

# Parallel performance of CFD applications and the ubiquitous need for HPC with high fidelity, multidisciplinary analysis and optimization (MDO)

Mark Kremenetsky\* and Srinivas Kodiyalam\*\*

\* Silicon Graphics Inc. (SGI), Sunnyvale, CA 94085, USA  
(Email: [mdk@sgi.com](mailto:mdk@sgi.com))

\*\* Silicon Graphics Inc. (SGI), Sunnyvale, CA 94085, USA  
(Email: [skodiyal@sgi.com](mailto:skodiyal@sgi.com))

---

**Abstract:** We discuss possible ways to improve CFD application software performance by highlighting key features of the HPC clusters, including, multi-core processor performance, host channel adapters (HCA), multi-rail networks, and message passing (MPI) implementations and more as well as briefly highlight the ever-growing need for HPC for solving computing intensive MDO problems.

*Keywords:* HPC, CFD, MDO, clusters, multi-core, multi-rails, MPI, parallel scalability

---

## 1.0 INTRODUCTION

Parallel processing on high performance computing environments has enabled much larger and complex CFD problems to be addressed [1, 2]. It is noteworthy that while High Performance Computing (HPC) has indeed facilitated solutions to many of the complex CFD simulations and the growth in the usage of CFD, the application of most detailed approaches (such as, direct numerical simulations (DNS) of the Navier-Stokes equations) for industry standard vehicle configurations with typical operating conditions are still beyond today's computing capability.

It is well known that the cost of software development exceeds hardware upgrade costs in later years of the product life cycle. Besides, a "simplistic" usage of the newer generation hardware does not provide the expected or desired level of performance improvement and very often the engineer or scientist is unaware of the possible hardware options that can be employed to improve performance. This is simply due to the information gap between the engineering users, who are typically experts in their technical disciplines/applications, and the hardware specialists who are well versed in the hardware usage.

In this paper we will discuss possible ways to improve the application software performance (i.e. CFD codes) by highlighting key features of the HPC clusters, including, multi-core processor performance, host channel adapters (HCA), multi-rail networks, message passing (MPI) implementations and such as well as the ever-growing need for HPC for solving computing intensive MDO problems with high fidelity analysis codes, such as CFD and Structures.

## 2.0 PERFORMANCE OF CFD APPLICATIONS

Majority of the CFD applications, with few exceptions, is implemented in a distributed memory paradigm and rely heavily on the use of explicit message passing technology (MPI). The driving force behind such an approach is the desire to reach global portability over all possible computer architectures. The way the cluster nodes are connected together has a significant influence on the overall application performance, i.e. the cluster interconnect is critical to the efficiency and scalability of the entire cluster, as it needs to handle the I/O requirements from each of the CPU cores without imposing networking overhead on the computational performance. In multi-core processor architectures, the performance and scalability bottlenecks have shifted to I/O and memory configurations.

### 2.1 Performance on a single node

2.1.1 Multi-core processor performance: For a very long time, the primary source of performance improvement in new processors was expected from a higher clock rate but this feature did not fully address the performance expectations. Chip designers not only face physical layout problems in further increasing the clock rate but also other challenges including specific requirements of application codes such as those related to memory referencing. The latter includes memory latency, cache utilization and memory bandwidth. Application codes require well balanced processors – not only pure computational power but also mechanism by which the data is delivered. Classification of an application code/algorithm based on bandwidth requirements is well known with the two most popular categories being the bandwidth hungry and latency driven algorithms. The typical measure of a bandwidth

driven algorithm is the well known Triad benchmark while the second category will include any algorithm that relies on sparse data sets. However, this is not a very accurate classification and in reality all computational algorithms are bandwidth hungry. The difference is in the time pattern of memory requests- the first group shows a continuous stream of memory transfer while with the second category we have a discontinuous pattern where high bandwidth areas look like spikes on time axis. This means that practically all algorithms will benefit from processors with a higher memory bandwidth. For multi-core processors, it becomes more complicated because we have to pay attention to the complete data path from memory to cache to computational units.

The following example is using FLUENT CFD application code, from ANSYS Inc., based on unstructured grid discretization that in turn defines the sparse nature of the main data structures. This example is used to illustrate the multi-core processor performance with 2 Intel Clovertown sockets in a node. The Intel Clovertown processor is a quad core processor that consists of two dies, each of those dies contains 2 computational cores and one shared L2 cache. We ran 4 threads computation on a single node that contains 2 Clovertown sockets using various threads to core allocation schemes and the results are provided below:

Table 1: Threads to core allocation

	4 threads/socket; shared L2 cache; 1 FSB	2 threads/socket; shared L2 cache, 2 FSB	2 threads/socket; non shared L2 cache; 2 FSB
Bandwidth (MB/sec)	550/1.0	504/1.09	426/1.29
Time (sec)	46/1.0	39/1.18	31/1.48

The FSB for this particular chipset ran at 1333 MHz so hardware bandwidth is about 10.6 GB/s. As seen, the average bandwidth consumed by the CFD application is much lower than available hardware bandwidth but nevertheless measured timings are quite different. Our interpretation is that using the optimal thread allocation with non-shared L2 cache results in a much lower number of memory requests and as such the memory consumption spike is much lower and that leads to a better application performance over all.

The bandwidth related problem that are specific to the current generation of multi-core processors are extensively resolved with the new Intel quad-core processor, Nehalem. Nehalem is one of the new processors that provide for a better balance between computational units and the memory path characteristics.

1. The Nehalem socket has a memory controller device directly on the die that allows doubling the available bandwidth. Depending on the memory type, the hardware bandwidth is in the range of 20GB/s - 35 GB/s;
2. Each core has its own L1 and L2 cache to substantially decrease the number of stalls in a data path;
3. The data pre-fetch algorithm for L2 and L3 caches has been reworked to achieve more effective data load.

The above improvements have contributed to substantial improvement in application performance over the previous generation of processors.

Following is a comparison of FLUENT standard benchmark suite performance on a node with 2 Harpertown quad-core sockets/processors running at 3GHz and a second node with 2 Nehalem quad-core processors at 2.66GHz. The performance metrics used in this study is the same rating number used by ANSYS that represents the number of jobs completed in a 24-hour time frame. Hence, a higher number represents better performance.

Table 2: Comparison of Intel Harpertown to Nehalem processors

# of cores (threads)	Aircraft (2M cells) Harpertown/Nehalem	Eddy (417K) H/N	Sedan (4M) H/N	Turbo (500K) H/N
1	94/134	110/138	61/105	444/617
2	183/266	218/268	122/213	866/1201
4	275/493	389/499	185/397	1400/2223
8	312/812	540/881	217/676	1835/3927

As seen from the above results, Nehalem processor with a lower clock rate performs significantly better than Harpertown processor. It is noteworthy that performance for larger models is more sensitive to the memory path characteristics of a particular processor and even more significant is the improvement with more number of threads. Most interesting is the performance comparison for a fully populated node. With 8 threads, Harpertown processor certainly hits a road block in memory path limit however the Nehalem processor performance is unaffected.

## 2.2 Factors influencing parallel scalability

It is well known that interconnect parameters (e.g. latency and bandwidth) are critical to the efficiency and scalability of complex cluster systems. Interconnect is critical for handling all communications requirements, transferring information from node to node without imposing a heavy overhead. Several publications have demonstrated the needs for a superior interconnect. It has been demonstrated that the low-bandwidth and high-latency characteristics of Gigabit Ethernet (GigE) make it a performance bottleneck, limiting the overall cluster efficiency and preventing any performance gain beyond four or five compute nodes. At the same time, an InfiniBand (IB) connected cluster has demonstrated higher performance (rating) than the GigE-based configuration [3]. Presently, Infiniband architecture provides for the most advanced interconnect for I/O and inter-process communications. High performance MPI implementations over Infiniband are also available. While a notable feature of Infiniband is its high bandwidth, the network bandwidth can still prove to be the performance bottleneck in some of today's most demanding applications. This is especially the case for clusters built with SMP nodes, in which multiple processes may run on a single node and must share a node bandwidth.

2.2.1 Host Channel Adapters (HCAs): Two ways to overcome bandwidth limitations are the use of more sophisticated Host Channel Adapter (HCA) cards and multi-rail networks. In this section, we highlight the importance of Host Channel Adapter (HCA) cards for servers and clusters with multi-core processors. A case study using FLUENT CFD code is performed on 2 pairs of node. The first pair of nodes is using IB HCA InfiniHost3 while the second pair of nodes is with IB ConnectX. The round robin allocation scheme is used so if we run on 2 cores then thread 0 is allocated on node 0, thread 1 is allocated on node 1. If we run with 4 threads, then threads 0 and 2 are allocated on node 0 and threads 1 and 3 get allocated on node 1 and so on. We intentionally chose a very small case (32,000 cells) so cost of communications will be quite high. Following are the timings from running the solver for 100 steps:

Table 3: Comparison of HCAs

# of cores (threads)	InfiniHost3	ConnectX	Speedup
2	9.605	8.214	1.17
4	5.687	4.351	1.31
8	3.368	2.560	1.32
16	6.504	1.812	3.59

It is obvious that IB ConnectX provides much better performance on a high number of threads. It is also noteworthy that it may be much cheaper to upgrade only the HCA cards on a cluster then rewrite the code of interest.

2.2.2 Multi-rail Network: Another interesting and useful feature of multi-core Infiniband clusters is the multi-rail network. Multi-rail networks can improve MPI (Message Passing) communication performance by distributing the communication traffic to multiple independent networks (rails). For multi-rail, the MPI library has to manage the network traffic between the available separate network fabrics.

The primary intent of having multiple network on a system is to separate communication and IO activities. However, several applications have limited IO activity and their performance may be limited by communication costs; so the choice of multiple networks seems very attractive towards improving communication costs in such application codes. One good example is the CFD steady state solvers. The main task of a steady state CFD solver is to arrive at a converged solution in the shortest possible time. Intermediate results are not of much interest so one will not necessarily save those results. With steady state codes, the primary IO activity typically happens in the initial and final phases of simulation so the main part of simulation time does not involve much IO activity.

The FLUENT standard benchmark suite, using FLUENT v12.0.9, running on an Altix ICE cluster is used to understand multi-rail network contribution towards improving the application performance. The cluster configuration is a 256 node Altix ICE+ (8200EX) cluster with 3.0GHz/12M/1600MHz 120W quad-core/node (x5472 Harpertown/Seaburg chipset) and 2GB/core memory. And, the performance metric is the same rating number used by ANSYS that represents the number of jobs completed in a 24-hour time frame. Hence, a higher number represents better performance in Table 4.

For smaller models, in general a single-rail network provides for sufficient bandwidth and there is little improvement in parallel scaling with dual-rail network. However, for larger cases, we see a significant difference between single-rail and dual-rail performance numbers beyond 512 cores. With a single-rail network, communication cost after 256 cores becomes expensive and we start to see negative scaling. At the same time dual-

rail network has enough bandwidth capacity to sustain a reasonable growth of parallel performance expansive. Indeed, it is worth using multi-rail features for large scale simulations and at high core counts.

Table 4: Single-rail/Dual-rail performance rating comparison

nprocs	Turbo (0.5M cells)	Eddy (417K cells)	Aircraft (2M)	Sedan (4M)	Truck (14M)	Truck_poly (14M)
16	3680/3790	1134/1130	590/630	440/440	69/70	70/70
32	7513/7697	2160/2152	1209/1180	804/905	140/139	138/135
64	13880/14281	3516/3473	1691/2066	1788/1784	281/281	253/268
128	14706/14460	4867/4916	2551/4414	3435/3442	556/540	440/500
256	18885/19750	6484/6496	4179/5647	5477/5494	1009/1019	869/853
512		4028/6912	1837/2171	4661/7464	1006/1546	858/1842
1024			901/1353	692/862	803/2796	772/2374

2.2.3 Large Model Parallel Scalability: A large truck model of 111M cells of mixed type is used to understand the benefits of multi-rail network and appropriate tuning of the hardware features. The problem involves external flow over the truck body and uses DES model with the segregated implicit solver. The results that are presented below are using FLUENT Version 12.0.9 on Altix ICE 8200EX (3GHz, quad-core) can also be seen at the ANSYS benchmark website at: [http://www.fluent.com/software/fluent/fl6bench/fl6bench\\_6.4.x/problems/truck\\_111m.htm](http://www.fluent.com/software/fluent/fl6bench/fl6bench_6.4.x/problems/truck_111m.htm)

Table 5: External flow over a truck, FLUENT v12.0.9, Altix ICE 8200EX scaling

CPUs of Cores	Machines or Blocks	Rating
128	16	60
256	32	124.1
512	64	244
1024	128	464.3
2048	256	741

The performance metric is the same rating number used by ANSYS that represents the number of jobs completed in a 24-hour time frame. Hence, a higher number represents better performance.

### 3.0 CONSIDERATIONS FOR FUTURE WORK

For parallel scalability a major contributing factor therefore is the performance of a particular MPI implementation on any particular computer architecture. A common belief is that the major part of parallel performance is defined by the quality of point-to-point communications. In general it is a valid statement but translating it to the network requirements can be non trivial. A simple way to state this is as follows: For a one level algorithm where all computations are done on one computational grid, the size of inter domain messages is defined by a size of domain faces and because it is typically quite large it is dependent on the network bandwidth. On the other hand for multi-level solvers (e.g. multi-grid algorithms) a large part of the communications is performed with small size messages and therefore depends mostly on network latency. However, such requirements will depend also on the problem size, number of nodes and other parameters.

In order to get an understanding of the communication scenario, performance instruments are wrapped around MPI calls that report the time spent in a specific communication primitive at runtime. Below is an example of MPI session timing taken from a CFD run on 32 cores (8 nodes):

*MPI\_Allreduce* 53.1 sec; *MPI\_Waitall* 13.2 sec; *MPI\_Send* 7.2 sec; *MPI\_Recv* 8.3 sec; *MPI\_Barrier* 5.2 sec  
*MPI Time* 112.3; *Wall Time* 374.4

Clearly, a large part of time is spent in the collective operator *MPI\_Allreduce*. However, when we instrument the same example with a tool (*MPIinside*) that uses an explicit synchronization call just before *MPI\_Allreduce* we observe the following: *MPIinside statistics*: *b\_Allred* 53.3446 sec; *Allred*: 5.6589 sec

Here *b\_allred* is the time to synchronize and *allred* is the time spent by the collective operator itself. As we can see the actual cost of collective operations is lower than point-to-point communication primitives but the implicit

synchronization time which is required for completion of collective operations is an expensive part of the overall communication cost. This application has almost ideal load balance measured by number of cells per parallel domain so the computation time scales well and according to Amdahl's law the overall scaling should also be good. However, we can see that a code will wait in certain specific points for implicit synchronization which we will refer to as micro-imbalance. The reasons for this imbalance can be several including that the actual amount of compute work is defined not only by number of computational cells but also through the variations of the physical models, variations of computational domains like internal and boundary domains, fluctuations in HCA loads, etc...The elimination of micro-imbalance can be a major factor in the future improvement of parallel scalability. One approach that can be utilized is the new MPI protocol (MPI-2) that introduces a new class of communication primitives called one-sided communications, example of such primitives being PUT and GET. Such an approach relies on large shared memory windows and minimal amount of direct copy operations which in turn leads to much better MPI performance. Using one-sided communications, one can observe the increase in bandwidth and even more impressive is the decrease in latency by a factor of about 3X. Another wonderful feature of one-sided communications is the asynchronous nature of communication algorithms based on those primitives.

The above performance enhancements in single discipline CFD codes allow engineers not only to perform more sophisticated and realistic fluid dynamic analyses but also permit the use of high fidelity CFD codes as an integral part of more complex design processes like Multidiscipline Analysis and Optimization (MDO) [4]. For example, a basic aircraft design does not require much computational power, however, in order to be more precise and model a more refined shape, 100's of design variables will be required to represent the variation in the shape of the airfoil at many stations along the wing as well as details of the internal structure. Clearly, this would call for a high fidelity CFD analysis of the air flow, corresponding to each change in design with the MDO process. The need for HPC is therefore ubiquitous for several reasons including: faster turn around times of solutions to complex design problems; dealing with ever-growing model sizes and more complete models such as in rotor-stator interactions in a turbine; handling more complex physics such as in combustors and Large Eddy Simulations; and, enabling formal MDO solutions in a timely manner to impact product design cycle times.

Key HPC requirements to systematically and rigorously investigate a large design space for MDO include:

- Balanced HPC environments to support multidiscipline analysis: The heterogeneous mix of simulations common to a MDO process tends to exhibit a range of HPC resource demands. For instance, the implicit structural analysis solvers for dynamic responses requires high rates of memory and IO bandwidth with processor speed as a secondary concern while explicit dynamics solvers for impact crash simulation benefits from a combination of fast processors for the required element force calculations and a high rate of memory bandwidth necessary for efficient contact resolution. CFD also requires a balance of memory bandwidth and fast processors, but benefits most from parallel scalability.
- High throughput efficiency: Most vehicle systems are multidisciplinary involving a heterogeneous mix of analysis codes, including high fidelity analyses codes; High throughput efficiency on a multiprocessor system allows for fast turnaround times of multiple jobs, enabling many runs to be made concurrently in a short amount of time as required in the construction of the approximation (surrogate) models for MDO [5].

For today's economics, these HPC resources such as CPU cycles, memory, system bandwidth and scalability, storage, file and data management – must deliver the highest levels of engineering productivity and HPC reliability that is possible from a platform environment.

#### 4.0 SUMMARY

Parallel processing on high performance computing clusters is enabling much larger and more complex CFD problems. However, in order to realize these performance gains, a knowledgeable use of the latest generation of HPC clusters with multi-core processors, multi-rail networks, MPI and such developments is critical. This paper details possible ways to improve CFD code performance on HPC clusters and the standard benchmark suite of the industry standard CFD code, FLUENT, is used in illustrating the performance improvements. Further, this paper addresses the ubiquitous need for HPC with high fidelity, multidisciplinary analysis and optimization.

#### REFERENCES

1. A. Jameson and M. Fatica, "Using Computational Fluid Dynamics for Aerodynamics," Stanford University, Stanford, California, 2006.
2. S. Kodiyalam, M. Kremenetsky and S. Posey, "Balanced HPC Infrastructure for CFD and associated Multidiscipline Simulations," Proceedings, 7<sup>th</sup> ASIAN CFD Conference, Bangalore, India, Nov 2007.
3. G. Shainer, S. Kher, & P. Alavilli, "Interconnect Helps Efficiency in Clusters," Desktop Engineering, Dec 2008.

4. J. S. Sobieski and R. T. Haftka, "Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments," *Structural Optimization*, pp. 1-23, Vol. 14, No. 1, August 1997.
5. S. Kodiyalam and J. S. Sobieski, "Bi-Level Integrated System Synthesis with Response Surfaces," *AIAA Journal*, Vol. 38, No. 8, pp. 1479-1485, August 2000.