INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

ITEA2 - 13024

The Concurrency and Locality Challenge

D3.7 COLOC state of the art

Editor:

Emmanuel Jeannot

Inria

Visibility	Restricted
Release	1.1
Date	21/10/2016
Authors	Inria, FOI, Bull, DA, UVSQ, Efieldl
Pages	19

History of Changes

Release	Author, Institution	Changes
1.0	Emmanuel Jeannot	First version
1.1	Q. Carayol, Q. Dinh (DA)	+ description of DA's CEM code + clarifications in section 6.2

Abstract

We present here the state of the art related to the different activities of the COLOC project: hardware, resource manager, performance modeling, etc.

Table of contents

1 Introduction

The ITEA 2 COLOC projects aims to provide simulation software developers with methodologies and tools to optimize their applications and HPC users to gain the most value from expensive and heterogeneous computing resources.

We here present the state of the art of the different parts of the project. In Section **2**, we survey the different hardware component of modern supercomputers. Programming models are surveyed in Section **3**. In Section **4,** we give an overview of performance and application modeling. Resources job management systems related to locality management are discussed in Section **5** and finally COLOC applications are shortly described in Section **6**.

2 Hardware

In COLOC we need to model multicore machine to be able to express and abstract locality.

2.1 NUMA Systems

The increasing number of cores accessing a shared memory at a same time has induced critical memory contentions. Non Uniform Memory Access (NUMA) architectures have been introduced to address this problem by providing separate memory for different set of cores. Each distributed compute node, i.e. NUMA node, has its own separate memory. Inside the NUMA nodes, each core has its own private Level 1 (L1) cache, and one or two additional cache levels shared with the other cores of the same processor. The main Random Access Memory (RAM) is shared among all the processors of the node. While data exchanges between distributed processes are handled through communications, every core within a shared memory space has a global vision of the main memory. However, the access times may vary a lot according to the proximity of the accessed data. Data locality is therefore more and more critical as the size of the NUMA nodes increases to reduce memory duplication and bandwidth contention.

Similarly, the increasing number of cores and larger caches within a same processor led to studies on Non Uniform Cache Access (NUCA) [40, 44]. In NUCA architectures, the cache is broken into smaller memory banks which can be accessed at different latencies. The D-NUCA design proposed in [40] has better scalability properties since accesses are serviced with different close banks.

Modeling the NUMA architecture is an important topic. Discovering all the computing and memory resources in computing platform has only been recently mastered with tools such as hwloc [8]. Former approaches were often less portable or do not expose as many details about cache sharing *etc.* MemAxes [31] offers fine-grained memory performance analysis with a graphical radial hierarchy display. However, it only focuses on static post-mortem analysis of memory accesses while our approach is dynamic and works for all performance metrics and more kinds of resource sharing. LIKWID [65] is a set of performance analysis tools that use advanced knowledge of the hardware topology. This knowledge is used for task placement while we also propose to combine it with performance monitoring for better analysis. LIKWID is actually complementary to our work, it will soon use hwloc for better topology discovery, while we may use LIKWID performance monitoring abilities when they will be exported as a C programming interface.

2.2 Manycore architecture

Recently, Intel has proposed with its Many Integrated Core (MIC) architecture as trade-off between GPUs and classical CPUs. The Intel Xeon Phi is the latest commercial release of the MIC architecture. Although it can be seen as an accelerator such as GPUs, it can be standalone and an Operating System (OS) can directly be installed on it. It is designed to exploit existing x86 parallel applications originally conceived for standard multicores. However, most applications designed for multicores will not have good performance when running on a Xeon Phi. The MIC architecture inherits many design elements from the Larrabee research project from 2008 [57]. A first prototype, called Knights Ferry (KNF), has been released to developers in 2010. In 2011, Intel officially released the Knights Corner (KNC) architecture under the Xeon Phi commercial name. It is composed of 61 cores clocked

at around 1 GHz and possesses 8 to 16 GB of GDDR5 memory depending on the version. Since the KNC can run an OS inside, a core is often used to service requests like interrupts and it may end up with 60 cores available for the user application. KNC proposes also large SIMD units of 512 bits and an adapted x86 Instruction Set Architecture (ISA) called Initial Many Core Instructions (IMCI). However, large access ranges involve more cache lines and therefore negatively impact the performance [16]. There is also support for masked instructions, where specific vector lanes can be excluded from an instruction. This KNC architecture has been integrated in many modern supercomputers present in the Top500 [64]. The Knights Landing (KNL) is the second architecture of this type. It will be composed of up to 72 Atom cores built at a 14 nm process size and will have up to 16 GB of 3D memory. It will still use 512 SIMD units but with the more generic AVX-512 ISA. On June 2016, a first version based on this architecture has been released.

2.3 Supercomputers

Supercomputers are large and powerful computational centers, a.k.a clusters. The first supercomputer was designed by Seymour Cray. At this time it was composed of a small number of processors and of specific vectorial computing units.

Supercomputers performance is measured in FLoating-point OPerations per Second (FLOPS). The 500 most powerful supercomputers around the world are ranked twice a year in lists made by the Top500 organization [64]. They are ranked according to their maximal performance, RMax, achieved using the High Performance LINPACK (HPL) benchmark [22] proposed by Dongarra *et al*. According to the last list of June 2016 the actual most powerful supercomputer is the Sunway Taihulight from China. It is composed of 10 649 600 cores coming from the Chinese Sunway processor. It reaches a maximal performance of 93.01 PFLOPS while consuming 15.4 MW.

The first supercomputer to reach the petascale, i.e. 10^{15} FLOPS, was the Roadrunner, built by IBM in 2008. The number of cores has then continued to grow until reaching the million with the IBM Sequoia supercomputer and more than three millions with Tianhe-2. Given the current speed of progress, supercomputers are projected to reach the exascale around 2023. However, mainly due to economic reasons, it is unlikely to reach the exascale by just increasing the number of parallel resources exploiting actual technologies. The projection of actual technologies used in Tianhe-2 or Titan into exascale machines leads to energy consumption of around 500 MW which is closed to the electric production of the first nuclear stations.

On behalf of system power other challenges are waiting supercomputer constructors. Memory are not keeping pace with the increase in flops, memory bandwidth and capacity per processor falling dramatically. The I/O system at all levels as well the parallel File System will need to be renewed to follow the machine speed. Data movement, both in energy consumed and in performance need to be minimized. Reliability and resiliency will be critical at the scale of billion-way concurrency...

3 Programming models

To make efficient use of the recent heterogeneous architectures presented previously, several parallelization strategies have to be combined together to exploit the multiple levels of parallelism. Usually at top level, there is the distributed memory parallelization model which consists in distributing the problem across a number of compute nodes with explicit message passing between them for synchronizing the application or exchanging data between the remote processes. The Message Passing Interface (MPI) presented in 3.1 has become the default implementation of this model. A recent alternative to message passing is the Partitioned Global Address Space (PGAS) model. It consists in splitting a memory region over distributed processes and providing them a direct remote access. At the lower levels, threads programming models (such as OpenMP) are used for programming shared memory systems. Other models such as task-based ones have also been proposed.

3.1 MPI Model - Message Passing Interface

To exploit the parallelism for distributed computing infrastructure, the MPI model has become a standard. Several MPI implementations exist such as MPICH [33], OpenMPI [28], or Intel MPI [38]. The idea of MPI is to provide an interface to parallelize an application by creating several processes, called *MPI ranks* or *tasks*, and to exchange data between them using messages. Each MPI rank is mapped to a distinct compute core. A subdomain part of the problem is assigned to each MPI rank.

However, MPI also handles shared memory resources to communicate between ranks on a same NUMA node. Furthermore, thread-based versions of MPI such as TMPI [61], Adaptive MPI (AMPI) [36], or MPC-MPI [51] have appeared. They map MPI ranks to threads instead of processes inside each NUMA node using process virtualization.

3.2 PGAS Model - Partitioned Global Address Space

A more recent approach used to exploit distributed memory parallelism is the *Partitioned Global Address Space (PGAS)* model. PGAS model consists in allocating a global memory space which is partitioned among the distributed processes resources. Each process has a local part of the segment and a direct access to the other remote parts of the segment. It is therefore possible to access, both in read and write, the memory of remote processes without their active involvement. These processes can be assigned to heterogeneous architectures.

For example, the GASPI API [34, 58] implemented in the GPI-2 library [24]. But many other PGAS languages exist such as Unified Parallel C (UPC) [11], Titanium [2], Co-Array Fortran [48], Chapel [13], or yet X10 [15]. MPI-RMA could also have been used for PGAS languages, but the strong restrictions on memory access pattern, the lack of semantic guarantees, and the absence of the remote completion concept restrain it [7].

3.3 OpenMP Worksharing Model

OpenMP [14] is a shared memory parallelization runtime using pragmas to create and manage threads. It first appeared in late 1990s and was initially published for the Fortran language and quickly after released for C and C++ languages. It is a bulk-synchronous fork-join model originally based on loop parallelization. This is done by the *omp parallel* pragma. To

parallelize loops, the *omp parallel for* pragma splits the loop iterations into independent chunks and schedules then among the different threads.

3.4 Cilk Plus

Cilk is a task-based runtime designed for multithreaded parallel programming and developed as an extension to the C programming languages [6, 26]. The C elision of a Cilk program produces a syntactically and semantically correct C code. Cilk was originally developed at the MIT in 1994 by Leiserson *et al*[26]. The name of Cilk is an allusion to the silk, which can be described as "nice threads", and to the C programming language. In 2006, Leiserson launched Cilk Arts and decided to modernize it into Cilk++ [43]. Cilk++ integrates the support for the C++ language, loop parallelization, and hyperobjects such as reducers or thread local storages named holders [25]. Since 2009, Cilk Arts is part of Intel Corporation and has been renamed into Cilk Plus.

4 Performance and application modeling

Performance analysis of a parallel program can be performed at different levels of granularity, from individual instructions up to the entire program. Analyzing the performance at the instruction level with tools such as MAQAO [4], PIN [46] or Intel VTune Amplifier is a part of the development and optimization process. Coarse-grain performance analysis still requires dynamic monitoring over time. It may involve real time tools such as `numatop` or tiptop [55], or offline temporal analysis with one of the existing tracing tools such as VampirTrace [41].

Multiple metrics may be used to diagnose topology-related performance issues, including memory link contention, cache conflicts, or computing unit shared accesses. Performance counters are the main solution for analyzing the behavior of codes and numerous tools are available such as PAPI [9] or the Linux `perf` utility. Other metrics exist for entire cluster-wide, such as congestion in network switches or links, which may be studied with tools such as SCALASCA [29] or Paraver [52].

A parallel application distributes its work among entities (MPI processes or threads) that run in parallel on the various physical computing units of the machine (processors or cores). These entities share data or exchange data through messages. The affinity between the processing entities is a useful abstraction to describe the fact that the performance of sharing data is more efficient when some pairs of processing entities are mapped closed to each other. Indeed, as parallel programs are running on NUMA systems with distributed memory nodes connected by a network communication, memory sharing costs depends on the location of these computing entities. Modeling this affinity is often done by a communication pattern matrix that describes the amount of data or the number of messages exchanged (or shared) between pairs [39].

5 Resource manager

In COLOC we extend the SLURM [68], to take into account locality constraints and application needs. The goal is to carefully map application taking into account energy, affinity and locality.

The idea of using the most adequate hardware resource to a specific application is not new and has been explored in previous work. It has been particularly popular in the context of grids environments ([45], [60], [56]) where it is important to select the best set of resources (clusters in this case) to use. Such work try to reduce the impact of WAN communication in grids but do not address the deeper details of the physical topology, such as NUMA effects or cache hierarchy for instance.

More recently, some works have targeted a specific type of applications, that is, MapReduce-based applications. For instance, the TARA [42] uses a description of the application to allocate the resources. However, this work is tailored for a very specific class of applications and does not address hardware details.

The mapping of a parallel application tasks to the physical processors based on the network topology can lead to important performance improvements [5]. Network topology characteristics can be taken into account by the scheduler [47] so as to favor the choice of group of nodes that are placed on the same network level, connected under the same network switch or even placed close to each other so as to avoid long distance communications. This kind of feature is taken into account by most of open-source and proprietary RJMSs. However even if most of them use the characteristics of the underlying physical topology, they eventually fail to take into consideration the application behavior when allocating resources and this is something that this work specifically addresses. HTCondor (formerly Condor) leverages a so-called *matchmaking* approach [53] that allows it to match the applications needs to the available hardware resources. However, the application *behavior* is not part of this matchmaking and HTCondor targets both clusters and networks of workstations. SLURM [68], as previously described, provides an option to minimize the number of network switches used in the allocation, so as to reduce the communication costs during the application execution (switches that are the deeper in the tree topology are supposed to be the less costly than upper ones). The same idea of topology-aware placement is exploited by PBS Pro [50], Grid Engine[49], and LSF [59]. Fujitsu [27] provides the same but only for its proprietary Tofu network. As far as our knowledge, SLURM [68] remains the only one providing a *best-fit* topology-aware selection whereas the others propose *first-fit* algorithms.

Some other RJMS offer task placement options that can enforce a clever placement of the application processes. That is the case of Torque [17] which proposes a NUMA-aware job task placement. OAR [10] uses a flexible hierarchical representation of resources which offers the possibility to place the application processes upon the hierarchy within the computing node. However, in these existing works, only the network topology is taken in account and the nodes internal architecture is left unaddressed when performance gains are expected from exploiting the memory hierarchy.

Jingjin Wu et al. in [66] introduced a hierarchical task mapping strategy for modern supercomputers based on generic recursive algorithms for both fat-tree and torus network topologies showing very good performance with low overhead. Rashti et al. [54] proposed a weighted graph model for the whole physical topology of the computing system, including both the inter and intra node topologies. Even if both previous related works have shown interesting results with application sets, they have not been integrated with real resource and job management system neither tested with real workload traces, which is our case in this paper.

A study for torus network topology [3] showed how processor ordering takes place based on space filling curve, such as Hilbert Curve, to map the nodes of the torus onto a 1-dimensional list in order to preserve locality information. This paper described the study about the allocation strategies implemented on the proprietary Cray Application Level Placement Scheduler (ALPS). Similar strategies, have been recently incorporated within SLURM with[1] (or without[2]) the use of ALPS. Another interesting work [67] adapted only for torus topology, presented a window-based locality-aware job scheduling strategy that tries to optimize job and system performance in the same time. Its goal is to preserve node contiguity by considering multiple jobs for scheduling while making use of the 0-1 Multiple Knapsack problem for resource allocation. The last 2 related works do not consider communication patterns as parameters within the algorithms.

Several binding policies are available, and they are compatible with the policies implemented in Open MPI. In all these solutions, the user has to retrieve the architectural details before submitting his job. Also, the placement options offered leave the user with the burden to determine his/her policy beforehand, and the application communication scheme is not taken into account.

In the COLOC project our goal is to improve this functioning on three levels: first, we take into account not only the network but also the node internal structure. The information used is based on the *structure* of the nodes and the memory hierarchy. In other words, we do not use latency and bandwidth figures to compute our allocation. Then, this information is retrieved directly by a plugin and does not have to be supplied by the user. All the technical details are hidden. Last, but not least, we also take into account not only the architecture but also the application behavior both for the allocation and the execution of a job.

[1] http://slurm.schedmd.com/cray_alps.html
[2] http://slurm.schedmd.com/cray.html

6 Applications

Here we first describe numerical methods (FEM, MoM, BEM and FMM) used in the applications that COLOC targets. And then we describe the applications themselves: AETHER, SPECTRE from DA, FOISOL from FOI and one from Efield.

6.1 Finite Element Methods

We describe here Irregular applications based on Finite Element Method (FEM) [21, 37], presented in and working on 3D unstructured meshes as it is one of the main COLOC applications. FEM applications commonly use the domain decomposition approach to generate data parallelism. Each process is assigned to the execution of a mesh subdomain and the communications between processes are usually handled by MPI. Unfortunately, most of the FEM applications currently in use exploit MPI domain decomposition both at distributed and shared memory levels. This results in increasing communication and memory bottlenecks. Indeed, in the FEM context, the large number of processes and subdomains induces data duplication of the frontier values, a.k.a halos, and a larger amount of communications.

In the context of finite element method, the main computational workloads consist of loops iterating over elements, nodes, or edges. These loops are non-trivial to parallelize in shared memory since they access large amount of data, generate many indirections, and bring several dependencies between iterations. In addition, most of the industrial HPC applications work on unstructured meshes with irregular geometries and variable number of neighbors, making their parallelization even more challenging.

6.2 Boundary Element Method, Method of Moments and Fast Multipole Method

In several domains of COLOC, applications such as electromagnetic or fluid mechanics, there is a need to solve linear partial differential equations formulated as integral equations. Boundary Element Method (BEM) is a numerical computation designed to solve such problems. One of the advantages of BEM is to be more interesting than FEM when the modeled domains become infinite. In BEM, the problem is transformed into an Algebraic form $Ax=B$ where x is the vector of unknown. The MoM[35] is a special case of BEM, which consists in applying the Galerkin method [23] (test functions and basis functions are identical).

Fast Multipole Method (FMM) is a particular solver used to accelerate the computation of MoM solutions. The FMM was first introduced by Greengard and Rokhlin [32]. The advantage of FMM over MoM solution by direct solvers is that it is faster and requires less memory. It is based on making a difference between far field and close field where far field interactions are approximated by using Green's function expansion into spherical harmonics, while close field interactions are computed exactly. Although the parallelization of FMM is not trivial, its hierarchical nature makes it a strong candidate for large-scale computing. Examples of parallel FMM library are PetFMM [18] or ScalFMM [1].

6.3 AETHER

For its routine aircraft design work, Dassault Aviation (DA) uses AETHER, a software developed in-house, which is a complete simulation environment for two- and three-dimensional CFD (Computational Fluid Dynamics) analysis. Based on sophisticated Finite Element Method (FEM, see [21, 37]) numerical formulations, the code solves the Navier-Stokes equations for compressible turbulent flows, discretized on fully unstructured meshes, which may be mixtures of numerous types of elements (triangles and quadrilaterals in 2-D; tetrahedral, bricks and prisms in 3-D). Started in the late 1980s and early 1990s, AETHER implementation (see [12]) relies heavily on mesh partitioning techniques for its distributed memory MPI- parallelization on clusters of SMPs (Symmetric Memory Processors). AETHER has also been ported to vector machines and shared-memory architectures, using multi-coloring techniques. In the COLOC project, updates of these shared-memory parallel implementations, based on the Divide and Conquer technique (see [63]), were successfully tested for modern multi-core and many-core architectures and will be gradually ported in AETHER.

6.4 FOISOL

FOI uses adaptive higher order finite element approaches (HOFEM) [19] in several disciplines, notably in computational structural mechanics (CSM) and in computational electromagnetics (CEM) applications. Most of the computing time is spent solving sparse symmetric linear equation systems Ax=b with huge dimensions N. Today N may be of order $O(10^9)$ unknowns for linear stress analysis in CSM, $O(10^6)$ degrees of fredom for nonlinear stress analyses. Challenging CEM scattering problems are today often of the order of $O(10^6)$ degrees of fredom. CEM analyses are linear in the harmonic case but the equation system is symmetric complex valued, which quadruples the number of operations compared to real arithmetic. FOISOL is a modularised version of the FOI FEM equation solver dealing with that size of equation systems coping with moderate resources of RAM and disk. While aeronautic CSM analyses concern stiffened shell-like structures (which have a thin direction), electromagnetic analyses usually involve solid threedimensional computational domains, possibly channel-like.FOISOL can handle multiple right-hand sides b which are provided in assembled form though.

6.5 Efield Software

The Efield software was developed during the late 90s in a research collaboration between Swedish defense industry, including Saab Aeronautics, Ericsson Microwave and the Swedish Defense Research Agency, and top universities, KTH, the Royal Institute of Technology, and Uppsala University. In 2006 the company Efield AB was founded for the commercialization of the software and was later acquired by ESI Group in 2012.

The current software suite consists of a range of different solvers: finite-element, finite-difference, boundary element and multi-pole with a common focus on full-wave analysis of electrically large problems in aeronautics, automotive and electromagnetic interference and compatibility (EMI/EMC).

In the COLOC project we address the parallel efficiency of four different solvers with related industrial applications where the current state-of-the-art in simulations does not fully meet the requirements: the multi-level fast multi-pole method (MLFMM) with applications in radar cross-section (RCS) and performance of installed antennas and sensors, the multi-

domain multi-method (MDMM) cavity tool for accurate RCS analysis of air-intakes and exhausts, the finite-element (FEM) solver for large closed domains at microwave frequencies and the finite-difference time-domain for broadband EMI/EMC analysis of complex systems.

6.6 SPECTRE

The SPECTRE Software is the in-house Dassault Aviation software for CEM. It has been developed since the late 80s, and includes a large number of methods and features. It is used routinely for antenna design and location on civil and military aircraft, and for RCS computation and stealth design on military aircraft. It is also used to compute acoustic propagation.

Some of the most commonly used features of SPECTRE are: asymptotic methods, Method of Moments on 2D or 3D multi-domain problems, use of geometric symmetries or quasi-symmetries, computation of periodic structures, high-order impedance boundary conditions, MLFMM solver, Domain Decomposition Methods, computation of structures above or buried in a PEC ground plane, design optimization using GA (Genetic Algorithms) or PSO (Particle Swarm Optimization), …

The Method of moments can use a highly efficient in/out of core direct solver on CPU and GPU architectures, or a multiple right-hand sides iterative solver with MLFMM acceleration and a hybrid MPI/OpenMP parallelism for shared and distributed memory architectures. In the COLOC project, DA focus is on the MLFMM solver, targeting many-core architectures where the memory size per core is dwindling.

7 Final Remarks

Parts of this document have been taken from different research papers and Ph.D dissertation published under the COLOC umbrella [62, 20, 30]. The complete list of contributors for this deliverable is: Erik Abenius, Quentin Carayol, Nicolas Denoyelle, Quang V. Dinh, Yannis Georgiou, Brice Goglin, Emmanuel Jeannot, Guillaume Mercier, Loïc Thébault, Adèle Villiermet and Adam Zdunek.

8 References

[1] Emmanuel Agullo, Bérenger Bramas, Olivier Coulaud, Eric Darve, Matthias Messner, and Toru Takahashi. Task-based fmm for multicore architectures. *SIAM Journal on Scientific Computing*, 36(1):C66–C93, 2014.
[2] Alex Aiken, Phil Colella, David Gay, Susan Graham, Paul Hilfinger, Arvind Krishnamurthy, Ben Liblit, Carleton Miyamoto, Geoff Pike, Luigi Semenzato, et al. Titanium: A high-performance java dialect. *Concurrency: Practice and Experience*, 10:11–13, 1998.
[3] Carl Albing, Norm Troullier, Stephen Whalen, Ryan Olson, Joe Glenski, Howard Pritchard, and Hugo Mills. Scalable node allocation for improved performance in regular and anisotropic 3d torus supercomputers. In *EuroMPI 2011, Santorini, Greece*, pages 61–70, 2011.
[4] Denis Barthou, Andres Charif Rubial, William Jalby, Souad Koliai, and Cédric Valensi. Performance Tuning of x86 OpenMP Codes with MAQAO. In Matthias S. MÃ¼ller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 95–113. Springer Berlin Heidelberg, 2010.
[5] Abhinav Bhatele, Eric J. Bohm, and Laxmikant V. Kalé. Topology aware task mapping techniques: an api and case study. In *PPOPP*, pages 301–302, 2009.
[6] Robert D Blumofe, Christopher F Joerg, Bradley C Kuszmaul, Charles E Leiserson, Keith H Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. *Journal of parallel and distributed computing*, 37(1):55–69, 1996.
[7] Dan Bonachea and Jason Duell. Problems with using mpi 1.1 and 2.0 as compilation targets for parallel language implementations. *International Journal of High Performance Computing and Networking*, 1(1-3):91–99, 2004.
[8] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In IEEE, editor, *PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, Pisa, Italie, February 2010.
[9] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, SC '00, Washington, DC, USA, 2000. IEEE Computer Society.
[10] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron, and Olivier Richard. A batch scheduler with high level components. In *Cluster computing and Grid 2005 (CCGrid05)*, Cardiff, United Kingdom, 2005. IEEE.
[11] William W Carlson, Jesse M Draper, David E Culler, Kathy Yelick, Eugene Brooks, and Karen Warren. *Introduction to UPC and language specification*. Center for Computing Sciences, Institute for Defense Analyses, 1999.
[12] F. Chalot, Q.V. Dinh, M. Mallet, A. Naïm, and M. Ravachol. A multi-platform shared- or distributed-memory navier-stokes code. In *Parallel CFD'97*, Manchester, UK, May 1997.
[13] Bradford L Chamberlain, David Callahan, and Hans P Zima. Parallel programmability and the chapel language. *International Journal of High Performance Computing Applications*, 21(3):291–312, 2007.
[14] Rohit Chandra. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.
[15] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph Von Praun, and Vivek Sarkar. X10: an object-oriented approach to non-uniform cluster computing. *Acm Sigplan Notices*, 40(10):519–538, 2005.
[16] Linchuan Chen, Peng Jiang, and Gagan Agrawal. Exploiting recent simd architectural advances for irregular applications. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, pages 47–58, 2016.
[17] Adaptive computing. Torque resource manager. http://docs.adaptivecomputing.com/torque/6-0-0/Content/topics/torque/2-jobs/monitoringJobs.htm.
[18] Felipe A Cruz, Matthew G Knepley, and Lorena A Barba. Petfmmâ€"a dynamically load-balancing parallel fast multipole library. *International Journal for Numerical Methods in Engineering*, 85(4):403–428, 2011.
[19] L Demkowicz, Jason Kurtz, David Pardo, M Paszyński, Waldemar Rachowicz, and Adam Zdunek. Computing with hp-adaptive finite element method. vol. ii. frontiers: Three dimensional elliptic and maxwell problems. *Chapmann & Hall/CRC Applied Mathematics & Nonlinear Science*, 2007.
[20] Nicolas Denoyelle, Brice Goglin, and Emmanuel Jeannot. A Topology-Aware Performance Monitoring Tool for Shared Resource Management in Multicore Systems. In Springer, editor, *Proceedings*

of Euro-Par 2015 – Parallel Processing Workshops: 3rd Workshop on Runtime and Operating Systems for the Many-core Era (ROME), Lecture Notes in Computer Science, Vienna, Austria, August 2015.
[21] Gouri Dhatt, Emmanuel Lefrançois, and Gilbert Touzot. *Finite element method*. John Wiley & Sons, 2012.
[22] Jack J Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: past, present and future. *Concurrency and Computation: practice and experience*, 15(9):803–820, 2003.
[23] Alexandre Ern and Jean-Luc Guermond. *Theory and practice of finite elements*, volume 159. Springer Science & Business Media, 2013.
[24] ITWM Fraunhofer. Gpi-global address space programming interface, 2013.
[25] Matteo Frigo, Pablo Halpern, Charles E Leiserson, and Stephen Lewin-Berlin. Reducers and other cilk++ hyperobjects. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 79–90. ACM, 2009.
[26] Matteo Frigo, Charles E. Leiserson, and Keith H. Randall. The implementation of the cilk-5 multithreaded language. *SIGPLAN Not.*, 33(5):212–223, May 1998.
[27] Fujitsu. Interconnect topology-aware resource assignment. http://www.fujitsu.com/global/Images/technical-computing-suite-bp-sc12.pdf.
[28] Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, et al. Open mpi: Goals, concept, and design of a next generation mpi implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 97–104. Springer, 2004.
[29] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The SCALASCA performance toolset architecture. In *Proc. of the International Workshop on Scalable Tools for High-End Computing (STHEC), Kos, Greece*, pages 51–65, June 2008.
[30] Yiannis Georgiou, Emmanuel Jeannot, Guillaume Mercier, and Adèle Villiermet. Topology-aware Resource Management for HPC Applications. In *18th International Conference on Distributed Computing and Networking (ICDCN 2017)*, Hyderabad, India, January 2017. To be published.
[31] Alfredo Gimenez, Todd Gamblin, Barry Rountree, Abhinav Bhatele, Ilir Jusufi, Peer-Timo Bremer, and Bernd Hamann. Dissecting On-Node Memory Access Performance: A Semantic Approach. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, pages 166–176, New Orleans, LA, November 2014. IEEE Press.
[32] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.
[33] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.
[34] Daniel Grünewald and Christian Simmendinger. The gaspi api specification and its implementation gpi 2.0. In *7th International Conference on PGAS Programming Models*, volume 243, 2013.
[35] Roger F Harrington and Jan L Harrington. *Field computation by moment methods*. Oxford University Press, 1996.
[36] Chao Huang, Orion Lawlor, and Laxmikant V Kale. Adaptive mpi. In *Languages and Compilers for Parallel Computing*, pages 306–322. Springer, 2003.
[37] T.J.R. Hughes. *The Finite Element Method*. Prentice-Hall Inc., 1987.
[38] Intel mpi library.
[39] Emmanuel Jeannot, Guillaume Mercier, and François Tessier. Process placement in multicore clusters: Algorithmic issues and practical techniques. *Parallel and Distributed Systems, IEEE Transactions on*, 25(4):993–1002, 2014.
[40] Changkyu Kim, Doug Burger, and Stephen W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. *SIGARCH Comput. Archit. News*, 30(5):211–222, October 2002.
[41] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S. Müller, and Wolfgang E. Nagel. The vampir performance analysis tool-set. In *Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing, July 2008, HLRS, Stuttgart*, pages 139–155, 2008.
[42] Gunho Lee, Niraj Tolia, Parthasarathy Ranganathan, and Randy H. Katz. Topology-aware resource allocation for data-intensive workloads. In *APSys '10*, pages 1–6, 2010.
[43] Charles E Leiserson. The cilk++ concurrency platform. *The Journal of Supercomputing*, 51(3):244–257, 2010.

[44] Javier Lira, Carlos Molina, and Antonio González. Analysis of non-uniform cache architecture policies for chip-multiprocessors using the parsec benchmark suite. In *Proceedings of the workshop on managed many-core systems*, pages 1–8, 2009.
[45] Chuang Liu, Lingyun Yang, Ian Foster, and Dave Angulo. Design and evaluation of a resource selection framework for grid applications. In *HPDC '02*, pages 63–, 2002.
[46] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, pages 190–200, New York, NY, USA, 2005. ACM.
[47] Javier Navaridas, José Miguel-Alonso, Francisco Javier Ridruejo, and Wolfgang Denzel. Reducing complexity in tree-like computer interconnection networks. *Parallel Computing*, 36(2-3):71–85, 2010.
[48] Robert W. Numrich and John Reid. Co-array fortran for parallel programming. *SIGPLAN Fortran Forum*, 17(2):1–31, August 1998.
[49] Oracle. Grid engine.
[50] PBSWorks. Pbs. http://www.pbsworks.com/PBSProduct.aspx? n=PBS-Professional&c=Overview-and-Capabilities.
[51] Marc Pérache, Patrick Carribault, and Hervé Jourdren. Mpc-mpi: An mpi implementation reducing the overall memory consumption. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 94–103. Springer, 2009.
[52] V. Pillet, J. Labarta, T. Cortes, and S. Girona. PARAVER: A Tool to Visualize and Analyze Parallel Code. In Patrick Nixon, editor, *Proceedings of WoTUG-18: Transputer and occam Developments*, pages 17–31, mar 1995.
[53] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *HPDC'7*, Chicago, IL, July 1998.
[54] Mohammad J. Rashti, Jonathan Green, Pavan Balaji, Ahmad Afsahi, and William Gropp. Multi-core and network aware MPI topology functions. In *EuroMPI 2011, Santorini, Greece*, pages 50–60, 2011.
[55] Erven Rohou. Tiptop: Hardware Performance Counters for the Masses. Research Report RR-7789, Inria, November 2011.
[56] Cipriano A. Santos, Akhil Sahai, Xiaoyun Zhu, Dirk Beyer, Vijay Machiraju, and Sharad Singhal. *Policy-Based Resource Assignment in Utility Computing Environments*, pages 100–111. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
[57] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, et al. Larrabee: a many-core x86 architecture for visual computing. *ACM Transactions on Graphics (TOG)*, 27(3):18, 2008.
[58] Christian Simmendinger, Mirko Rahn, and Daniel Gruenewald. The gaspi api: A failure tolerant pgas api for asynchronous dataflow on heterogeneous architectures. In *Sustained Simulation Performance 2014*, pages 17–32. Springer, 2015.
[59] C. Smith, B. McMillan, and I. Lumb. Topology aware scheduling in the lsf distributed resource manager. In *Proceedings of the Cray User Group Meeting*, 2001.
[60] O. Sonmez, H.H. Mohamed, and D.H.J. Epema. Communication-aware job placement policies for the koala grid scheduler. In *e-Science'06*, pages 79–86, Dec 2006.
[61] Hong Tang and Tao Yang. Optimizing threaded mpi execution on smp clusters. In *Proceedings of the 15th international conference on Supercomputing*, pages 381–392. ACM, 2001.
[62] Loïc Thébault. *Algorithmes parallèles efficaces appliqués aux calculs sur maillages non structurés*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines (UVSQ), 2016.
[63] Loïc Thebault, Eric Petit, Marc Tchiboukdjian, Quang Dinh, and William Jalby. Divide and conquer parallelization of finite element method assembly. *Parallel Computing: Accelerating Computational Science and Engineering (CSE), Advances in Parallel Computing 25*, 2014.
[64] Top500 lists november 2015.
[65] Jan Treibig, Georg Hager, and Gerhard Wellein. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In Wang-Chien Lee and Xin Yuan, editors, *ICPP Workshops*, pages 207–216. IEEE Computer Society, 2010.
[66] Jingjin Wu, Xuanxing Xiong, and Zhiling Lan. Hierarchical task mapping for parallel applications on supercomputers. *The J. of Supercomputing*, 71(5):1776–1802, 2015.
[67] Xu Yang, Zhou Zhou, Wei Tang, Xingwu Zheng, Jia Wang, and Zhiling Lan. Balancing job performance with system performance via locality-aware scheduling on torus-connected systems. In *Cluster'2014*, pages 140–148, 2014.

[68] AndyB. Yoo, MorrisA. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.