Micro-Applications for Communication Data Access Patterns and MPI Datatypes

Timo Schneider[1], Robert Gerstenberger[1], and Torsten Hoefler[1,2]

[1] University of Illinois at Urbana-Champaign, IL, USA
{timos,gerro,htor}@illinois.edu
[2] Department of Computer Science, ETH Zurich, Switzerland
htor@inf.ethz.ch

Abstract. Data is often communicated from different locations in application memory and is commonly serialized (copied) to send buffers or from receive buffers. MPI datatypes are a way to avoid such intermediate copies and optimize communications, however, it is often unclear which implementation and optimization choices are most useful in practice. We extracted the send/recv-buffer access pattern of a representative set of scientific applications into micro-applications that isolate their data access patterns. We also observed that the buffer-access patterns in applications can be categorized into three different groups. Our micro-applications show that up to 90% of the total communication time can be spent with local serialization and we found significant performance discrepancies between state-of-the-art MPI implementations. Our micro-applications aim to provide a standard benchmark for MPI datatype implementations to guide optimizations similarly to SPEC CPU and the Livermore loops do for compiler optimizations.

1 Introduction

The MPI (Message Passing Interface) Standard [14] has become the de-facto standard to write distributed high-performance scientific applications. The advantage of MPI is that it enables a user to write performance-portable codes. This is achieved by abstraction: Instead of expressing a communication step as a set of point-to-point communications in a low-level communication API it can be expressed in an abstract and platform independent way. MPI implementers can tune the implementation of these abstract communication patterns for specific machines. MPI plays a similar role in the development of performance portable codes than high-level languages: Instead of coding a loop in inline assembly and using SIMD instructions the same loop can be expressed in a high-level language, using auto-vectorization features of the compiler. The programmer does not have to understand the details of the target platform and possible optimization techniques to write efficient application kernels.

MPI Derived Datatypes (DDTs), allow the specification of arbitrary data layouts in all places where MPI functions accept a datatype argument (i.e., MPI_INT). We give an an example for the usage of DDTs to send/receive a

vector of integers in Figure 1. All elements with even indices are to replaced by the received data, elements with odd indices are to be sent. Without the usage of MPI DDTs one would have to allocate temporary buffers and manually pack-/unpack the data. The usage of MPI DDTs greatly simplifies this example. If the used interconnect supports non-contiguous transfers (such as Cray Gemini [2]) the two copies can be avoided completely. Therefore the usage of DDTs not only simplifies the code but also can improve the performance due to the zero-copy formulation.

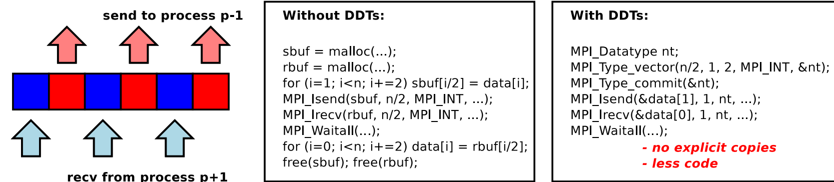


Fig. 1. An example use case for MPI derived datatypes

Not many scientific codes leverage MPI DDTs, even though their usage would be appropriate in many cases. One of the reasons might be that current MPI implementations in some cases still fail to deliver the expected performance, as shown by Gropp et al. in [9], even though a lot of work is done on improving DDT implementations [6, 18, 20]. Most of this work is guided by a small number of micro-benchmarks. This makes it hard to gauge the impact of a certain optimization on real scientific codes.

Coming back to the high-level language analogy made before and comparing this situation to the that of people developing new compiler optimizations techniques or microachitecture extensions we see that, unlike for other fields, there is no application derived set of benchmarks to evaluate MPI datatype implementations. Benchmark suites such as SPEC [8] or the Livermore Loops [13] are used by many (e.g., [1]) to evaluate compilers and microarchitectures. To address this issue, we developed a set of micro applications[3] that represent access patterns of representative scientific applications as optimized pack loops as well as MPI datatypes. Micro applications are, similarly to mini-applications [3, 5, 10], kernels that represent real production level codes. However, unlike mini-applications that represent whole kernels, micro-applications focus on one particular aspect (or "slice") of the application, for example the I/O, the communication pattern, the computational loop structure, or, as in our case, the communication data access pattern.

1.1 Related Work

Previous work in the area of MPI derived datatypes focuses on improving its performance, either by improving the way derived datatypes are represented in MPI or by using more cache efficient strategies for packing and unpacking the datatype to and from a contiguous buffer [6]. Interconnect features such

[3] which can be downloaded from http://unixer.de/research/datatypes/ddtbench

as RDMA Scatter/Gather operations [20] have also been considered. However, performance of current datatype implementations remains suboptimal and has not received as much attention as latency and bandwidth, probably due to the lack of a reasonable and simple benchmark. For example Gropp et al. found that several basic performance expectations are violated by MPI implementations in use today [9].

The performance of MPI Datatypes is often measured using micro-benchmarks such as those proposed by Reussner [16]. Several application studies demonstrate that MPI datatypes can outperform explicit packing in real-world application kernels [11, 12]. Those results are often either artificial (randomly chosen access patterns) or too complex to compare different implementations efficiently (part of a large application for which the performance is influenced by too many factors such as CPU speed). For example, many datatype optimization papers ignore the *unstructured access* class that we identify in this work completely even though this access pattern is found in many molecular dynamics and finite element codes.

However, the issue of preparing the communication buffer has received very little attention compared to tuning the communication itself. In this work, we show that the serialization parts of the communication can take a share of up to 90% of the total communication overheads because they happen at the sender *and* at the receiver.

Our micro-applications offer three important features: (1) they represent a comprehensive set of application use cases, (2) they are easy to compile and use on different architectures, and (3) they isolate the data access and communication performance parts and thus enable the direct comparison of different systems. They can be used as benchmarks for tuning MPI implementations as well as for hardware/software co-design of future (e.g., exascale) network hardware that supports scatter/gather access.

2 Representative Communication Data Access Patterns

We analyzed many parallel applications, miniapps and application benchmarks for their local access patterns to send and receive memory. Our analysis covers the domains of atmospheric sciences, quantum chromodynamics, molecular dynamics, material science, geophysical science, and fluid dynamics. We created 7 micro apps to span all application areas. Table 1 provides an overview of investigated application classes, their test cases, and a short description of the respective data access patterns. In detail, we analyzed the complex applications WRF [17], SPECFEM3D_GLOBE [7], MILC [4] and LAMMPS [15], representing the fields of weather simulation, seismic wave propagation, quantum chromodynamics and molecular dynamics. We also included existing parallel computing benchmarks and mini-apps, such as the NAS [19], the Sequoia benchmarks as well as the Mantevo mini apps [10].

We found that MPI derived datatypes (DDTs) are rarely used and thus we analyzed the data access patterns of the (pack and unpack) loops that are used

Application Class	Testname	Access Pattern
Atmospheric Science	WRF_x_vec WRF_y_vec WRF_x_sa WRF_y_sa	struct of 2D/3D/4D face exchanges in different directions (x,y), using different (semantically equivalent) datatypes: nested vectors (_vec) and subarrays (_sa)
Quantum Chromodynamics	MILC_su3_zd	4D face exchange, z direction, nested vectors
Fluid Dynamics	NAS_MG_x NAS_MG_y NAS_MG_z	3D face exchange in each direction (x,y,z) with vectors (y,z) and nested vectors (x)
	NAS_LU_x NAS_LU_y	2D face exchange in x direction (contiguous) and y direction (vector)
Matrix Transpose	FFT	2D FFT, different vector types on send/recv side
	SPECFEM3D_mt	3D matrix transpose,
Molecular Dynamics	LAMMPS_full LAMMPS_atomic	unstructured exchange of different particle types (full/atomic), indexed datatypes
Geophysical Science	SPECFEM3D_oc SPECFEM3D_cm	unstructured exchange of acceleration data for different earth layers, indexed datatypes

Table 1. Overview of the Application Areas, Represented Scientific Applications, and Test Names for our Micro-Applications.

to (de-)serialize data for sending and receiving. Interestingly, the data access patterns of all those applications can be categorized into three classes: *Cartesian Face Exchange*, *Unstructured Access* and *Interleaved Data*.

In the following we will describe each of the three classes in detail and give specific examples of codes that fit each category.

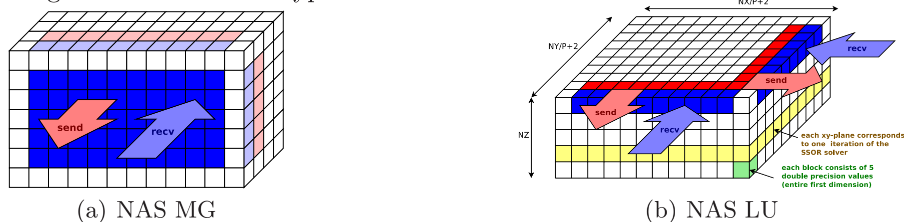
2.1 Face Exchange for n-dimensional Cartesian Grids

Many applications store their working set in n-dimensional arrays that are distributed across one or more dimensions. In a communication face, neighboring processes then exchange the "sides" of "faces" of their part of the working set. For this class of codes, it is possible to construct matching MPI DDTs using the subarray datatype or nested vectors. Some codes in this class, such as WRF, exchange faces of more than one array in each communication step. This can be done with MPI DDTs using a struct datatype to combine the sub-datatypes that each represents a single array.

The **Weather Research and Forecasting (WRF)** application uses a regular three-dimensional Cartesian grid to represent the atmosphere. Topographical land information and observational data are used to define initial conditions of forecasting simulations. WRF employs data decompositions in the two horizontal dimensions only. WRF does not store all information in a single data structure, therefore the halo exchange is performed for a number of similar arrays. The slices of these arrays that have to be communicated are packed into a single buffer. We create a struct of hvectors of vector datatypes or a struct of subarrays datatypes for the WRF tests, which are named WRF_{x,y}_{sa,vec}, one test for each direction, and each datatype choice (nested vectors or subarrays).

NAS MG communicates the faces of a 3d array in a 3d stencil where each process has six neighbors. The data access pattern for one direction is visualized

in Figure 2(a). The pack function in MG could be replaced by constructing an appropriate subarray datatype or using vector datatypes. Our NAS_MG micro-app has one test for the exchange in each of the three directions NAS_MG_{x,y,z} using nested vector datatypes.



(a) NAS MG (b) NAS LU

Fig. 2. Data Layout of the NAS LU and MG benchmark

The **NAS LU** application benchmark solves a three dimensional system of equations resulting from an unfactored implicit finite-difference discretization of the Navier-Stokes equations. In the dominant communication function, LU exchanges faces of a four-dimensional array. The first dimension of this array is of fixed size (5). The second (nx) and third (ny) dimension depend on the problem size and are distributed among a quadratic processor grid. The fourth (nz) dimension is equal to the third dimension of the problem size. Figure 2(b) visualizes the data layout. Our NAS_LU micro-app represents the communication in each of the two directions NAS_LU_{x,y}.

The **MIMD Lattice Computation (MILC)** Collaboration studies Quantum Chromodynamics (QCD), the theory of strong interaction, a fundamental force describing the interactions of quarks and gluons. The MILC code is publicly available for the study of lattice QCD. The su3_rmd application from that code suite is part of SPEC CPU2006 and SPEC MPI. Here we focus on the CG solver in su3_rmd. Lattice QCD represents space-time as a four-dimensional regular grid of points. The code is parallelized using domain decomposition and must be able to communicate with neighboring processes that contain off-node neighbors of the points in its local domain. MILC uses 48 different MPI DDTs [11] to accomplish its halo exchange in the 4 directions. The MILC_su3_zd micro-app performs the communication done for the $-z$ direction.

An important observation we made from constructing datatypes for the applications in the face-exchange class is that the performance of the resulting datatype heavily depends on the data-layout of the underlying array. For example, if the exchanged face is contiguous in memory (e.g., for some directions in WRF and MG), using datatypes can essentially eliminate the packing overhead completely. That is the reason we included tests for each direction applicable.

2.2 Exchange of Unstructured Elements

The codes in this class maintain some form of scatter-gather lists which hold the indices of elements to be communicated. Molecular Dynamics applications (e.g., LAMMPS) simulate the interaction of particles. Particles are often distributed based on their spatial location and particles close to boundaries need to be

communicated to neighboring processes. Since particles move over the course of the simulation each process keeps a vector of indices of local particles that need to be communicated in the next communication step. This access pattern can be captured by an indexed datatype. A similar access pattern occurs in Finite Element Method (FEM) codes (i.e., Mantevo MiniFE/HPCCG) and the Seismic Element Method (SEM) codes such as SPECFEM3D_GLOBE. Here each process keeps a mapping of mesh points in the local mesh defining an element and the global mesh. Before the simulation can advance in time the contributions from all elements which share a common global grid point need to be taken into account.

LAMMPS is a molecular dynamics simulation framework which is capable of simulating many different kinds of particles (i.e., atoms, molecules, polymers, etc.) and the forces between them. Similar to other molecular dynamics codes it uses a spatial decomposition approach for parallelization. Particles are moving during the simulation and may have to be communicated if they cross a process boundary. The properties of local particles are stored in vectors and the indices of the particles that have to be exchanged are not known a priori. Thus, we use an indexed datatype to represent this access. We created two tests, LAMMPS_{full,atomic}, that differ in the number of properties associated with each particle.

SPECFEM3D_GLOBE is a spectral-element application that allows the simulation of global seismic wave propagation through high resolution earth models. It is used on some of the biggest HPC systems available [7]. Grid points that lie on the sides, edges or corners of an element are shared between neighboring elements. The contribution for each global grid point needs to be collected, potentially from neighboring processes. Our micro-app representing SPECFEM3D has two tests, SPECFEM3D_{oc,cm}, which differ in the amount of data communicated per index.

Our results show that current derived datatype implementations are often unable to improve such unstructured access over packing loops. Furthermore, the overheads of creating datatypes for this kind of access (indexed datatypes) are high.

2.3 Interleaved Data or Transpose

Fast Fourier Transforms (FFTs) are used in many scientific applications and are among the most important algorithms in use today. For example, a two-dimensional FFT can be computed by performing 1d-FFTs along both dimensions. If the input matrix is distributed among MPI processes along the first dimension, each process can compute one a 1d-FFT without communication. After this step the matrix has to be redistributed, such that each process now holds complete vectors of the other dimension, which effectively transposes the distributed matrix. After the second 1d-FFT has been computed locally the matrix is transposed again to regain the original data

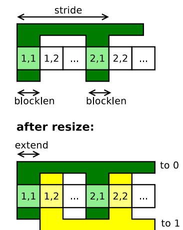


Fig. 3. Datatype for 2D-FFT

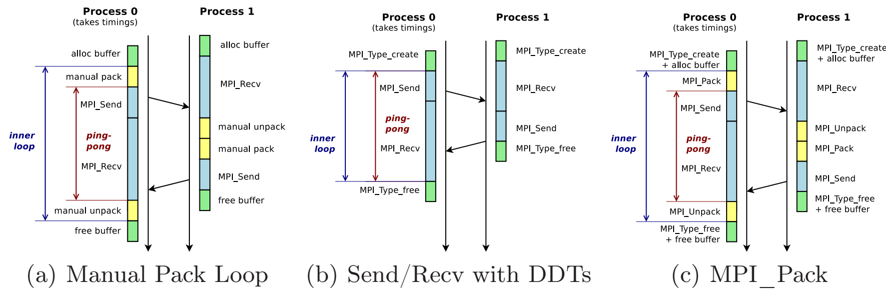
layout. In MPI the matrix transpose is naturally done with an MPI_Alltoall operation. Hoefler and Gottlieb presented a zero-copy implementation of a 2d-FFT using MPI DDTs to eliminate the pack and unpack loops in [11] and demonstrated performance improvements up to a factor of 1.5 over manual packing. The FFT micro-app captures the communication behavior of a two-dimensional FFT.

SPECFEM3D_GLOBE exhibits a similar pattern, which is used to transpose a distributed 3D array. We used Fortran's COMPLEX datatype as the base datatype for the FFT case in our benchmark and a single precision floating point value for the SPECFEM3D_MT case. The MPI DDTs used in those cases are vectors of the base datatypes where the stride is the matrix size in one dimension. To interleave the data this type is resized to the size of one base datatype. An example for this technique is given in Figure 3.

3 Micro-Applications for Benchmarking MPI Datatypes

We implemented all data access schemes that we discussed above as micro applications with the various tests. For this, we use the original data layout and pack loops whenever possible to retain the access pattern of the applications. We also choose array sizes that are representing real input cases. The micro-applications are implemented in Fortran (the language of most presented applications) and compiled with highest optimization.

We then perform a ping-pong-like benchmark between two hosts using MPI_Send() and MPI_Recv() utilizing the original pack loop and our datatype as shown in Figure 4. We also perform packing with MPI using MPI_Pack() and MPI_Unpack(), cf. Figure 4(c). For comparison we also perform a traditional ping-pong of the same data size as the MPI DDTs type size.



(a) Manual Pack Loop (b) Send/Recv with DDTs (c) MPI_Pack

Fig. 4. Measurement Loops for the Micro-Applications

The procedure runs two nested loops: the outer loop creates a new datatype in each iteration and measures the overhead incurred by type creation and commit; the inner loop uses the committed datatype a configurable number of times. Time for each phase (rectangles in Figure 4) is recorded to a result file and is analyzed with statistical software packages such as GNU R, for which we provide some example scripts. Measurements are done only on the client side, so the benchmark does not depend on synchronized clocks.

Let t_{pp} be the time for a round-trip including all packing operations (implicit or explicit) and t_{pack} the time to perform explicit packing (manual loop or pack-/unpack). In the DDT case is $t_{pack} = 0$. The network communication part can then be expressed as $t_{net} = t_{pp} - t_{pack}$ and is equivalent to a traditional normal ping-pong result. The overhead for packing relative to the communication time can be expressed as $ovh = \frac{t_{pp}-t_{net}}{t_{pp}}$.

The serial communication time t_{net} was practically identical for the tested MPI implementations ($< 5\%$ variation). This enables us to plot the relative overheads for different libraries into a single diagram for a direct comparison. Figure 5 shows those relative pack overheads for some representative micro-application tests performed with Open MPI 1.6 as well as MVAPICH 1.8 on a cluster with AMD Opteron 270 HE dual core CPUs and an SDR Infiniband interconnect; we always ran one process per node to isolate the off-node communication. We observe that the datatype engine of Open MPI performs better than MVAPICH's implementation. We also see that the dimensions/direction in which face exchanges occur have a significant impact on their performance (cf. WRF tests where the y direction has a much smaller packing overhead). This can be explained if we consider the memory layout of the underlying array - for some dimensions contiguous "strips" of data can be sent, while for others each element to be sent has a large stride. The SPECFEM3D tests show that unordered accesses with indexed datatypes are not implemented efficiently by both Open MPI and MVAPICH. This benchmark shows the importance of optimizing communication memory accesses: up to 80% of the communication time of the WRF_x_vec test case are spend with packing/unpacking data, which can be reduced to 70% with MPI DDTs. In the NAS_LU_x case, which sends a contiguous buffer, using MPI DDTs reduce the packing overhead from 40% to 15%.

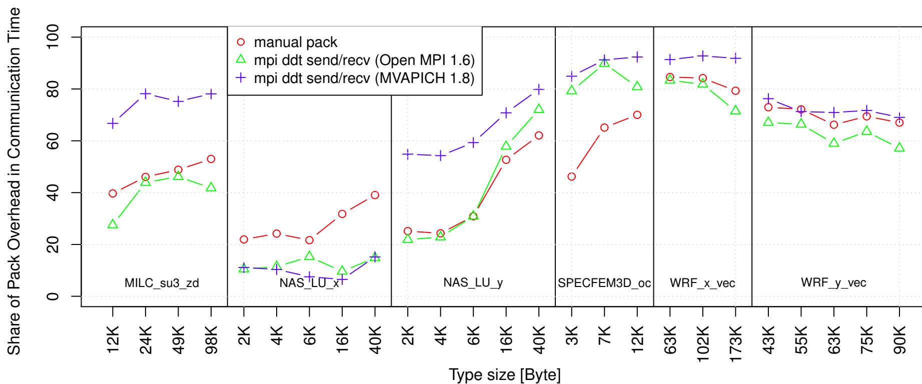


Fig. 5. Packing overheads (relative to communication time) for different micro-apps and datasizes

Note that the overhead for the creation of the datatype was not included in the calculations of the packing overheads in Figure 5. We show the creation

overheads and the absolute times for small number of tests in Figure 6 (the available space does not allow for presenting all collected results). We plot t_{net} as the communication time for the manual packing case. We note that the explicit packing numbers in the plot were doubled for a comparison with DDTs because DDTs implicitly pack at the sender and unpack at the receiver. Our results

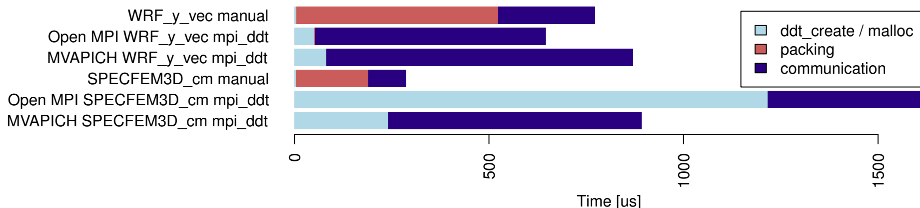


Fig. 6. Representative absolute benchmark results for comparing datatype creation and commit overheads, manual packing, and datatype communication overheads.

indicate that Open MPI's DDT engine is faster than manual packing for WRF, even if the datatypes were created for each communication (which is unnecessary in this case). But we also see that Open MPI has a much higher overhead for creating indexed datatypes, as used in SPECFEM3D, than MVAPICH.

4 Conclusions and Future Work

We analyzed a set of scientific applications for their communication buffer access patterns and isolated those patterns in micro-applications to experiment with MPI datatypes. In this study, we found three major classes of data access patterns: Face exchanges in n-dimensional Cartesian grids, irregular access of datastructures of varying complexity based on neighbor-lists in FEM, SEM and molecular dynamics codes as well as access of interleaved data in order to redistribute data elements in the case of matrix transpositions. In some cases (such as WRF) several similar accesses to datastructures can be fused into a single communication operation through the usage of a struct datatype. We provide the micro-applications to guide MPI implementers in optimizing datatype implementations and to aid hardware-software co-design decisions for future interconnection networks.

We demonstrated that the optimization of data packing (implicit or explicit) is crucial, as packing can make up up to 90% of the communication time with the data access patterns of real world applications. We showed that in some cases zero-copy formulations can help to mitigate this problem.

In the future we plan to extend our benchmark to allow for assessment of the overlap potential of different datatype engines. Another interesting possibility is studying how well different MPI DDT implementations make use of the available cache hierarchy. Of course the benchmark can also be extended by incorporating more application derived access patterns, for example by investigating parallel graph algorithms and codes.

Acknowledgments

This work was supported by the DOE Office of Science, Advanced Scientific Computing Research, under award number DE-FC02-10ER26011, program manager Sonia Sachs.

References

1. Aiken, A., Nicolau, A.: Optimal loop parallelization. SIGPLAN Not. 23(7), 308–317 (Jun 1988), http://doi.acm.org/10.1145/960116.54021
2. Alverson, R., Roweth, D., Kaplan, L.: The Gemini System Interconnect. In: 18th IEEE Symp. on High Performance Interconnects. pp. 83–87 (2010)
3. Barrett, R.F., Heroux, M.A., et al.: Poster: mini-applications: vehicles for co-design. In: Proceedings of the 2011 companion on High Performance Computing Networking, Storage and Analysis Companion. pp. 1–2. SC '11 Companion, ACM (2011)
4. Bernard, C., Ogilvie, M.C., DeGrand, T.A., et al.: Studying quarks and gluons on MIMD parallel computers. High Performance Computing Applications (1991)
5. Brunner, T.A.: Mulard: A multigroup thermal radiation diffusion mini-application. Tech. rep., DOE Exascale Research Conference (2012)
6. Byna, S., Gropp, W., Sun, X.H., Thakur, R.: Improving the performance of MPI derived datatypes by optimizing memory-access cost. In: Cluster Computing (2003)
7. Carrington, L., Komatitsch, D., et al.: High-frequency simulations of global seismic wave propagation using SPECFEM3D_GLOBE on 62K processors. In: ACM/IEEE conference on Supercomputing (2008)
8. Dixit, K.M.: The SPEC benchmarks. Parallel Computing 17 (1991)
9. Gropp, W., Hoefler, T., Thakur, R., Träff, J.: Performance expectations and guidelines for MPI derived datatypes. In: EuroMPI'10) (2011)
10. Heroux, M.A., Doerfler, D.W., et al.: Improving performance via mini-applications. Tech. rep., Sandia National Laboratories, SAND2009-5574 (2009)
11. Hoefler, T., Gottlieb, S.: Parallel Zero-Copy Algorithms for Fast Fourier Transform and Conjugate Gradient using MPI Datatypes. In: EuroMPI'10 (2010)
12. Lu, Q., Wu, J., Panda, D., Sadayappan, P.: Applying MPI derived datatypes to the NAS benchmarks: A case study. In: Intl. Conf. on Parallel Processing (2004)
13. McMahon, F.H.: The Livermore Fortran Kernels: A computer test of the numerical performance range. Tech. rep., Lawrence Livermore National Laboratory, UCRL-53745 (1986)
14. MPI Forum: MPI: A Message-Passing Interface Standard. Version 2.2 (2009)
15. Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. Computational Physics 117(1) (1995)
16. Reussner, R., Träff, J., Hunzelmann, G.: A benchmark for MPI derived datatypes. Recent Advances in Parallel Virtual Machine and Message Passing Interface (2000)
17. Skamarock, W.C., Klemp, J.B.: A time-split nonhydrostatic atmospheric model for weather research and forecasting applications. J. Comput. Phys. 227(7), 3465–3485 (Mar 2008), http://dx.doi.org/10.1016/j.jcp.2007.01.037
18. Träff, J., Hempel, R., Ritzdorf, H., Zimmermann, F.: Flattening on the fly: Efficient handling of MPI derived datatypes. EuroPVM/MPI (1999)
19. der Wijngaart, R.F.V., Wong, P.: NAS parallel benchmarks version 2.4. Tech. rep., NAS Technical Report NAS-02-007 (2002)
20. Wu, J., Wyckoff, P., Panda, D.: High performance implementation of MPI derived datatype communication over infiniband. In: Parallel and Distributed Processing Symposium (2004)