Numerical applications with PVM on workstation clusters

Hartmut Schwandt, Annette Jaekel, Bogdan Pelc

Technische Universitaet Berlin, Dept. of Mathematics

In this paper we report some results of experiments with two numerical applications for various versions of PVM on homogeneous clusters of IBM workstations. It is intended to demonstrate the possibility of getting satisfactory speedups in this context for typical parallel numerical algorithms and to illustrate the effect of different networks (Ethernet, FDDI, serial optical links (SOCC)).

Configuration:

The following tests have been carried out on clusters of IBM RS 6000 workstations:

1) 1 × 590 (256 MB), 1 × 560 (256 MB), 2 × 550 (128 MB) for public domain PVM 3.3 on ethernet with TCP/IP and for PVM/6000 from IBM on FDDI and SOCC.
2) 11 × 220 (each with 16 MB) for PVM 3.3 on ethernet with TCP/IP.

The three networks are briefly characterized in the first table by a pingpong algorithm between a 560 and a 550. PVME denotes an older version from IBM based on PVM 2.3 customized for SOCC´s. Unless otherwise indicated we report real time in seconds (always the best of at least three runs) and we further use the AllowRouteDirect and RawData options in PVM 3.3 and PVM/6000.

		maximal bandwidth (Mbit/s)	measured latency (µs)	measured maximal bandwidth (MBit/s)
PVM 2.4 - vsnd/vrcv	Ethernet - TCP/IP	10	2000	8.4
PVM 3.3	Ethernet - TCP/IP	10	1500	8.4
PVM 3.3	SOCC - TCP/IP	n/a	1700	23.7
PVME	SOCC	220	1800	42.3
PVM/6000	FDDI	100	1100	48.9
PVM/6000	SOCC	220	570	70.9

A direct parallel solver for block tridiagonal linear system of equations:

We apply a parallel direct solver [Schw] which is based on the idea of using Schur complements to a linear system of equations $M\,x = y$, $M = (B_{j-1}, A_j, C_j)$ with block dimension q and matrix blocks A_j, B_j, C_j of size $m \times m$, i.e a system size of $N \times N$, $N = m*q$. Given p processors, we partition $q = (p - 1) * bloc + rest + p - 1$, i.e. we parallelize over the block dimension and obtain p partial systems $T_i = (B_{j-1}, A_j, C_j)$, $1 \le i \le p$, of dimension $bloc$ and $rest$, resp., and p - 1 coupling block equations. By a formal reordering, we move the coupling equations to the end and we obtain a system

$$\begin{pmatrix} T & F \\ G & H \end{pmatrix} \begin{pmatrix} x_p \\ x_c \end{pmatrix} = \begin{pmatrix} y_p \\ y_c \end{pmatrix}$$

where the block diagonal matrix $T = \mathrm{diag}_B(T_i)$ describes the partial systems with appropriate vectors of unknowns x_p and a righthand side y_p and where all block equations corresponding to the coupling equations are collected in a block tridiagonal matrix H with corresponding vectors x_c and y_c. F, G are block bidiagonal matrices which contain the last block B_j and the first block C_j succeeding to each partial system, F, G link both parts of the above system. The algorithm consists of three major steps:

1) compute the Schur complement of the above system with respect to H, i.e. compute $C := H - G\,T^{-1}\,F$, $w := y_c - G\,T^{-1}\,y_p$
2) solve the blocktridiagonal coupling system $C\,x_c = w$ for the unknowns of the coupling equations
3) solve for the unknowns of the partial systems $x_p = T^{-1}(y_p - F\,x_c)$.

By interpreting these steps blockwise, step 1 and 3 can be fully parallelized with respect to the block dimension q while the coupling system is principally a sequential problem. The whole algorithm is characterized by matrix and matrix vector operations and by block tridiagonal partial systems which are solved in this context by block Gaussian elimination. The advantage of this algorithm is due to the fact that all partial systems can be solved by an arbitrary solver which is suitable in this context and that the coupling system has only block dimension p - 1.

We use a host node model. The communication structure is similar to that of other block tridiagonal solvers. For simplicity, we assume *rest* = *bloc*.

1) Before step 1, each processor receives its partial matrix T_i from the host, i.e. three arrays A, B, C each of size *bloc* * m^2 and the corresponding part of the righthand side y of size *bloc* * m. In total, approx. 3 m N words have to be sent. This is not negligible, but it is tolerable in view of a total arithmetic complexity of $O(m^2 N)$.

2) Before step 2, each processor $P(i)$ except the last processor first sends $2m$ words to processor $P(i+1)$. Then he sends $3 m^2 + m$ words to the first processor which solves the coupling system.

3) After step 2, processor $P(1)$ sends the $(p\text{-}1) * m$ unknowns of the coupling equations to the host (running also on $P(1)$) and then 2 m words to each other processor $P(i)$, $1 \leq i \leq p$.

4) After step 3), each processor sends *bloc* * m unknowns to the host.

Obviously, the major part of the data volume has to be sent when initializing the coefficient matrices before starting the algorithm. It is higher by an order of magnitude than that in the algorithm itself. During the execution of the algorithm, relatively small data volumes have to be sent. The number and the size of the messages only depends on the number p of processors and of the block size m, but not on the block dimension q, i.e. not directly on the total problem size.

In the first example, we fix the problem size to q = 127, m = 100, i.e. N = 12 700 and we assume p = 3 processors (one 560, two 550). In this case, the influence of the sequential solution of coupling system should be negligible. In the next table we report the real time for various situations. In program version v1, each processor receives its data from the host as described above. The total time includes also data and process initialization. In version v2 data and process initialization are measured separately by introducing barriers: The total time only includes the time which is necessary for the execution of the algorithm provided all data have been previously distributed to the processors. In version v3 all processors initialize their data themselves. They only receive the necessary information on the array sizes.

				total real time (sec)	data initialization (sec)	process initialization (sec)
v1	PVM 2.4	Ethernet TCP/IP	snd/rcv	82.4	–	–
v1	PVM 2.4	Ethernet TCP/IP	vsnd/vrcv	65.9	–	–
v1	PVM 2.4	SOCC TCP/IP	vsnd/vrcv	55.2	–	–
v2	PVM 2.4	Ethernet TCP/IP	vsnd/vrcv	49.9	24.4	0.14
v2	PVM 2.4	SOCC TCP/IP	vsnd/vrcv	49.2	9.1	0.15
v2	PVME (2.3)	SOCC		49.1	3.6	0.50
v2	PVM 3.3	Ethernet TCP/IP	DataRaw, AllowDirect	50.0	31.8	0.37
v2	PVM/6000	FDDI	DataRaw	49.1	4.8	1.36
v2	PVM/6000	SOCC	DataRaw	48.9	3.8	1.13
v3	PVM 3.3	Ethernet TCP/IP	DataRaw, AllowDirect	51.3	–	–
v3	PVM/6000	FDDI	DataRaw	51.8	–	–
v3	PVM/6000	SOCC	DataRaw	52.0	–	–

As predicted, the communication under PVM during the execution of the algorithm is almost independent on the network and the specific PVM version in this example. The influence of a fast network is significant only in the data initialization when large arrays with the matrix coefficients have to be sent from the host to all nodes. The last row for version v3 shows, that the data initialization on the processors takes at most 1 - 2 sec, i.e. if the nodes initialize the coefficient matrices locally. This time would be difficult to reach by sending the data (i.e. v2) except on an extremely fast network. In addition, v3 needs significantly less storage as each processor keeps is own data while in v1 and v2 the host has to store all data. In the above and all subsequent examples, we observed that for PVM/6000 the first message requires an extremely large time (3 sec to 12 sec !). We therefore included a 0 Byte dummy message and substracted the corresponding time from the total run time.

In the second example we only consider version v3. The processors initialize their data and communication is independent of q and it is characterized by relatively few and short messages, i.e. we have neither a problem of latency nor one of bandwidth for sufficiently large system sizes. We choose m = 100 for p = 3,4 and various values for q. The host program and the solution of the coupling system are always carried out on the 560.

		configuration	$q = 31$	$q = 63$	$q = 127$	$q = 255$	$q = 511$
PVM 3.3	Ethernet-TCP/IP	2 × 550,1 × 560	13.3	25.2	51.3	101.2	204.1
PVM 3.3	SOCC-TCP/IP	2 × 550,1 × 560	13.5	25.2	51.3	101.7	204.4
PVM/6000	FDDI	2 × 550,1 × 560	13.8	25.7	51.8	101.8	n/a
PVM/6000	SOCC	2 × 550,1 × 560	13.8	25.6	52.0	101.9	203.6
PVM 3.3	Ethernet-TCP/IP	2 × 550,1 × 560, 1 × 590	10.5	20.5	40.1	81.4	162.2
PVM/6000	FDDI	2 × 550,1 × 560, 1 × 590	9.8	19.2	38.0	77.0	155.5

On three workstations times are, as expected, almost identical for TCP/IP on Ethernet and on SOCC´s. The addition of a fourth, significantly faster 590 yields a speedup which is identical to that one could expect from from a third 550, i.e. due to the given communication structure with forced synchronization around the solution of the coupling system, the real time is defined by the slower 550. The use of the 'data inplace' option instead of the default led in all cases to significantly higher times. A further improvement can be reached by a load balancing according to the different performance of the machines, in particular that of the 590. On the other hand, we used PVM to compute large examples which need the accumulated memory of all four machines (2 x 128 MB + 2 x 256 MB).

The next table shows the effect of scaling the problem with an increasing number of processors on the cluster of 11 RS 6000/220, each with 16 MB of main memory.

		m = 30			m = 50		
p	q	real time	seq. time	Speedup	real time	seq.time	Speedup
1	100		2.39			12.48	
2	201	8.05	9.20	1.14	30.59	48.78	1.59
3	302	9.15	15.93	1.74	37.70	84.63	2.24
4	403	9.62	22.55	2.34	38.62	120.06	3.11
5	504	12.21	29.40	2.41	38.61	156.78	4.06
6	605	12.78	36.16	2.82	40.58	194.67	4.80
7	706	12.40	42.65	3.44	40.56	230.04	5.67
8	807	12.51	49.53	3.96	44.91	266.88	5.94
9	908	13.76	56.39	4.10	46.84	303.90	6.49
10	1009	16.33	63.46	3.89	41.30	340.80	8.25
11	1110	15.06	70.27	4.67	44.80	377.70	8.43

The CPU times for sequential execution on one RS6000/220 have been estimated for larger values of q. This example shows that even on ethernet satisfactory results can be obtained. The real times strongly varied (more than 100%) from run to run despite an empty network. While for m = 30 the arithmetic complexity is not high enough to hide the overhead of communication and of PVM, the values for m = 50 already lead to a good scalability. When compared to the optimal sequential algorithm for m = 50, q = 1110 under the assumption that one 220 with more than 140 MB of main memory were available, one would obtain a maximal speed up of about 3.2.

An iterative parallel solver for five point difference methods for Poisson's equation:

We discretize a Dirichlet problem $\Delta u = f$, $u = g$ on a rectangle which results in a sparse block tridiagonal system with five diagonals. We get a system with 127 * 127 unknowns. We apply a parallel additive Schwarz procedure with minimal overlap to a domain decomposition in three and four strips, resp., of almost equal size [Pelc], [Rodr]. We illustrate the effect of varying the number of inner and outer iterations (in-it and out-it) on three and four processors and on different networks. In order to make the results comparable, we fix the total number of iterations always to exactly 10000. This enables us to compare the parallel and the sequential execution of the parallel algorithm. A direct comparison of optimal parallel and optimal sequential algorithms is not intended in this context due to varying iteration numbers for a given convergence criterion. We use again a host node model.

While the sequential execution is almost insensible to the variation of the number of domains or of the repartition of inner and outer iterations, the parallel execution times heavily depend on these parameters. This is due to the fact that after each outer iteration, all processes are synchronized as follows. Each processor sends one row of unknowns as approximate boundary values to its left and right neigbour (except for the first and last which have only one neighbour). Then each processor sends a ready message to the host which returns the message to continue to all processors. This results in a rather small data volume per message (twice 127 words per processor at each synchronization) which are exchanged between neighbours only. But all processors communicate also with the host. The number of messages depends directly on the number of outer iterations. Therefore, satisfactory speedups are obtained only for lower numbers of outer iterations for which the real times drastically drop. On the other hand, faster networks have a drastical effect for larger numbers of outer iterations while their influence obviously decreases with a decreasing communication. Note that we are faced mainly with a latency problem as the data volumes are very small. It should be noted that versions with a minimized number of outer iterations are not necessarily optimal algorithms in numerically realistical situations where the number of iterations is not fixed but depends on a convergence criterion. In the sequential case one usually interested to minimize the number of inner iterations in contrast to the parallel case.This problem exceeds the scope of the present paper.

	$p = 3$ (560,550, 550)			$p = 4$ (590,560, 550,550)		
	10000 out-it 1 in-it	1000 out-it 10 in-it	100 out-it 100 in-it	10000 out-it 1 in-it	1000 out-it 10 in-it	100 out-it 100 in-it
PVM3.3 - ethernet - TCP/IP	375.7	143.1	121.0	331.3	113.9	92.2
PVM3.3 - SOCC - TCP/IP	378.8	143.6	120.7	–	–	–
PVM/6000 - FDDI	223.7	139.5	123.1	n/a	n/a	n/a
PVM/6000 - SOCC	174.1	128.9	122.1	–	–	–
sequential (550)	365.0	350.3	355.8	379.2	364.1	362.2

We finally note that the sequential block Jacobi algorithm, i.e. without domain decomposition, required 334.3 sec on a 550 for $\omega = 1$ and 10000 iterates.

Conclusion:

We have presented results for two typical numerical applications with different communication requirements on two workstation clusters, one with a few high performance workstations equipped with different networks, the other with a larger number of smaller machines in an ethernet. In those situations in which only a few short messages have to be exchanged, even PVM on ethernet yields satisfactory speedups in various contexts. As one could expect, the effect of a very fast network like FDDI or SOCC becomes significant in the two examples with a measurable communication: initialization of large arrays by the host on all nodes at the start of the algorithm and very frequent communication and synchronization in an iterative method after a prescribed number of iterations.

References:

[Pelc] B. Pelc, Numerische und Analytische Untersuchungen zur Schwarz´schen Alternierenden Prozedur, Diplomarbeit, FB Mathematik, TU Berlin, 1992.

[Rodr] G. Rodrigue, Inner/Outer Iterative Methods and Numerical Schwarz Algorithms, Computing 2 (1985), 205 - 218.

[Schw] H. Schwandt, A parallel Schur complement method for the direct solution of blocktridiagonal systems of equations, in preparation.

Hartmut Schwandt
Technische Universitaet Berlin
Dept. of Mathematics, Sekr. MA 6-4
Strasse des 17. Juni 136
D-10623 Berlin, Germany

Tel.: (+49 30) 314 23495
FAX: (+49 30) 314 21110

e-mail: schwandt@math.tu-berlin.de