



## **An LPAR-customized MPI\_AllToAllV for the Materials Science code CASTEP**

M. Plummer and K. Refson<sup>†</sup>

CLRC Daresbury Laboratory, Daresbury, Warrington, Cheshire, WA4 4AD, UK

<sup>†</sup>CLRC Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX

### **Abstract**

We describe a breakdown of an MPI\_AllToAllV transformation into groups of processors such that MPI\_AllToAllV is only performed with 1 processor per group. The application to the HPCx machine, which has processors logically partitioned into groups (LPARs) with fast internal communication and slower external communication, is discussed. The effects of IBM's 'fast-track' procedure for dealing with short messages are noted. We show how current use of the method on 8-processor LPARs may be extended to HPCx phase 2 with 32-processor LPARs, as well as to more general commodity clusters of SMP many-processor nodes.

**This is a Technical Report from the HPCx Consortium.**

Report available from <http://www.hpcx.ac.uk/research/publications/HPCxTR0401.pdf>

**© UoE HPCx Ltd 2004**

Neither UoE HPCx Ltd nor its members separately accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

<b>1</b>	<b><i>Introduction</i></b>	<b>3</b>
<b>2</b>	<b><i>A brief summary of the customized MPI_AllToAllV</i></b>	<b>4</b>
<b>3</b>	<b><i>Performance</i></b>	<b>6</b>
<b>4</b>	<b><i>Acknowledgements</i></b>	<b>10</b>
<b>5</b>	<b><i>References</i></b>	<b>10</b>

## 1 Introduction

The phase 1 HPCx machine consists of logical partitions (LPARs) containing eight 1.3 GHz Power4 processors. Each LPAR operates as an SMP machine and OpenMP may be used for inter-processor communication to take advantage of this. The implementation of MPI used within an LPAR is also designed to take full advantage of the shared-memory configuration. The 'Colony' SP Switch2 interconnect between LPARs places a limit on communication speed with approximately 20 microseconds latency and 350 MBytes per second bandwidth. The phase 2 upgrade to HPCx (June 2004) will have LPARs containing thirty-two 1.7 GHz Power4 processors and the new 'Federation' interconnect between LPARs.

The work described here was inspired by a qualitative comparison of HPCx scaling performance of the materials science code Castep [1] and the quantum mechanical molecular dynamics simulation code CPMD [2]. The two codes are distinct entities but share the technique of expansion of electronic wavefunctions in plane waves with many associated transformations between real and reciprocal space during a direct minimization of the electronic energy [1, 2]. The codes use three-dimensional Fast Fourier Transforms (3d-FFTs) on a distributed grid, performed using all\_to\_all collective communication combined with serial 1d-FFTs. Castep uses a 'column' distribution of the grid to maximise load balancing throughout the program, resulting in two MPI\_AllToAllV calls per 3d-FFT. CPMD uses a 'plane' distribution of the grid which requires one all\_to\_all communication per 3d-FFT. The version of CPMD optimized for SMP clusters also utilizes OpenMP to replace the 'internal' part of the data redistribution within each LPAR so that the distributed all\_to\_all communication involves 1 or more processors per LPAR [3].

On HPCx phase 1 the Castep code scales relatively poorly compared to other scientific parallel applications [4]. Although the load balancing is extremely good the dominance of collective communication results in relatively poor performance on the 'fast processor with slow communication' machine. This is in contrast to Castep performance on 'fast communication' machines such as the Cray T3E. The SMP cluster version of CPMD, carefully optimized for IBM architecture, seems to scale relatively well [3]. The possibility of reducing the number of 3d-FFTs that take place in a Castep electronic minimization step by manipulation of algorithms has been investigated with some success [5]: sample results presented in section 3 include this optimization. In this report we describe how an MPI\_AllToAllV call may be split into 'internal' and 'external' parts to take full advantage of the fast implementation of MPI within an LPAR. We also discuss how the performance is affected by overall message size and by IBM's use of a fast-track procedure for 'short' messages [6] during production runs. Changing the definition of 'internal' processors to take full advantage of these considerations also prepares the ground for efficient use of the method on the phase 2 HPCx machine.

## 2 A brief summary of the customized MPI\_AllToAllV

We assume a processor manipulates data then rearranges it into  $N$  components of variable length in array *send\_buffer*. In the all-processor MPI\_AllToAllV we then

```
call MPI_AllToAllV ( &
send_buffer, out_num_data, out_node_data_start, complex_type, &
recv_buffer, in_num_data, in_node_data_start, complex_type, &
gv_communicator, error)
```

with integer arrays for the number of counts and starting positions in the (complex) send and receive buffers. The *gv\_communicator* links the  $N$  processors over which the grid is distributed.

We now introduce two new communicators, *lpar\_communicator* and *inter\_lpar\_communicator* with associated sizes *num\_in\_lpar* and *num\_lpars* respectively. In the simplest implementation they are related to *gv\_communicator* as follows (note that *id\_in\_gv\_group* has values from 1 to  $N$ ).

```
lpar_group_id = id_in_gv_group / num_in_lpar
id_in_lpar = MOD(id_in_gv_group, num_in_lpar)
if (id_in_lpar == 0) then
    id_in_lpar = num_in_lpar
    lpar_group_id = lpar_group_id - 1
end if
lpar_group_id = lpar_group_id + 1
```

```
call MPI_Comm_Split (gv_communicator, lpar_group_id-1, id_in_lpar-1,&
                    lpar_communicator, error)
call MPI_Comm_Split (gv_communicator, id_in_lpar-1, lpar_group_id-1,&
                    inter_lpar_communicator, error)
```

We are assuming for simplicity here that *num\_in\_lpar* is a factor of  $N$ . This condition may be relaxed to give *num\_in\_lpar* a different value for each *lpar\_communicator* if required. Similarly the definition of the 'master' processor in each *lpar\_communicator* may be generalized to take account of any non-uniformity of hardware architecture for the group of processors. The communicators are set during the initialization stage of the program, as are the relations between the original counting arrays and the new arrays introduced below. The actual MPI\_AllToAllV call is replaced by a sequence of the following form. First, arrays *bigbuf* and *bigbuf2* are allocated on processors with *id\_in\_lpar* = 1. Then we gather data across *lpar\_communicator*.

```
if (id_in_lpar == 1) then
    inum = gather_disps(num_in_lpar) + gather_counts(num_in_lpar)
    inum = MAX ( &
inum, (scatter_disps(num_in_lpar) + scatter_counts(num_in_lpar)))
    allocate(bigbuf(inum), bigbuf2(inum), stat=error)
    if (error.ne.MPI_success) then
```

```

        write(*,*) 'Error comms_transpose_exchange: allocate bigbuf failed.'
        call comms_abort()
    end if
end if

is_count = gather_counts(id_in_lpar)
call MPI_GatherV (send_buffer, is_count, complex_type, &
                bigbuf, gather_counts, gather_disps, complex_type, &
                0, lpar_communicator, error)

```

Following this the non-communicating processors in *lpar\_communicator* remain idle while the *inter\_lpar\_communicator* operations are performed.

```

if (id_in_lpar == 1) then

    do il = 1, num_in_lpar
        do i = 1, num_in_gv_group
            ist = out_start(i,il)
            ifin = out_fin(i,il)
            inum = ifin - ist
            istoryart = out_start2(i,il)
            bigbuf2(istoryart:istoryart+inum) = bigbuf(ist:ifin)
        end do
    end do

    call MPI_AllToAllV ( &
        bigbuf2, out_big_countp, out_big_dispp, complex_type, &
        bigbuf, in_big_countp, in_big_dispp, complex_type, &
        inter_lpar_communicator, error)
    if (error.ne.MPI_success) then
        write(*,*) &
            'Error comms_transpose_exchange: MPI_alltoallv failed.'
        call comms_abort()
    end if

    do il = 1, num_lpars
        do i = 1, num_in_lpar
            jstart = in_start(i,il)
            istoryart = in_start2(i,il)
            inum = in_data(i,il)
            bigbuf2(istoryart:istoryart+inum) = bigbuf(jstart:jstart+inum)
        end do
    end do

end if

```

Finally we scatter the data to the processors in *lpar\_communicator*.

```

is_count = scatter_counts(id_in_lpar)
call MPI_ScatterV (bigbuf2, scatter_counts, scatter_disps, complex_type, &
                recv_buffer, is_count, complex_type, &
                0, lpar_communicator, error)

```

The original MPI\_AllToAllV is now completed and we deallocate *bigbuf* and *bigbuf2*. The data in *recv\_buffer* is then reordered locally for the next stage of the 3d-FFT

process or the main program. The above code is written to be concise within the generality requirements of the Castep 3d-FFT, although further refinement may be introduced in the future. Assuming an efficient implementation of MPI within an LPAR, the only possible disadvantage with respect to use of OpenMP for the *lpar\_communicator* operations is the allocation of the large arrays *bigbuf* and *bigbuf2*. The current HPCx technical settings restrict each MPI task to the LPAR working memory divided by *tasks\_per\_node* with *tasks\_per\_node* the number of MPI tasks in the LPAR assigned to the program. With OpenMP used internally there will be fewer MPI tasks per LPAR and more memory available for the buffer arrays whereas the current procedure assigns an MPI task to each processor.

### 3 Performance

At first glance, the faster communication within an LPAR may suggest that the customized method will be much faster than the standard method, even taking into account the internal MPI\_GatherV/MPI\_ScatterV and buffer to buffer copying. In practice the customized method becomes faster when communication is latency restricted: we shall quantify this below. Following considerations of whether the communication is bandwidth or latency restricted, the other factor affecting the process is the fast-track treatment of 'short' messages [6,7]. Messages shorter than a baseline  $K$  are treated using a relaxed 'buffered send' protocol that effectively speeds them up. The baseline has a maximum value  $K = 64$  kBytes set using the environment variable `MP_EAGER_LIMIT = 65536`. Note that for development and debugging purposes `MP_EAGER_LIMIT = 0` should be used so that all MPI is treated strictly. The baseline  $K$  may be reduced internally by LoadLeveler as the total number of processors required for a job is increased.

We now consider the Castep MPI\_AllToAllV. For a grid array of dimension  $G = ngx * ngy * ngz$ , the approximate size of *send\_buffer* is  $S = G / N$  in units of 16 Bytes for double precision complex array elements. The approximate individual message size in the same units is  $G / (N*N)$ . For an 8-way LPAR the *bigbuf* arrays are of size  $S * 8$ , divided among  $N / 8$  processors to give an approximate message size  $G * 64 / (N*N)$ . For  $num\_in\_lpar = j$ , this generalizes to  $G * (j*j) / (N*N)$ . An advantage of the customized method is that the value of *num\_in\_lpar* may be chosen to give the best results. This will be important for phase 2 of HPCx for which choosing  $j = 32$  will cause problems with both long messages and large buffers.

If the message size in both methods is long enough for communication to be bandwidth dominated, ie if the 350 Mbytes per second message takes many times longer than the latency of 20 microseconds, then the customized method will take longer than the standard method. This is due to the increased message length in the customized method. A 20 microsecond message translates into a length of about 460 of our 16 Byte units. As message size decreases and latency considerations become paramount we may expect the customized method to become faster. However the crossover between the two regimes is affected by `MP_EAGER_LIMIT`. If the message size using the standard method is 'short' whereas the message size for the customized method is 'long' then the standard method may still be faster due to use of the fast-track

procedure. The buffered send has three times lower latency than the full synchronous send [7]. We note that  $K = 4096$  in units of 16 Bytes. Once message size is ‘short’ for both methods, however, the customized method with fewer but still efficiently manageable messages becomes faster.

We illustrate these ideas with three examples using the Castep code. In addition to *gv\_communicator* parallelism, Castep has a higher parallelism in which processors are first distributed among Brillouin zone sampling k-points [1]. Processors associated with each k-point form a *gv\_communicator*. Since different k-points require communication infrequently during the energy minimization process this form of parallelization is an efficient way of reducing  $N$  for the MPI\_AllToAllV calls. We consider energy minimization of bulk  $\text{Al}_2\text{O}_3$ . Example 1 uses a cell with 120 atoms and a grid size  $G = 576000$ . This calculation has 5 k-points [1] and the total number of processors used is divided into 5 groups which perform 3d-FFTs independently. Example 2 uses a cell with 270 atoms and a grid size  $G = 1296000$ . This calculation has 2 kpoints and the processors divide into 2 groups. Example 3 uses the cell and grid of example 1 but enforces an alternative parallelization technique without kpoint parallelism, ie with 1 group in which the grid is distributed across all the processors.

Tables 1–3 show the approximate message size for the three examples respectively for the standard method and the customized method with  $num\_in\_lpar = 2, 4$  and 8. Figures 1–3 show execution times for the respective examples. Figure 1 shows times for a full minimization with 24 iterative SCF cycles [1], figure 2 shows times for 9 iterative SCF cycles and figure 3 shows times for 8 iterative SCF cycles.

Figure 1: 120 atom cell, 5 k-points

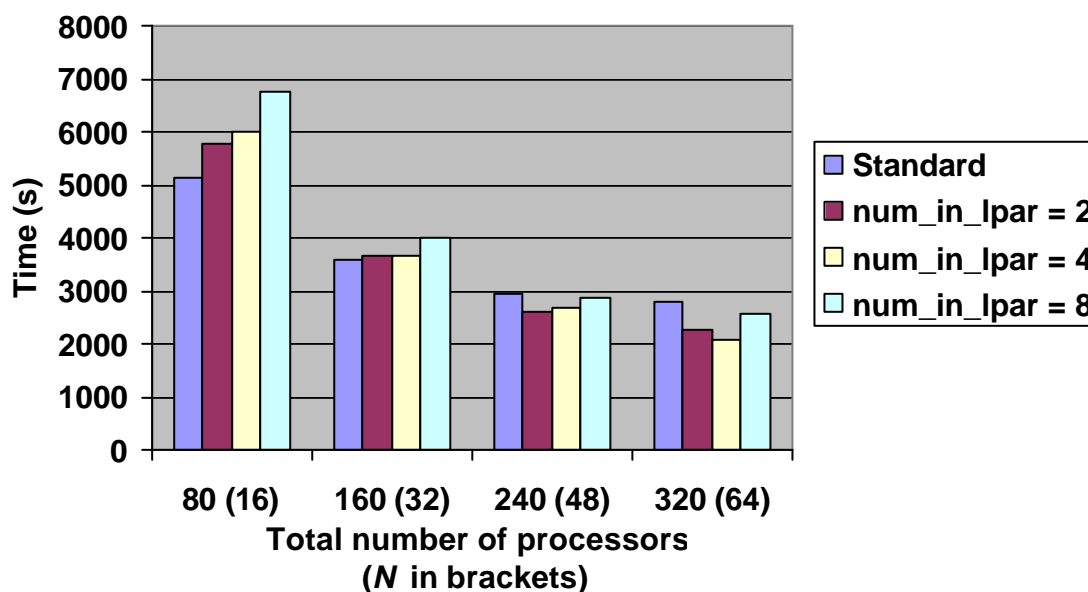


Table 1: approximate message sizes (16 Bytes) for  $G = 576000$   
 $num\_in\_lpar$

$N$	Standard	2	4	8
16	2300	9000	36000	140000
32	560	2300	9000	36000
48	250	1000	4000	16000
64	140	600	2300	9000

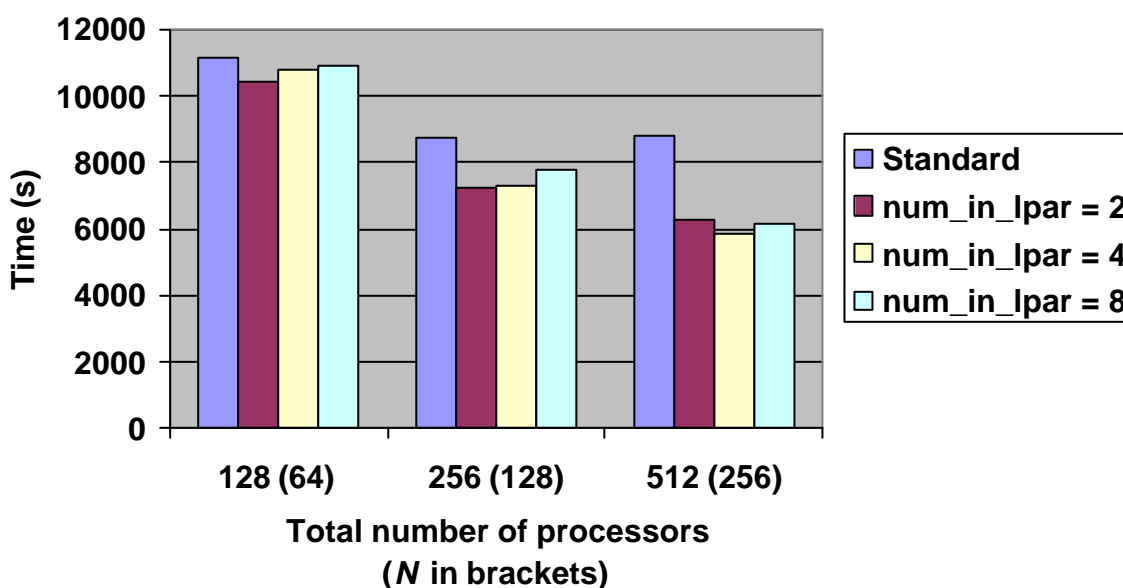
We see from the figures that a flexible value of  $num\_in\_lpar$  is important to make full use of the customized method in situations where  $num\_in\_lpar = 8$  is not improving performance.

For a fixed value of  $num\_in\_lpar$  the reduced performance for lower numbers of processors enhances the apparent scaling with processor numbers. Comparing the figures with the tables, we see that the customized methods begin to show advantages for approximate message size between 1000 and about 5000 units. With all message sizes below  $\sim 1000$  units the largest value of  $num\_in\_lpar$  should give the best results. This has been programmed into Castep on HPCx, with an option to read in either  $num\_in\_lpar$  or a threshold message size for more general use on clusters of SMP nodes.

**Table 2: approximate message sizes (16 Bytes) for  $G = 1296000$**

$N$	Standard	2	4	8
64	320	1300	5100	20000
128	80	320	1300	5100
256	20	80	320	1300

**Figure 2: 270 atom cell, 2 k-points**



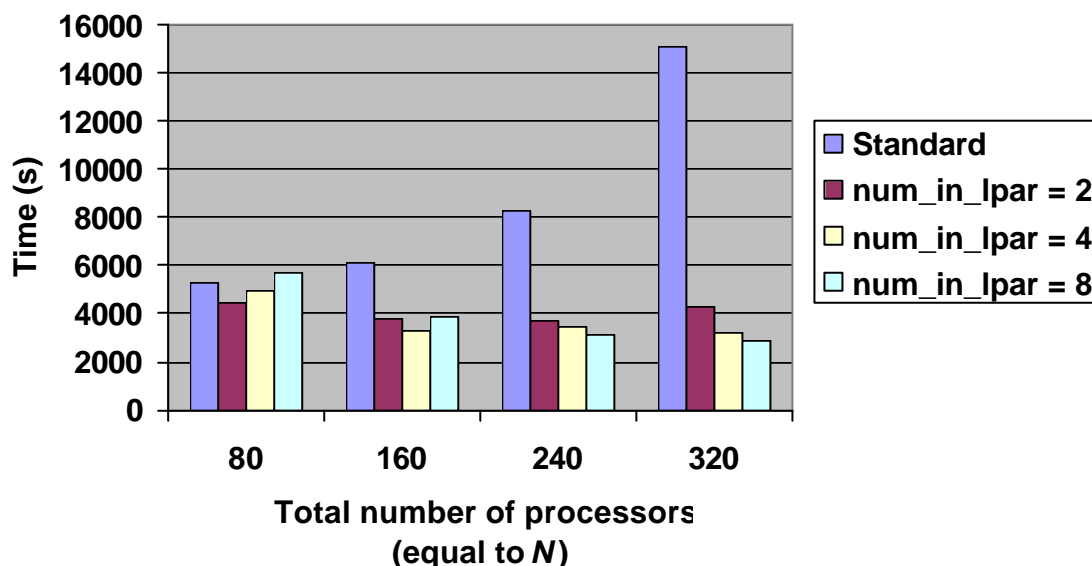
We note the following about figure 3. The data for figures 1 and 2 were collected while HPCx was fairly busy and represents reasonable timings, although we expect better

performance on an empty machine. The data for figure 3 were collected while HPCx was extremely busy with all (or nearly all) production processors in use. This situation has a detrimental effect on latency and thus collective communications. From this point of view codes which rely on MPI\_AllToAllV across large numbers of processors are prone to bad performance on systems such as HPCx. The use of the parallelization across k-points in Castep is thus very important as may be seen by comparing figures 1 and 3. For systems with only 1 k-point, as modelled by the enforced 1-group parallelism of example 3, the use of the customized MPI\_AllToAllV becomes crucial. Although we do not expect quite the same increase in performance relative to the standard method as is shown for  $N = 320$  in figure 3 for more favourable runs, we may point out that the customized method will be much less sensitive to this peak-period loss of performance.

**Table 3: approximate message sizes (16 Bytes) for  $G = 576000$**

$N$	Standard	$num\_in\_lpar$			
		2	4	8	
80	90	360	1400	5800	
160	20	90	360	1400	
240	10	40	160	640	
320	10	20	90	360	

**Figure 3: 120 atom cell, 5 k-points (1 group)**



HPCx phase 2 will have the Federation switch, faster processors and 32-way LPARs. Further investigations to optimize the customized MPI\_AllToAllV will be carried out when this system becomes operational [8]. Investigations of additional algorithmic optimizations [5] which may reduce the number of 3d-FFTs required per iterative cycle are also continuing. The customized MPI\_AllToAllV is also directly applicable to other more general cluster based systems with SMP many-processor nodes.

## 4 Acknowledgements

Very useful discussions are noted with Dr M Ashworth and Dr P Sherwood. This work was partly financed by EPSRC grants (GR/N02337/01, GR/S14658/01: the UKCP consortium).

## 5 References

- [1] M D Segall, P J D Lindan, M J Probert, C J Pickard, P J Hasnip, S J Clark and M C Payne, 'First-principles simulation: ideas, illustrations and the CASTEP code,' J Phys. Condensed Matter 14, 2717 (2002). Castep is distributed commercially by Accelrys Ltd (<http://www.accelrys.com> . UK academics may obtain the code and free licences from [ukcp@dl.ac.uk](mailto:ukcp@dl.ac.uk).
- [2] <http://www.cpmid.org> and references.
- [3] J Hutter and A Curioni, 'Dual-level Parallelism for ab-initio Molecular Dynamics: reaching Teraflop Performance with the CPMD code,' IBM Research Report RZ 3503 (2003), also submitted to Parallel Computing.
- [4] M Ashworth, I J Bush, M F Guest, M Plummer A G Sunderland; S Booth, D S Henty, L Smith and K Stratford, 'HPCx: a new resource for UK Computational Science' in Proceedings of the 17<sup>th</sup> Annual International Symposium on High Performance Computing Systems and Applications and the OSCAR Symposium, editor D Sénéchal (NRC Research Press, Canada 2003). This document contains benchmark tests for an earlier version of Castep (v4.2) which indicate general scaling properties.
- [5] M Plummer, K Refson and P J Hasnip, private communication.
- [6] IBM Parallel environment for AIX: MPI Programming Guide, chapter 11, available online(<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a2274221.pdf> ).
- [7] J Hein, S P Booth and M Bull, 'Exchanging multiple messages via MPI', technical report ([http://www.hpcx.ac.uk/research/hpc/technical\\_reports/HPCxTR0308.pdf](http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0308.pdf)).
- [8] The phase 2 HPCx machine will consist of 32-way LPARs built from four 8-processor units of the type which make up the phase 1 LPARs, connected with shared memory. The detail of this configuration may make particular selections of the processors which make up *inter\_lpar\_communicator* more advantageous than others.