots, 16$. As for CN the number of BiCGstab itera-

Table 1. Timings and speedups for Example 1 solved with CN with BiCGstab accelerated by diagonal and `mixed` preconditionings on the IBM SP5.

	Diagonal					`mixed` $\epsilon = 0.05$				
p	`iter`	T_p	T_{sol}	CPU	S_p	`iter`	T_p	T_{sol}	CPU	S_p
1	36224	19.8	5765.7	5872.2		14694	180.6	3870.7	4137.0	
2	36140	10.1	2911.36	2968.5	2.0	14653	94.3	1907.0	2048.4	2.0
4	36382	4.4	1040.18	1078.5	5.4	14696	46.9	813.4	878.0	4.7
8	36196	1.7	471.4	480.8	12.2	14689	23.7	409.6	440.4	9.4
16	36276	0.9	254.5	260.5	22.5	14665	12.6	202.6	226.5	18.3

tions (`iter`), and the CPU times for computing the preconditioner (T_p), for the iterative solver (T_{sol}) and the overall CPU time are given, whereas for ReLPM we provide the number of steps, the number of total inner iterations (`iter`) and the overall CPU time. The `mixed` preconditioning strategy results in a reduction of number of linear iterations and CPU time with respect to the diagonal preconditioner.

The ReLPM has been run using $\eta = 0.02$ and 0.05, with similar performances. Obviously, the value $\eta = 0.02$ allows a better tracking of the solution.

Table 2. Timings and speedups for Example 1 solved by parallel ReLPM on the IBM SP5.

	$\eta = 0.02$				$\eta = 0.05$			
p	Steps	`iter`	CPU	S_p	Steps	`iter`	CPU	S_p
1	239	17332	1282.2		98	18201	1343.3	
2	239	17332	634.9	2.0	98	18201	663.2	2.0
4	239	17332	228.9	5.6	98	18201	237.1	5.7
8	239	17332	108.9	11.8	98	18201	113.4	11.8
16	239	17332	58.2	22.1	98	18201	63.1	21.3

In Table 3 we report the summary of the results of the same runs on the CLX machine. Here, the speedup values are between 11 and 16 hence yielding a parallel efficiency of at least 70% on 16 processors. The timings results demonstrate

Table 3. Summary of results on the CLX machine.

	Crank Nicolson				ReLPM			
	Diagonal		`mixed`		$\eta = 0.02$		$\eta = 0.05$	
p	CPU	S_p	CPU	S_p	CPU	S_p	CPU	S_p
1	8771.9		6484.1		1627.1		1425.7	
16	810.2	10.8	552.8	11.7	99.7	16.3	119.2	11.9

that both codes scale well with increasing number of processors. Moreover, they show a superspeedup when using more than 2 processors due to cache effects, since only for $p \geq 4$ the local matrix resides entirely in the Level 3 cache when performing the matrix vector product.

4.4. Results concerning Example 2. As in the previous example, CN has been run with variable stepsize, leading to 479 time steps to complete the simulation. We used the `mixed` preconditioning strategy and set the limit number for Jacobi preconditioning to 60 iterations. The same exit test as in Example 1 was used.

Table 4. Timings and speedups for Example 2 solved with CN accelerated with the `mixed` preconditioner and ReLPM with $\eta = 0.02$ on the SP5. Symbol † stands for "out of memory". The speedups for CN have been computed as $S_p^* = 8 * T_8/T_p$.

	Crank Nicolson `mixed` $\epsilon = 0.1$					ReLPM $\eta = 0.02$			
p	`iter`	T_p	T_{sol}	CPU	S_p^*	steps	`iter`	CPU	Sp
1	†	†	†	†		238	17710	5308.2	
2	†	†	†	†	†	238	17710	2672.6	2.0
4	†	†	†	†	†	238	17710	1342.2	4.0
8	18050	62.9	2337.0	2453.7	8.0	238	17710	661.4	8.0
16	18006	31.8	1020.0	1073.8	18.3	238	17710	237.2	22.4
32	18105	16.6	433.5	457.8	42.9	238	17710	132.1	40.2
64	18086	10.4	250.2	267.9	73.3	238	17710	66.4	79.9

We report in Table 4 the timings concerning Example 2 on the SP5 with $p = 1, \cdots, 64$. As for CN, the code could not run for $p = 1, 2$ and 4. This is so because the limit of 1.667 GB of available memory of the SP5 nodes is not sufficient to hold the local system and preconditioner matrices when less than 8 processors are used. As in the previous example, both codes scale well with increasing number of processors again achieving a superspeedup for $p \geq 16$ due to cache effects. We recall that both examples rely on the same differential problem; however the discretization of Example 2 is made on a finer grid, which yields an algebraic problem roughly four times larger than that of Example 1. We note that CN is on the average four times slower than our ReLPM.

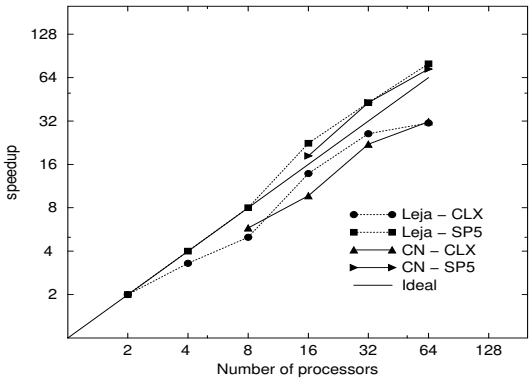

Fig. 1. Speedups vs p for ReLPM and CN on the SP5 and CLX supercomputers.

The summary of the performance results are reported in Figure 4. For ReLPM on the SP5 we obtain perfect speedups up to 8 processors. For $p > 8$ the curves of both CN and ReLPM are above that of the ideal speedup. Figure 4 demonstrates that the scaling of the codes in the CLX machine decreases when using more than 32 processors. We think that a potential source of this poor scaling in the CLX machine is the smaller bandwidth and higher latency of the interconnection network with respect to the HPS of the SP5 machine.

5 Conclusions

A parallel implementation of two algorithms for the solution of advection-diffusion equations on 3D domains has been proposed. Parallelization of ReLPM revealed almost straightforward, being based on matrix-vector products and not requiring linear system solutions. The CN solver has been carefully parallelized, with special emphasis on the selection of an efficient parallel preconditioner. Results on two supercomputers in the solution of a problem of more than 2 million unknowns show the very good scalability of the two codes, enhancing at the same time the efficiency of ReLPM both in terms of CPU time and memory occupancy.

References

1. J. Baglama, D. Calvetti, and L. Reichel. Fast Leja points. *Electron. Trans. Numer. Anal.*, 7:124–140, 1998.
2. L. Bergamaschi, M. Caliari, and M. Vianello. The ReLPM exponential integrator for FE discretizations of advection-diffusion equations. In M. Bubak, et al., editors, *ICCS 2004, Proceedings, Part IV, LNCS 3036*, pages 434–442. Springer, 2004.
3. L. Bergamaschi and A. Martínez. Parallel acceleration of Krylov solvers by factorized approximate inverse preconditioners. In M. Daydè et al., editor, *VECPAR 2004, LNCS 3402*, pages 623–636, Springer, 2005.
4. L. Bergamaschi and M. Putti. Efficient parallelization of preconditioned conjugate gradient schemes for matrices arising from discretizations of diffusion equations. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, March, 1999. (CD–ROM).
5. M. Caliari, M. Vianello, and L. Bergamaschi. Interpolating discrete advection-diffusion propagators at spectral Leja sequences. *J. Comput. Appl. Math.*, 172(1):79–99, 2004.
6. M. Hochbruck, C. Lubich, and H. Selhofer. Exponential integrators for large systems of differential equations. *SIAM J. Sci. Comput.*, 19(5):1552–1574, 1998.
7. L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings I. Theory. *SIAM J. Matrix Anal. Appl.*, 14:45–58, 1993.
8. MPI Forum. *MPI: A message passing interface standard*, 1995. also available online at `http://www.mpi-forum.org/`.
9. H. A. van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13(2):631–644, 1992.