A Fair Benchmark for Evaluating the Latent Potential of Heterogeneous Coupled Clusters

Carsten Clauss, Stephan Gsell, Stefan Lankes, Thomas Bemmerl
Chair for Operating Systems, RWTH Aachen University,
Kopernikusstr. 16, 52056 Aachen, Germany
{clauss, gsell, lankes, bemmerl}@lfbs.rwth-aachen.de

Abstract

Coupled clusters usually exhibit a heterogeneous but also hierarchical structure in terms of communication and computation. Therefore, it is inevitable to adapt parallel applications to such systems in order to gain reasonable performance results. Moreover, also regular benchmark tools are not capable of exposing the latent potential of such coupled cluster systems. Though without adapted (or better self-adapting) benchmark tools for such systems, it is almost not possible to forecast the scalability of well-adapted applications and one is not able to compare the possibly achievable performance in an application independent manner. In this paper we present such a fair, self-adapting and meaningful benchmark tool for heterogeneous coupled cluster systems, following the MPI standard.

1. Introduction

Two significant trends can be recognized in the domain of high performance computing in the the past few years: On the one hand, cluster systems mainly build from commodity hardware have become more and more common. On the other hand, there is also a strong trend towards distributed scientific computing, as for example provided by grid computing environments. Thus, by combining distributed computing resources (namely computing clusters) into a higher level, an even higher degree of parallelism can be achieved and even larger problem instances can be solved.

1.1. Cluster of Clusters

Such a coupled system, which can be understood as a cluster of clusters (CoC), is heterogeneous by nature. Nevertheless, the system is presented to the application as a self-contained parallel computer, which still can be used in a transparent way. Therefore, such a system is also commonly called a *meta-computer*. Since MPI [20] is the most important API for implementing parallel programs, several MPI libraries have been extended to meet the demand of distributed and heterogeneous computing. The most common meta-computing and grid-enabled MPI libraries are MPICH-G2 [17], PACX-MPI [5], MPICH/Madeleine [2], GridMPI [18], StaMPI [25] and MetaMPICH [23], which are all proven to run large-scale applications on coupled cluster systems.

All those libraries offer the user and its applications a transparent view onto a pretended homogeneous MPI world, whereas the underlying computing environment is heterogeneous in various ways. The interlinking network between the coupled clusters usually constitutes the system's bottleneck. Additionally, different individual CPU power, different number of nodes and also different internal networks can lead to varying performance between the coupled clusters. While the libraries' feature of transparency makes the porting of existing MPI applications to such a system quite easy, the slowest network connection and the slowest compute node will dominate the achievable performance and scalability. Therefore, it is inevitable to adapt the applications to the underlying heterogeneous system in order to gain reasonable performance results. Although such an adaptation may cause the loss of source code compatibility with regular MPI implementations, many of the above mentioned grid-enabled MPI implementations offer a diversity of additional adaptation features.

1.2. Need for Adapted Benchmarks

Moreover, just like non-adapted applications do not scale, even regular benchmarks will just report poor performance when measuring the heterogeneous system in a transparent way. In fact, it is very difficult to forecast the scalability of adapted applications because of the lack of fair benchmark tools for such heterogeneous systems. In this paper, we describe the development of such a fair bench-

mark tool, which also exploits the latent potential of coupled clusters by taking the underlying heterogeneity into account.

The presented benchmark is based on the common and well-known parallel Jacobi Over Relaxation (JOR) algorithm, which is illustrated in many essential publications [7] [1] [15] [26]. Although the chosen algorithm is quite common, it can be seen as a simplification of various real-world solving algorithms. Thus, the benchmark process can easily be understood, while the measured results are still significant related to real applications.

The algorithm's adaptation on the underlying coupled system is done automatically by exploiting information about the system's heterogeneous nature. Instead of gaining this information by using incompatible customization features provided by the respective grid-enabled MPI library, the benchmark tool explores the system's topology in an initial benchmark step without sacrificing compatibility. That way, the adaptation is independent from the library used so that the tool can also be deployed for performance comparisons among different library implementations.

1.3. Related Work

Meta-computing and grid applications are still very active research areas. Nevertheless, the basic concepts of linking two or more compute sites into a higher unit are not very new and a lot of research has already been accomplished in these areas. While much of the prior work deals with the architectural approaches for building meta-computing environments up from dedicated supercomputers like MPPs [14] [24] [22], recent work often deals with grid-enabled solutions based on coupled cluster systems [21] [9] [8].

Since reasonable performance can only be obtained if the applications are optimized to take the heterogeneous environments into account, a lot of publications focus on this issue of optimizing and analyzing the achievable performance gain [3] [16] [11].

2. An Approach for a Fair Benchmark

In this section, we detail our approach to develop a fair and meaningful benchmark tool for systems characterized in Section 1.1.

On first consideration, an appropriate kernel algorithm has to be chosen. The elected algorithm must fulfill certain requirements concerning scalability and adaptability. For instance, the algorithm has to scale inherently with the number of processes on homogeneous systems. This means that the order of computation costs should exceed the order of communication costs. Furthermore, the algorithm must offer potentials to adapt the load of computation and communication to heterogeneity. Additionally, the algorithm should be comprehensible and significant for scientific computing.

Therefore, we have decided to base our benchmark on the Jacobi Over Relaxation algorithm (JOR), because it offers all those required attributes and is quite common at the same time.

2.1. The JOR Kernel

The simulation kernel being at the bottom of our benchmark is based on the well-known Laplace problem of heat transfer, a model partial differential equation (PDE). The mathematical task is to solve the Dirichlet boundary value problem for Laplace's equation on a rectangular domain. When discretizing Laplace's PDE on a two dimensional $N \times N$ mesh, the resulting linear equation system can be solved iteratively by applying the JOR method. The resulting Jacobi iterations are then described by the following formula:

$$x_i^{k+1} = \frac{x_{i-N}^k + x_{i-1}^k + x_{i+1}^k + x_{i+N}^k}{4}$$

where x_i is the value of the ith mesh point computed from the ith equation of the system, while the superscript k indicates the respective iteration step. This common scheme, known as the *five-point-stencil*, is shown in Figure 1.

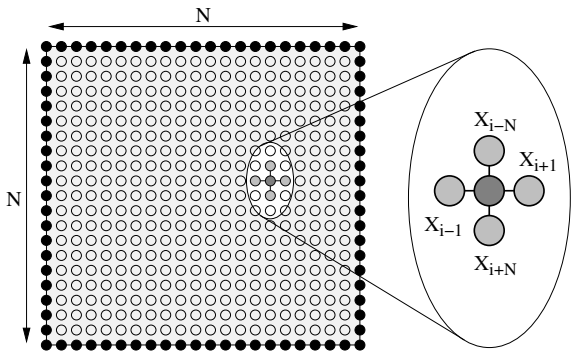

Figure 1. The Laplace Problem on a Rectangular Mesh

We emphasize that the Laplace equation as well as the JOR method are the simplest representatives of their classes respectively. Although there exist plenty more recent and more appropriate techniques to solve even more complex and even more significant PDEs, we use the simple Laplace problem and the simple JOR solver as a fundamental example of simulation algorithms.

2.2. Parallelization

When parallelizing the algorithm, a decomposition of the discrete mesh either into square blocks or rectangular stripes or columns can be applied. By assigning each of those same-sized areas to one of the parallel processors, a good load balance can be achieved in a homogeneous system. In order to determine the exact same approximation of the PDE as in a serial run, each processor must communicate with all its adjacent neighbors after each iteration step (see Figure 2). Note that obtaining the same numerical results in a parallel run as in a serial run is an attribute of the JOR algorithm, whereas the convergence behavior of other solvers, e.g. the Successive Over Relaxation method (SOR), is not independent of the number of processors used.

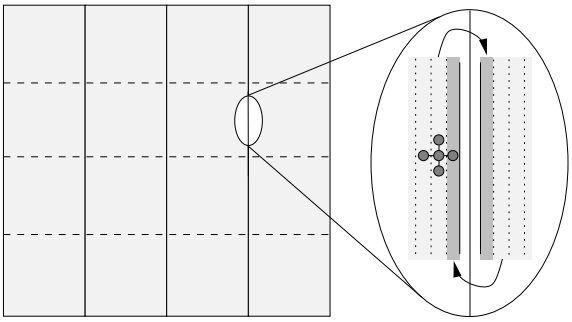

Figure 2. Domain Partitioning and Communication

Just like for the sequential algorithm, the Jacobi iterations should terminate when the approximated solution is good enough. Therefore, each processor must initially determine the current local error of its sub-domain during the iterations. Subsequently, after a reduce operation has been performed among all processors, the global termination condition can be checked during the iterations.

2.3. Adaptation to Heterogeneity

Obviously, when transferring the algorithm from a homogeneous system to a heterogeneous system, load balancing gets more complicated. If retaining a static assignment of the mesh areas onto the heterogeneous processors, information about the respective computational performance must be gathered and introduced into the primary domain decomposition. Therefore, the same algorithm should be deployed as an initial serial benchmark in order to measure the performance characteristics of each processor involved. Thus, a fair and adapted benchmark tool for heterogeneous systems must include a preliminary benchmark-run that serves to explore the unknown nature of the system.

Besides a non-uniform distribution of compute power, heterogeneous systems often exhibit an inhomogeneous communication infrastructure, too. In case of the Laplace problem and the applied JOR solver, the chosen domain decomposition also governs the communication pattern of the algorithm. Therefore, a fair and adapted benchmark tool must provide an optimal mapping of the logical communication structure onto the underlying hardware communication infrastructure by means of a proper decomposition scheme.

2.4. Asynchronous Relaxation

Furthermore, in case of the JOR solver, an additional dependency between the computational load and the communication structures can be introduced: While the common parallel JOR algorithm is *synchronous* in a way that all processors are working at the same iteration step at the same time, there is also an *asynchronous* relaxation procedure thinkable. Suppose the blocking communication (and thus the synchronization) after each iteration to be removed, a processors would be able to go on with the next iteration step after *checking* for new messages without the need of *waiting* for them. The idea of such a *chaotic* updating scheme was already introduced by Chazan and Miranker in their seminal paper [10] in 1969. Therein, they stated a simple necessary and sufficient condition that guarantees the convergence also in case of such asynchronous iterations. Since its introduction, asynchronous relaxation has been studied by many authors [13] [4] [19] [6], whereas much work deals with the special case of *periodic* relaxation.

Such a periodic relaxation scheme exhibits a well-defined periodic updating order, in which each processor performs a fixed number of relaxation iterations without communication before being synchronized. While the local errors decrease during such an asynchronous relaxation period, they suddenly rise again at the beginning of the next period. This is because the updated neighbor values disturbs the local relaxation after communication. For that reason, the global error and the respective termination condition can only be checked after a synchronization step.

In Figure 3, the error evolution is plotted once for a synchronous and once for a periodic (aka *partially synchronous*) relaxation method with a period of $\Delta = 10$. Note that the line of the partially synchronous evolution is just plotted as the envelope of the actual evolution according to the periodic updating order. As one can see, the convergence rate for the partially synchronous method is smaller than that of the full synchronous one, but the number of communication cycles is reduced (by a factor of ten in this example). Thus, it is possible within this algorithm to compensate weak communication resources by deploying com-

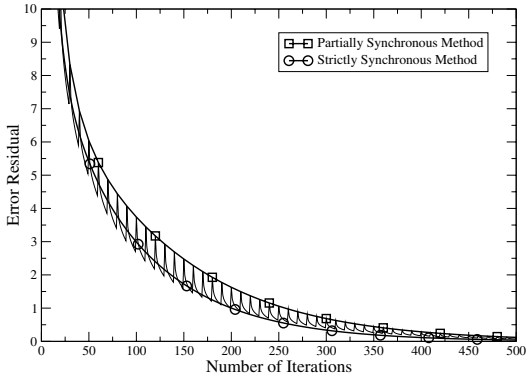

Figure 3. Error Evolution for Strictly and Partially Synchronous Methods

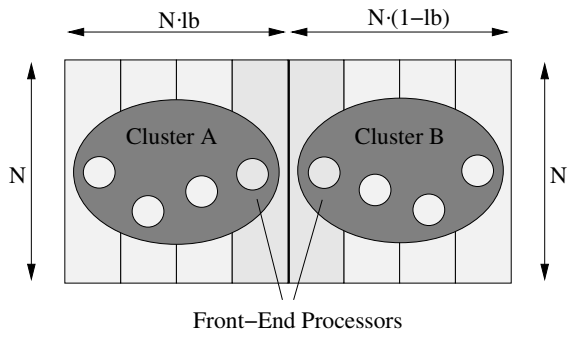

Figure 4. Initial Domain Decomposition for the two Clusters

puting power by means of performing additional iteration steps.

3. Benchmark Description

The benchmark assumes a heterogeneous system build of *two* homogeneous computing sites (e.g. two clusters), each with an arbitrary number of parallel processors. The benchmark tool measures the parallel solving time for those processors applying a partially synchronous JOR solver for the Dirichlet problem of Laplace's equation on a rectangular domain. Thereby the tool performs multiple runs with varying parameters.

The decomposition of the $2 \cdot N \times N$ domain is based on a strip partitioning, which is denoted in Figure 4 (lb is a load balancing factor). Such a strip partitioning has the advantage that each processor only needs to communicate at most with *two* neighbors, and that in turn reduces the impact of latency. Note that there are only two processors to communicate across the inter-cluster-link, which is assumed to be the system's bottleneck. Therefore, we call those distinctive processors the *front-ends* of each site. The benchmark automatically chooses that processor pair to be the front-ends, which exhibit the fastest inter-cluster communication characteristic.

3.1. The Start-Up Phase

First of all, the benchmark tool must be aware of the supposed two-tier system hierarchy. Therefore, the benchmark offers two possibilities: either the tool can determine the hierarchy by itself by performing simple round-trip time measurements among all participating nodes, or the site affiliation must be stated explicitly by the user via a configuration file or via a hostname mask. While the first option is quite user friendly, the search for the hierarchy classes demands start-up communication in the magnitude of $O(p^2)$, which is not feasible in large parallel environments.

3.2. Determining the Power Distribution

The next step is to determine the relative computational power of each site. For this purpose, each cluster site performs a local parallel run on the $N \times N$ domain on its own by applying the fully synchronous JOR method *internally* without any interaction between the sites. While the number of iterations to be performed is the same for both sites, the measured times lead to the ratio of computing power between the two clusters.

Note that this ratio is also governed by the cluster internal communication because of its impact on the local parallel run-time. Thereby, the chosen partitioning scheme of the solution domain potentially impacts the performance. This is due to the fact that a block partitioning scheme is best for networks with small latencies, while a striped scheme is best in case of larger communication latencies [26]. Nevertheless, since the *inter*-cluster communication pattern becomes more complex in case of a block-wise decomposition, we still assume a striped partitioning. Note that the number of processors per cluster may differ between the sites.

After the first local run on an $N \times N$ domain, a second local run on a $2 \cdot N \times N$ domain is going to be performed. The reason for that further local run is to analyze the impact of the caches on the performance of the coupled system. This is because the cache size may be large enough to hold the local problem of a *coupled* cluster configuration, but too small to carry the whole problem, when it is only spread among a *single* cluster. Conversely, if the measured time ratios of the local runs on the differently sized domains are not approximately the same, then a cache boundary has been crossed on one of the clusters. Hence, in such a case,

these cache effects have also to be taken into account when determining an optimized load balancing.

3.3. A Transparent Run

Although the system's heterogeneous nature is now determined, the next step is just a *transparent* benchmark-run with both coupled clusters working together. This run is transparent in that way that there is still *no* load balancing applied between the clusters and that communication is performed straight forward in each iteration step. Hence, the measured run-time gives information about the performance gain (or even the performance loss) when running an accordingly unadapted application on the coupled system instead of using the most capable of both sub-systems without the other's help.

It has to be mentioned at this point that all those benchmark-runs clock the time of iterations needed to divide the initial error by a factor of ten. In order not to include time consumed by the termination detection mechanism (that are mainly the all-reduce operations for calculating the global error evolution), the number of needed iterations is predetermined before the actual measuring run. Thus, a clocked benchmark-run always executes a fixed number of iterations while disregarding the current error evolution. In real world applications, the impact of the global termination detection mechanism is usually reduced by checking the abort criteria just after a certain amount of iterations in a cyclic manner. This approach works because the JOR method converges quite slowly. [15]

3.4. The Best Periodic Updating Scheme

Now it is time to determine the best periodic updating scheme concerning the inter-cluster communication. Since the link between the front-end nodes is assumed to be the system's bottleneck, the *partially* synchronous scheme is just applied to this pair of nodes, whereas the local updating scheme is still a synchronous one. Therefore, the tool performs a series of benchmark-runs each with a different amount of local iterations between the inter-cluster synchronization steps. The first run of this series is just a fully synchronous relaxation, while the updating period gets respectively increased in each next run. Thereby, the number of iterations needed for the sufficient relaxation is furthermore predetermined for each applied updating period in order to clock just the respective solving time. Note that the number of iterations to be performed also depends on the load balance applied because the domain frontier between the clusters may be displaced.

All of those runs are performed with balanced load. This means that the $2 \cdot N \times N$ domain is distributed onto the two clusters according to the previously measured power ratio. Hence, the first and thus the fully synchronous run of the series gives information about the impact of an adapted load distribution compared to the results of the transparent run.

As already explained, when increasing the updating period between the clusters, the inter-cluster communication effort decreases, whereas the number of iterations to be performed increases. Therefore, in case of a slow interlinking network, there must be an optimal periodic updating scheme which exhibits a period of $\Delta > 1$. This is because up to this certain period, the impact of the reduced communication time dominates over the increased computation time caused by the additional iterations. On the other hand, if the interlinking network is about as fast as the internal networks, the synchronous scheme with a period of $\Delta = 1$ becomes potentially the best choice.

This correlation is denoted in Figure 5 of the next section, where the run-time of the solver is plotted over the period applied. As one can see in this example, in case of a slow inter-cluster network, an adaptation by means of an optimal chosen updating period can enhance the performance of a coupled system in a distinctive way.

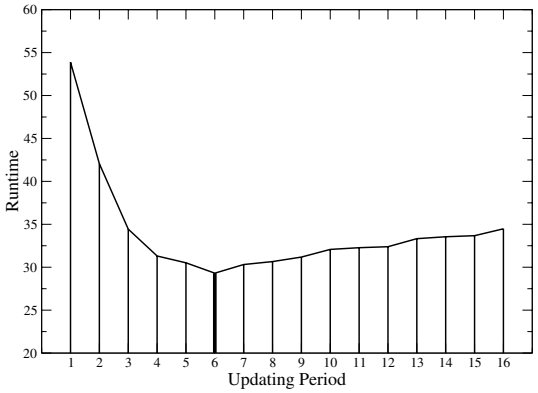

Figure 5. Runtime over Updating Periods Δ for a Class A Problem

3.5. Summary of Benchmark Procedure

At first, the benchmark tool performs local runs on an $N \times N$ and a $2 \cdot N \times N$ domain without any inter-cluster communication. The results of these two runs are used to determine the distribution of compute power in order to provide an optimal load balancing.

Next, there is a transparent run on a $2 \cdot N \times N$ domain still without load balancing applied, but with an inter-cluster synchronization after each iteration step. Comparing this run-time with the the shortest local run-time on the $2 \cdot N \times N$

domain provides the information if also an unadapted run can already benefit from the cluster coupling. This is what we call the *Artless Speedup*.

The next series of runs are all performed with an optimal balanced load, while the synchronization period gets increased in each run. Finally, the run-times measured in this manner help to dentify the best period, which in turn implies the best run-time. Hence, the fraction of the best local run-time with this best global run-time is here called the *Artful Speedup*. Note that these speedup values refer to the coupled system of *two* clusters, thus an optimal speedup is about a factor of *two*.

4. Selected Results

In this section, we want to present some results that we have measured with the benchmark tool introduced. All these benchmark runs have been performed on a meta environment composed of the following two clusters, which are both equipped with the Scalable Coherent Interface (SCI) as fast internal networks (275 MB/s MPI-Bandwidth, 4μs MPI-Latency) and with Gigabit-Ethernet:

P4-Cluster:

- 8 Nodes, 16 CPUs
- Dual Intel Xeon 2.4 GHz, 512 kByte Cache
- 400 MHz FSB, 8 GByte Total Memory
- Scalable Coherent Interface (SCI), 2*4 2D-Torus

PD-Cluster:

- 16 Nodes, 16*2 (DualCore) CPUs
- Intel PentiumD 2.8 GHz, 1 MByte Cache
- 533 MHz FSB, 32 GByte Total Memory
- Scalable Coherent Interface (SCI), 3*4 2D-Torus

We have used MetaMPICH as grid-enabled MPI library in order to link both clusters. MetaMPICH is a part of the MP-MPICH project, which in turn also includes the SCI-MPICH library [27]. That way, MetaMPICH supports a direct access to the underlying local SCI networks for cluster-internal communication, while the interlinking communication is based on TCP drivers.

4.1. The Problem Classes

The size N of the problem solved by the benchmark tool during a run is governed by a so-called problem class, which has to be chosen at startup time. This approach is borrowed from the NAS benchmark suite [12], whereas the classes and their sizes are chosen here as follows:

Class	Size	Purpose
S	16×16	Small Systems and Test-Runs
W	128×128	Small Clusters of Workstations
A	256×256	Midsize Productivity Clusters
B	512×512	High Performance Clusters
C	1024×1204	Coupled Top-500 Systems

In our first benchmark scenario, the two clusters were coupled via Gigabit-Ethernet, while the internal communication was handled over the SCI network. Thereby, only the two front-end nodes had to communicate across the interlinking bottleneck as it has already been denoted in Figure 4. Note again that the benchmark automatically chose the best pair of nodes for the inter-cluster communication.

In Figure 6 and Figure 7, the measured speedup is plotted over the number of nodes used *per* cluster. That means that the number of processes was identical for both clusters. Note that in this case the power distribution and thus the load balance is almost symmetric because the performance characteristics are quite similar for both systems. We have only measured the speedups for problems of the classes W,A and B. The reason was that the S class is just intended for test runs and the C class is too large for our cluster testbeds.

As one can see, for small problem sizes like in class W, the *Artless Speedup* is rather a loss of speedup than a performance gain. However, when increasing the problem sizes, the achievable speedup rises, too. This is due to the fact that the order of the computational load is $O(N^2)$, whereas the communication is about $O(N)$. That means that the impact of the inter-cluster bottleneck shrinks when the problem size grows. On the other hand, the impact of the internal communication increases when the number of nodes per cluster gets incremented. That is the reason why the speedup drops for larger environments.

Nevertheless, when looking at the *Artful Speedup*, it becomes clear that customizing the application (here represented by the adapted benchmark run) is inevitable on heterogeneous systems in order to achieve efficient performance gains.

4.2. Load Balancing

In Figure 8, the applied load balancing is plotted over the number of nodes for the different problem classes. Here, the percentage deviations for the applied decomposition of the $2 \cdot N \times N$ domain for the two clusters are shown. Remarkable is that the load balance and thus the determined distribution of computational power varies between the different classes. While the measured stand-alone performance of the two cluster systems is quite similar for the W class, the P4 cluster seems to weaken especially when applying a class

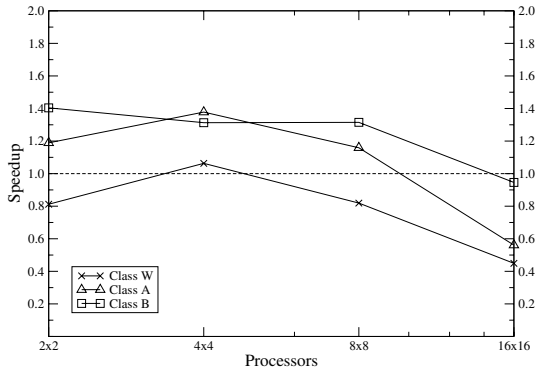

Figure 6. Measured *Artless* Speedup for Different Problem Classes

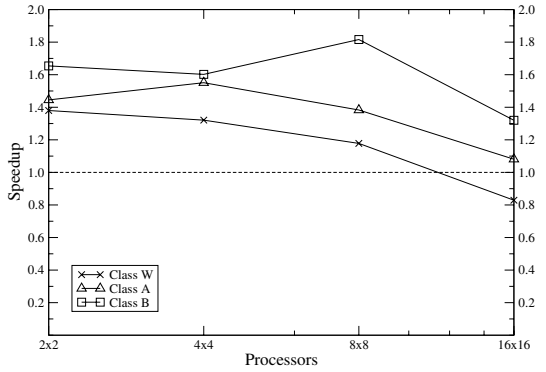

Figure 7. Measured *Artful* Speedup for Different Problem Classes

A or B problem. The reason for this strange behavior are the different cache sizes of the two cluster systems. While a CPU of the PD-Cluster posseses a cache size of $1MB$, the P4-Cluster just exhibit $512KB$ cache memory. The class A problem demands a total memory of

$$2 \cdot (2 \cdot 256 \times 256) \cdot 8Bytes = 2MBytes$$

on a $2 \cdot N \times N$ domain. Note that the JOR method needs a duplicated solution matrix. Thus, when just deploying the P4-Cluster to solve the whole problem on its own, the problem would not fit into its CPU caches when using up to four processor nodes. This issue can clearly be recognized in Figure 8, where the assigned load is unequal up to a coupling of two four-node systems. Note that for an optimal load balance, these issues have to be taken into account. Although up to eight nodes the class B problem ($8MB$) even does not fit into the caches of the PD-Cluster, the PD-Cluster outperforms the P4-Cluster also for such large problems due to its faster main memory bandwidth.

A further discussion on load balance can be based on Figure 9, where the measured speedups are plotted for an asymmetric number of nodes applied for each cluster. In this benchmark scenario, there are always twice as much processes started on the P4-Cluster as on the PD-Cluster. Remarkable are the speedup measurements for the (4x2)-nodes environment, where the Artful Speedup exhibits an efficiency of up to 90%. This is due to fact that both systems would not be able to hold the whole problem in their caches in case of a stand-alone run. Though when working together, the subproblems fit into the caches so that coupling the systems is very worthwhile in this case.

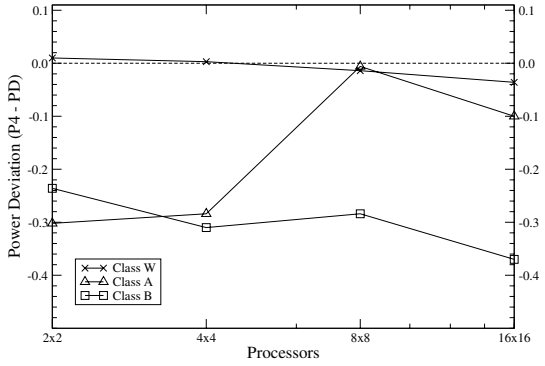

Figure 8. Measured Power Ratio between P4- and PD-Cluster

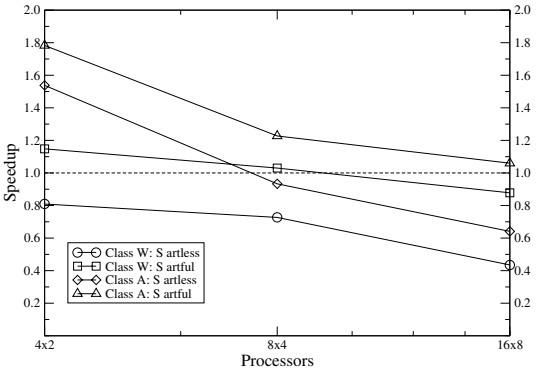

Figure 9. Speedups for Asymmetric Environments

5. Conclusions and Outlook

In this paper, we have presented a tool to benchmark coupled clusters, so-called meta-computers. With that benchmark, one can check if it makes sense to couple two clus-

ters (of different performance) to solve a given problem, or if it even slows down the faster cluster. Also different configurations can be checked, by taking the load balance into account. Since our tool is independent of the underlying, grid-enabled, MPI library, it can also be used to compare these different implementations with each other. We have shown that the speedup can indeed be increased by adapting the existing MPI-application to the given meta-computer, rather than just coupling two clusters and let the application run transparently on them.

In the future, we might add an option to split the $2 \cdot N \times N$ area into blocks, rather than stripes, since this is better for intra-cluster networks with low latency [26]. Although such a decomposition pattern does not reduce the impact of the inter-cluster latency, the aggregated bandwidth between the clusters may also be increased due to the pairs of processes exchanging data across the inter-cluster link in this case.

References

[1] Y. Aoyama and J. Nakano. *RS/6000 SP: Practical MPI Programming (SG24-5380/ISBN 0738413658)*. IBM Japan redbook, 1999. RS/6000 Product Management & Marketing, IBM Japan.

[2] O. Aumage and G. Mercier. MPICH/MADIII: a Cluster of Clusters Enabled MPI Implementation. In *Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages pp. 26–36, May 2003.

[3] H. E. Bal, A. Plaat, M. G. Bakker, P. Dozy, and R. F. H. Hofman. Optimizing parallel applications for wide-area clusters. In *IPPS/SPDP*, pages 784–790, 1998.

[4] G. M. Baudet. Asynchronous iterative methods for multiprocessors. *JACM: Journal of the ACM*, 25:226–244, 1978.

[5] T. Beisel, E. Gabriel, M. Resch, and R. Keller. Distributed Computing in a Heterogeneous Computing Environment. In *Proc. of the 5th European PVM/MPI Users' Group Meeting*, pages pp. 180–187, September 1998.

[6] D. Bertsekas and J. Tsitsiklis. Convergence Rate and Termination of Asynchronous Iterative Algorithms. In *Proceedings of the 1989 International Conference on Supercomputing*, pages 461–470, 1989.

[7] D. P. Bertsekas and J. N. Tsitsikilis. *Parallel and distributed computation: numerical methods*. Prentice Hall, Englewood Cliffs, N.J., 1989.

[8] B. Bierbaum, C. Clauss, T. Eickermann, L. Kirtchakova, A. Krechel, S. Springstubbe, O. Wäldrich, and W. Ziegler. Orchestration of distributed MPI-Applications in a UNICORE-based Grid with MetaMPICH and MetaScheduling. In *Proceedings of the European PVM/MPI Users Group Meeting 2006*, Bonn, Germany, September 2006.

[9] B. Bierbaum, C. Clauss, M. Pöppe, S. Lankes, and T. Bemmerl. The new Multidevice Architecture of MetaMPICH in the Context of other Approaches to Grid-enabled MPI. In *Proceedings of the European PVM/MPI Users Group Meeting 2006*, Bonn, Germany, September 2006.

[10] Chazan, D. and Miranker, W. L. Chaotic relaxation. *Linear Algebra and its Applications*, 2:199–222, 1969.

[11] C. Clauss, M. Pöppe, and T. Bemmerl. Optimising MPI Applications for Heterogeneous Coupled Clusters with MetaMPICH. In *Proceedings of the IEEE International Conference on Parallel Computing in Electrical Engineering*, Dresden, Germany, September 2004.

[12] D. Bailey et al. The NAS parallel benchmarks. Technical Report RNR-91-002, NAS Systems Division, Jan. 1991.

[13] J. D. P. Donnelly. Periodic chaotic relaxation. *Linear Algebra and Application*, 4:117–128, 1971.

[14] T. Eickermann, J. Henrichs, M. M. Resch, R. Stoy, and R. Völpel. Metacomputing in gigabit environments: Networks, tools, and applications. *Parallel Computing*, 24(12-13):1847–1872, 1998.

[15] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI - second edition*. The MIT Press, 1999.

[16] J. Henrichs. Optimizing and load balancing metacomputing applications. In *International Conference on Supercomputing*, pages 165–171, 1998.

[17] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Grid-enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63(5):551 – 563, 2003.

[18] M. Matsuda, Y. Ishikawa, Y. Kaneo, and M. Edamoto. Overview of the GridMPI Version 1.0. In *Proc. of the SWoPP05, Japan*, 2005.

[19] D. Mitra. Asynchronous relaxations for the numerical solution of differential equations by parallel processors. *SIAM Journal on Scientific and Statistical Computing*, 8(1):S43–S58, Jan. 1987.

[20] MPI Forum. MPI: A message-passing interface standard. *International Journal of Supercomputing Applications*, 1994.

[21] M. S. Müller, M. Hess, and E. Gabriel. Grid enabled MPI solutions for clusters. In *CCGRID*, pages 18–25. IEEE Computer Society, 2003.

[22] S. Pickles, F. Costen, J. Brooke, E. Gabriel, M. S. Müller, M. M. Resch, and S. Ord. The problems and the solutions of the metacomputing experiment in SC99. In *Proceedings of the 8th International Conference on High-Performance Computing and Networking (HPCN)*, Amsterdam, The Netherlands, 2000.

[23] M. Pöppe, S. Schuch, and T. Bemmerl. A Message Passing Interface Library for Inhomogeneous Coupled Clusters. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, France, April 2003.

[24] M. M. Resch, D. Rantzau, and R. Stoy. Metacomputing experience in a transatlantic wide area application testbed. *Future Generation Computer Systems*, 15(5–6):807–816, Oct. 1999.

[25] H. K. Toshiyuki Imamura, Yuichi Tsujita and H. Takemiya. An Architecture of Stampi: MPI Library on a Cluster of Parallel Computers. In J.Dongarra et al., editor, *EuroPVM/MPI2000*, pages 200–207. Springer-Verlag Berlin Heidelberg, 2000.

[26] B. Wilkinson and M. Allen. *Parallel Programming*. Prentice Hall, 2nd edition, 2005.

[27] J. Worringen. SCI-MPICH - The Second Generation. In *Proceedings of SCI-Europe 2000 (Conference Stream of Euro-Par 2000)*, pages 11–20, Munich, Germany, August 2000.