Considering Core Density in Hybrid Clusters

Eduardo Grosclaude, Claudio Zanellato, Javier Balladini,
Rodolfo del Castillo, Silvia Castro[1]

Facultad de Informática, Universidad Nacional del Comahue
[1]Dpto. de Cs. e Ing. de la Computación, Universidad Nacional del Sur
{oso,czanella,jballadi,rolo}@uncoma.edu.ar, smc@cs.uns.edu.ar

Abstract. As new HPC technologies appear, small local research laboratories face uncertain options when building hybrid clusters. Users may find difficult to choose among several multicore products with different core densities. Our research project intends to build knowledge about HPC problems to be able to help local researchers. We analyze NUMA hardware for use in clusters and present a case study. We run a well-known benchmark over MPI and advise the user depending on application features.
Key words: Hybrid Clusters, NUMA, MPI.

Local Situation

While exceptions do exist, research labs in Argentina's scientific community are frequently built on skim resources. Often, local research groups do their computational work with low-entry equipment such as commodity computers and consumer-grade interconnects. Most HPC users among local researchers have been using message-passing parallel applications over clusters of desktop PCs for some time now. These clusters are usually built upon regular, consumer hardware, and have proved to offer a good platform to run scalable parallel applications while being inexpensive to expand.

In the past, users knew that just buying new hardware would make their programs run faster. However, as computer industry meets physical and engineering dead ends, this is no longer true. Machines are and will be more complex, yet not necessarily faster -unless parallelism fits into the scene [7,9]. From now on, new nodes added to clusters will invariably have multiple cores. Clusters will exhibit a two-level set (intra-node, inter-node) of parallel resources. This hierarchical cluster architecture -often called a hybrid cluster- poses many questions about the best way to take advantage of multi-level parallelism. Applications' characteristics such as scalability, computation/messaging granularity or memory usage patterns, are to be analyzed and understood for the suitable hardware to be identified.

Clusters will continue to be used, and they will be more and more heterogeneous. This adds to complexity, given the diversity of offerings:

- Older, unicore cluster nodes are still useful.
- Newer computing hardware trends favor slower cores but fast internal interconnects.
- Low-entry hardware carries a smaller number of cores.

– However, new products continually increase this number.
– Massively parallel hardware like GPUs enables new possibilities.
– Meanwhile, network interconnects advance at a much slower pace.

And given the restrictions they suffer:

– Memory is often a bottleneck.
– Applications have their own scalability limits.
– Some form of load balance is needed to mask away heterogeneity.
– Power and thermal needs also bring their own restrictions to the scene.

If users need to increase their computing resources, they can seriously wonder how to. As new multicore and manycore architectures hit the market, users face new decisions to make. However, it is not an easy task for a user to match applications which have "just run" for years on whatever platform, to new, unknown architectures. Exposed to new computational tools, most non-expert users will be uncertain when judging:

– To what extent their applications will benefit from a given change of platform -if at all.
– How their applications should be reprogrammed -if needed.
– How scalability problems uncovered by the new platforms may be detected and corrected when possible.
– What new optimization strategies may be applicable under the new platforms to make their applications run better.

Our Research Project

Our recently started research group at Universidad Nacional del Comahue aims to build knowledge about new resources, and help local researchers from other scientific and engineering fields to better understand them and adopt them when they prove suitable [1]. To this end we need to develop a thorough understanding of capital ideas and facts in HPC. This involves general, theoretical constructs such as problems and models, but also case studies, such as algorithms and applications, rooted into particular disciplines.

This research project activities have begun in 2010. The project also intends to help some of its members pursue their postgraduate studies. The present work is a preliminary step towards the formulation of one Magister thesis in HPC, oriented towards application performance prediction in hybrid environments.

The Present Work

Our present study is a first inquiry into how a process of technological change in computing facilities can be faced by a researcher who is not a computer specialist. We envision a fairly common scene in the local scientific community: a local engineer or scientist who runs applications in a clustered environment wants to update her computing equipment. Our user has been following the traditional practices of building unicore clusters. Being offered multicore hardware, she now can make herself quite a few questions.

– She may want to know how the computing power of a cluster with n nodes with c cores each compares to a single $n * c$ cores multicore machine. Will she be better off buying a single, powerful machine or several less powerful machines for the same amount of money? Will any scalability concerns be revealed? This question is ultimately related to how applications behave regarding communications patterns.
– She may ask herself whether her new equipment will just plug and play into her cluster. Will she still be able to use her older equipment? Will the extended cluster perform better? This is a question related to how applications manage load balance, and to how they tolerate heterogeneity. Applications favoring "work stealing" approaches are succesful in these environments, while static strategies may perform badly.
– Will she need any change in her software, working practices or environment? If applications are not to be modified, which is the set of tuning strategies and techniques that can be readily applied to existing environments, with the minimal disturbance to the existing user practices and with minimal intervention from users or system administrators? The user wants to keep costs, under the form of disturbances or learning curves, at a minimum. This question is related to how existing runtime libraries such as MPI can work on different underlying platforms.

These questions have risen in an actual case where we served as consultants, which led us to the present study. Our goal is to make the user a sensible recommendation backed by quantitative reasoning, and look for general principles we can apply to other cases. To this end we devise a simple experiment with a black box approach to compare two proposed configurations.

In the next sections we present some problems related to hybrid clusters and some methodological remarks. Later on, we make a small survey of run-time tuning techniques for parallel applications. In the next sections we describe our experience, explain our results and describe some future research plans.

Hybrid Clusters

The current evolutionary stage of multicore machines includes Non Uniform Memory Access (NUMA) designs. As the number of cores increases, access to memory becomes a bottleneck. As a result, memory controllers are engineered on-chip and given their own memory banks to work with. A typical processor now packs a number of computing cores, along with a "noncore" or common area to accomodate interconnect and memory controllers. Systems are built on several of these processors. Special point-to-point interconnects are designed to carry data and cache coherency traffic across processors, replacing former bus strategies [13].

These inherently parallel designs are already being offered to customers as commodity hardware, and raise questions about how existing applications will best leverage these complex architectures. Current Intel QPI [3] or AMD Hypertransport [2] interconnects are capable of transferring data among processors at hundreds of Gbps. Their bandwidth and latency properties compare at very high ratios to inexpensive LAN links commonly found in clusters.

Several approaches to using clusters of multicores have been described. Some of them exploit both levels of parallelism by multithreading individual processes [6,8,17,16]. Others examine the effect of running several pure MPI processes over multicore machines [11]. The latter approach is the regular practice our non-expert researcher has been following. Keeping on with this practice is desirable, to avoid reprogramming of applications or modifying the user's regular work logistics such as job launching or scripting techniques.

Given the fact that our user plans to acquire specific NUMA equipment, we translated the user's questions into the following technical formulation.

- Should the advance in interconnect speed linearly reflect into applications' behaviour?
- Will applications naturally and automatically profit from this increase both in processing power and in communication speed?

To test these ideas we selected NAS Parallel Benchmarks, a set of parallel kernels and applications [4]. NAS Parallel Benchmarks (NPB) is a well-known, well-established suite which offers a broad range of characteristic programs. NPB is a benchmark designed by NASA for hardware testing, consisting of several programs related to Computational Fluid Dynamics (Table 1). The whole suite comes in several implementations: serial, shared-memory parallel (Open MP) and message-passing parallel (MPI). Most programs are coded in Fortran, some in C. Implementations for Java and HPFortran are available as well since version 3.3.

Programs in the suite solve problems in several different preset data sizes called classes. Classes have a different meaning for each particular program, but are equally ranked for every program (Table 2). In our laboratory, "D" and upper classes caused swapping, so we did not take them into account. Classes "S" and "W" were too small sized to give reliable time estimations, so they were discarded as well. The MPI version of the benchmarks was finally run over classes "A", "B" and "C", on a cluster of NUMA machines.

MPI on NUMA Machines

An interesting question is how does MPI view platforms other than unicore clusters, especially NUMA machines. MPI runtime systems offer mechanisms to tune certain action modes, some of them automatically falling back to sane defaults.

Core allocation Users of the Linux SMP kernel are able to modify natural kernel policies regarding core allocation to programs [14]. This can be achieved by means of several actions. At the system level, special boot directives can be added to kernel parameters at boot time (core isolation), or cores can be taken offline or online dynamically during execution of the system. At the process level, processes can be run with affinity hints to tell the scheduler to exclude them from certain cores.

MPI Execution Parameters The Linux scheduler tends to keep a process in the same CPU where it started. This is convenient for reasons of cache reusing, but it especially applies to NUMA nodes, where thread migration is even more costly across

BT	BT is a simulated CFD application that uses an implicit algorithm to solve 3dimensional (3D) compressible NavierStokes equations. The finite differences solution to the problem is based on an Alternating Direction Implicit (ADI) approximate factorization that decouples the x, y, and z dimensions. The resulting systems are BlockTridiagonal of 5x5 blocks and are solved sequentially along each dimension.
SP	SP is a simulated CFD application that has a similar structure to BT. The finite differences solution to the problem is based on a BeamWarming approximate factorization that decouples the x, y, and z dimensions. The resulting system has scalar Pentadiagonal bands of linear equations that are solved sequentially along each dimension.
LU	LU is a simulated CFD application that uses symmetric successive overrelaxation (SSOR) method to solve a seven block diagonal system resulting from finite difference discretization of the NavierStokes equations in 3D by splitting to into block Lower and Upper triangular systems.
FT	FT contains the computational kernel of a 3D fast Fourier Transform (FFT) based spectral method. FT performs three one dimensional (1D) FFT's, one for each dimension.
CG	CG uses a Conjugate Gradient method to compute an approximation to the smallest eigenvalue of a large, sparse, unstructured matrix. This kernel tests unstructured grid computations and communications by using a matrix with randomly generated locations of entries.
EP	EP is an Embarrassingly Parallel benchmark. It generates pairs of Gaussian random deviates according to a specific scheme. The goal is to establish the reference point for peak performance of a given platform. EP is almost independent of the interconnect as communication is minimal.
MG	MG uses a Vcycle MultiGrid method to compute the solution of the 3D scalar Poisson equation. The algorithm works continuously on a set of grids that are made between coarse and fine. It tests both short and long distance data movement.
IS	IS is a parallel integer sort algorithm that is very sensitive to latency of the interconnect.

Table 1: NAS NPB3.3 acronyms and their meaning.

	BT	CG	EP	FT	IS	LU	MG
S	12x12x12	1400	33554432	64x64x64	65536	12x12x12	32x32x32
W	24x24x24	7000	67108864	128x128x32	1048576	33x33x33	128x128x128
A	64x64x64	14000	536870912	256x256x128	8388608	64x64x64	256x256x256
B	102x102x102	75000	2147483648	512x256x256	33554432	102x102x102	256x256x256
C	162x162x162	150000	8589934592	512x512x512	134217728	162x162x162	512x512x512

Table 2: Problem data size for NPB3.3 classes

processor boundaries [12]. Also, memory in NUMA nodes is allocated to threads according to the first-touch policy. This means that a thread who first references a memory location will cause such memory to be taken from the processor where it runs, so as to minimize access distance [10]. Open MPI runtime system allows for parallel execution policies to encourage or discourage attachment of parallel processes to cores. Open MPI allows for narrow control of CPU allocation at run time by enabling the user to precisely describe the underlying hardware and mapping MPI processes to individual cores. The notion of slot (an identifier for independent threads –or equivalently, cores, in a node) fits to this end.

Message size Some MPI implementations allow the user to tune the boundary between *eager* (short messages) and *rendezvous* (long messages) protocols for maximal efficiency.

Byte transfer layer (BTL) MPI run-times usually can switch their messaging protocol implementation to adapt to multicore architectures. Open MPI can automatically switch between shared memory and TCP segments communication modes, depending on communicating processes being or not on the same node. While these modes can be forcibly induced in Open MPI by using the Modular Component Architecture (MCA) general mechanism, multicore architectures are exploited by Open MPI by automatically using the Byte Transfer Layer framework. As a necessary condition upon the code of programs, any tunable constants or strategies should be factored out from the code and specified as MCA parameters for this tuning mechanism to be taken advantage of.

Users have an array of tools to tune the performance of systems running MPI parallel programs. These tools help avoid rewriting code, and can easily modify performance or efficiency of the running applications when applied by users or by system administrators. We are excluding all these explicit tuning actions from our tests in this occasion because the default behaviours are good enough. However, they are all extremely interesting for future development of performance models.

Laboratory

We will use the NAS Parallel Benchmarks NPB3.3 on MPI using Open MPI 1.3.2. Open MPI is an open source, freely available implementation of both the MPI-1 and MPI-2 documents. Our OS is Linux CentOS 5.4 x86_64, with an updated, unmodified kernel. We have two identical machines with two-socket Intel S5500BC motherboard, both sockets populated with Intel Xeon E5502 processors at 1.87GHz, with 16GB RAM. These are NUMA machines with the topology shown in Fig. 1, as found by the hwloc program [5]. As can be seen in the picture, each processor has two cores with private L1 and L2 cache but a shared L3 ("uncore") cache.

Our laboratory situation is depicted in Figs. 2a and 2b. We seek to compare performance of a set of four processes running on A) one two-sockets, two cores-per-socket machine, to B) a two-node cluster where only two cores on each machine (located on the same socket) will be used.

In both scenarios, four cores will be working and four processes will be run. One MPI process will run on each core. However, scenario A is a core-wise “denser” sce-

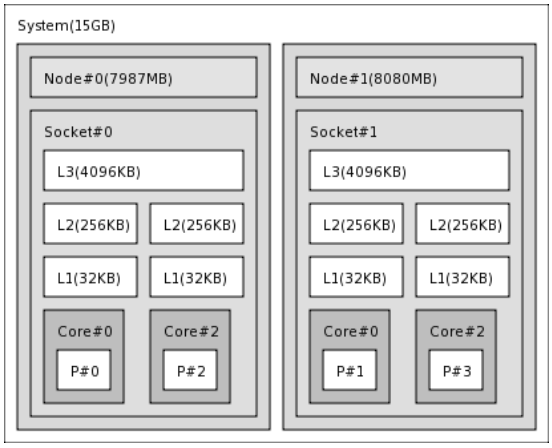


Fig. 1: Processor and memory architecture for the dual-socket system used as seen by hwloc.

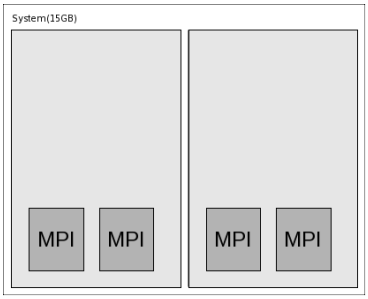
nario than B. As we are interested in evaluating the performance gain in switching from a less dense scenario to a denser one, we will design as "speedup" the ratio in execution time from scenario B to scenario A. We will compute this speedup for every program in the benchmark (EP, LU, BT, SP, MG, CG, FT, IS) and every problem size (A, B, C).

Both machines in scenario B, the two-node cluster, boot with only two cores (located in the same socket) online. This is achieved through the *isolcpus* kernel boot directive, thus realizing the situation desired in Fig. 2b. The same hardware is used in both scenarios to keep them as comparable as possible. However, in scenario B, both nodes are connected by a dedicated 1Gbps switched Ethernet network. The memory amount on each node is reduced to match the situation in scenario A (4GB per core). Again, this is achieved by using the *mem* kernel boot directive. The hwloc program is able to describe the new platform on each machine (Fig. 3).

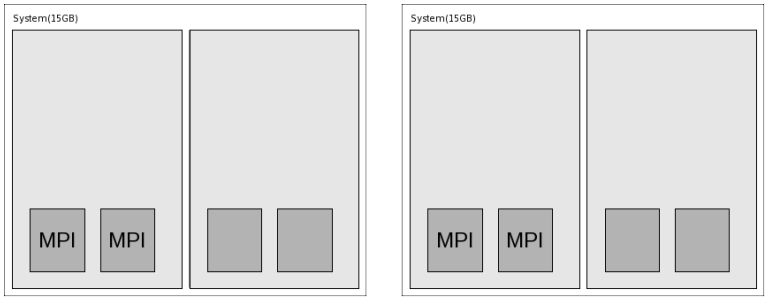
The benchmark is run on both scenarios. Our intuition tells us that scenario A should always excel B's performance, as the network link in B is orders of magnitude slower than its counterpart in A. The actual comparison results can be seen in Table 4a, with the ranking shown by the graph in Fig. 4b.

The best speedups in the test are at around 2.5. As we were expecting, none of the programs runs faster in scenario B than in A. However, as the picture reveals, some programs like EP, LU or BT attain speedups quite near to 1, i.e. there is no considerable gain in the passage from the clustered environment to the multicore machine. This may come as a surprise at first sight, as QPI interconnect specifications report a theoretical speed rate of some 250 times over 1Gbps Ethernet [15], as attested by our MPI bandwidth measurements (Fig. 5).

As Amdahl's Law predicts, such low speedups are explained by the nature of applications. Packet traffic and network bandwidth usage, as measured on scenario B, show a natural ranking of the programs (Fig. 6). In fact, this ranking is the same as in Fig. 4b: the lower the communication requirements, the lower the speedup from scenario B to



(a) Scenario A, one dual-socket system machine with four cores, running four MPI processes.



(b) Scenario B, a cluster with two dual-socket systems with four cores each, only two cores in each machine running MPI processes.

Fig. 2: Laboratory scenarios.

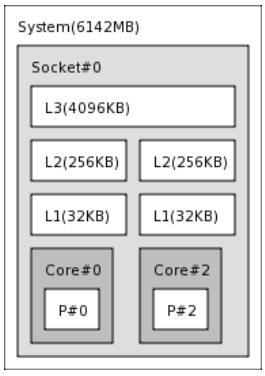
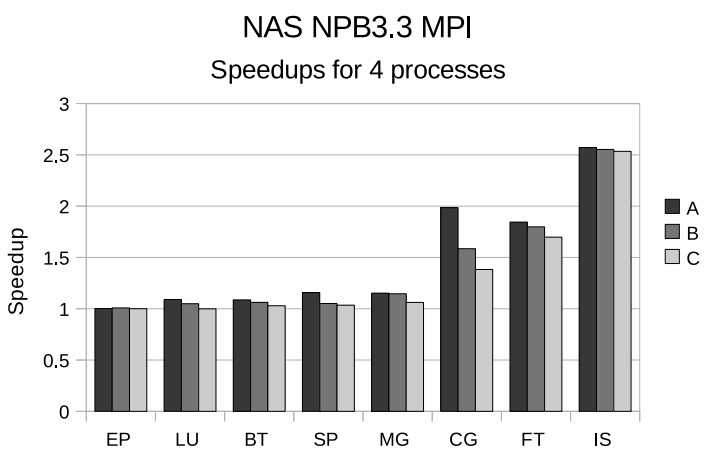


Fig. 3: Processor and memory architecture of one machine in the modified cluster, after setting offline two cores.

	A	B	C
EP	1	1.01	1
LU	1.09	1.05	1
BT	1.09	1.06	1.03
SP	1.16	1.05	1.04
MG	1.15	1.15	1.06
CG	1.99	1.59	1.38
FT	1.84	1.8	1.7
IS	2.57	2.55	2.53

(a) Execution time ratio from B to A



(b) Execution time ratio from B to A, graphically

Fig. 4: Comparison results for both laboratory scenarios. Speedups for NPB3.3 over MPI on four processes, four cores, from our Scenario B to Scenario A.

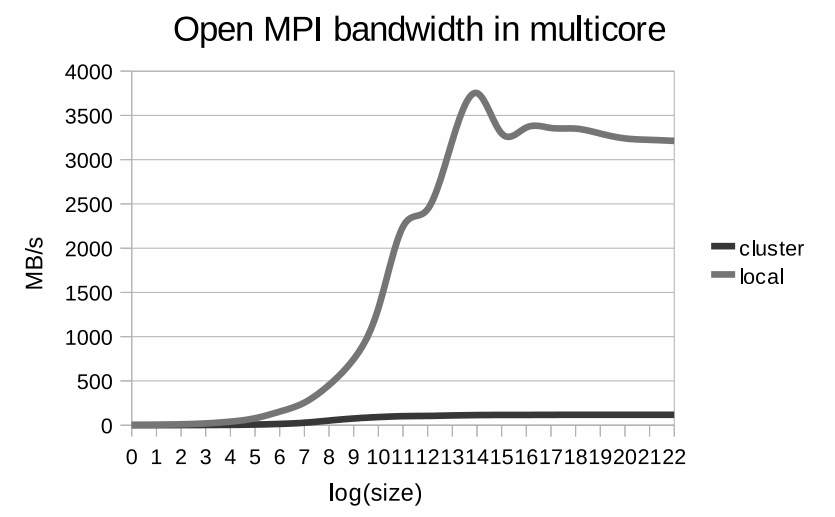


Fig. 5: Traffic bandwidth obtained for different packet sizes over QPI Interconnect (*local*) and over Ethernet link (*cluster*).

scenario A. On the lower end of the speedup ranking is EP, the Embarrassingly Parallel test. On the opposite end dwells IS, an Integer Sort application which is highly sensitive to latency.

In short, if our user knows that her applications have low network requirements, she may well consider not worth to acquire this particular core-dense hardware. She may prefer to keep her cluster as homogeneous as possible by distributing her investment over a greater number of lower-profile machines. On the contrary, if she knows that her applications do have greater bandwidth requirements, she will do right in considering migration to higher core-dense equipment.

Conclusions

We have run a well established benchmark to acquire a first impression of the behaviour of varied applications on new commodity hardware. This particular hardware was formerly unknown to us, and our tests confirm some intuitions and let us learn something new. In sight of our preliminar tests, we can recommend the user to know her applications and estimate her speedup needs before deciding for a growth strategy. Although multicore hardware looks like a promising path to upgrade (and the only one, for market reasons), her investment may be tuned to her needs by deciding about the number of cores. If her application is the "embarrassingly parallel" type, there will be no point in choosing denser hardware at higher costs, as these applications are highly scalable. On the other hand, if her application is heavily dependent on networking, a single machine may more than double two's performance, at the same total number of cores. Obviously, other considerations (room, power, thermal) are left out of this analysis.

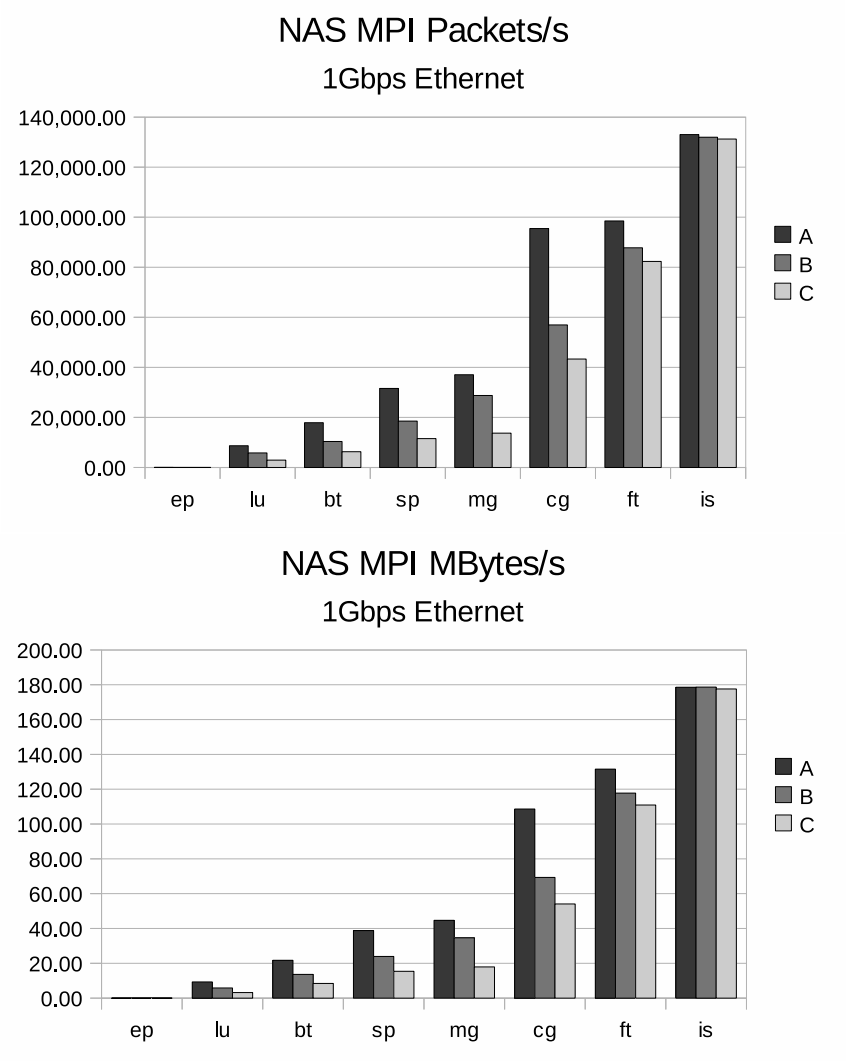


Fig. 6: Clustered NPB3.3 network traffic, packets per second and MegaBytes per second.

Our laboratory shows that there is a definite ordering in NAS NPB3.3 programs with respect to communication patterns. This ordering is quantitatively shown by statistics on packets per second and megabytes per second transferred. This can be taken as a starting point for comparison to other applications on a black box approach. This comparison will allow us to obtain a first-sight appreciation of how a new application can behave in hybrid clustered scenarios. The knowledge required about the application is as shallow as possible, i.e. it is next to our black box ideal.

Although program rewriting with involvement of threads programming will help to fully exploit the hardware, our laboratory shows us that MPI is an effective software platform that is still useful in the multicore cluster age "as is", with little or no impact in users' regular practices.

Future Work

While the present work considered only network bandwidth needs to characterize applications, finer performance models may allow us to generalize results across different platform architectures. We are currently studying profiling methods for parallel applications. We hope to use them in future work on modeling classes of applications for performance prediction. We are interested in selecting a general form of black box profile analysis that may give us a clearer picture of the communication pattern among all processes of a user's parallel application. This picture may serve to find a more accurate model for performance prediction that we can generalize to other topologies. The form of Amdahl's Law $f = a(s-1)/[s(a-1)]$ relates the program's enhanced fraction, f, to the amount of enhancement, a, and the speedup after enhancement, s. Given an arbitrary application, the program's enhanced fraction f, of communications, would be our first piece of evidence to model its performance.

Task partitioning, allocation and communication in parallel algorithms should follow the structure provided by the underlying architecture. Multicores offer new structural platforms to users building clusters, and the question about to what extent existing software may be optimized for new hardware, remains open. We are looking forward to take up further investigation in this domain.

References

1. Cómputo de altas prestaciones. http://hpc.uncoma.edu.ar/.
2. HyperTransport™ technology. http://www.amd.com/us/products/technologies.
3. Intel©QuickPath technology: Unleashing the performance. http://www.intel.com/technology/quickpath/.
4. NASA advanced supercomputing (NAS) division home page. http://www.nas.nasa.gov/.
5. Portable hardware locality (hwloc) documentation: v0.9.3. http://www.open-mpi.org/projects/hwloc/doc/v0.9.3/#examples.
6. Performance modeling of communication and computation in hybrid MPI and OpenMP applications. http://www.computer.org/portal/web/csdl/doi/10.1109/ICPADS.2006.81, 2006.
7. K. Asanovic, R. Bodik, B. C Catanzaro, J. J Gebis, P. Husbands, K. Keutzer, D. A Patterson, W. L Plishker, J. Shalf, S. W Williams, et al. The landscape of parallel computing research: A view from berkeley. *Electrical Engineering and Computer Sciences, University of California at Berkeley, Technical Report No. UCB/EECS-2006-183, December*, 18(2006-183):19, 2006.

8. J. Cai, A. P. Rendell, P. E. Strazdins, and H. J. Wong. Performance model for cluster-enabled OpenMP implementations. In *Proceeding of 13th IEEE Asia-Pacific Computer Systems Architecture Conference*, pages 1–8, 2008.
9. Jack Dongarra, Dennis Gannon, Geoffrey Fox, and Ken Kennedy. The impact of multicore on computational science software. *CTWatch Quarterly*, 3(1), February 2007.
10. U. Drepper. What every programmer should know about memory. *Eklektix, Inc., Oktober*, 2007.
11. G. Jost, H. Jin, D. an Mey, and F. F Hatay. Comparing the openmp, mpi, and hybrid programming paradigms on an smp cluster. In *Proceedings of EWOMP*, volume 3, 2003.
12. V. Kazempour, A. Fedorova, and P. Alagheband. Performance implications of cache affinity on multicore processors. *Lecture Notes in Computer Science*, 5168:151–161, 2008.
13. M. A Khan. Optimization study for multicores.
14. A. Kleen. A NUMA API for linux. *Novel Inc*, 2005.
15. Daniel Molka, Daniel Hackenberg, Robert Schone, and Matthias S. Muller. Memory performance and cache coherency effects on an intel nehalem multiprocessor system. In *2009 18th International Conference on Parallel Architectures and Compilation Techniques*, pages 261–270, Raleigh, North Carolina, USA, 2009.
16. R. Rabenseifner. Hybrid parallel programming on HPC platforms. In *proceedings of the Fifth European Workshop on OpenMP, EWOMP*, volume 3, pages 22–26.
17. L. Smith and M. Bull. Development of mixed mode MPI/OpenMP applications. *Scientific Programming*, 9(2):83–98, 2001.