Multicore Clusters for CFD Simulations Comparative Study of Three CFD-Softwares

A. de Blanche, N. Namaki, S. Mankefors-Christiernin
Division of Automation and Computer Engineering
University West, Trollhättan, Sweden

Abstract

Multicore processors have come to stay, fulfill Moore's law and might very well revolutionize the computer industry. However, we are now in a transitional period before the new programming models, numerical algorithms and general computer architecture have been developed and the software has been rewritten. This paper focuses on the effects multicore based systems have on industrial computational fluid dynamics (CFD) simulations. The most significant finding was that five of the models ran faster when only one process was executed on each multicore node instead of two. In these cases the execution time was increased by between 6.5% and 64% with a median increase of 10% when utilizing both cores.

Keywords: multicore, cluster, non-uniform inter-process communication

1 Introduction

Getting the most out of any computer system, and especially a new architecture, has always been a challenging task. When new hardware technologies are introduced and put to use it is not obvious that the application performance will increase – even if the technology is immensely superior. The existing applications have been optimized to run on the current systems and they might not at all be suitable for the next-generation high-performance computers. The latest major addition is the introduction of multicore systems.

As suggested by James Peery et. al. [1], at Sandia national laboratory, the execution time might increase if more cores are utilized. Using Sandia key algorithms they showed that when using sixteen cores the performance was barely on par with two cores. According to Arun Rodrigues this is an issue to which the industry has no known solution and that often is ignored [1]. The addition of a second core could theoretically double the computing capacity of a computer node [2], however in many cases the bottleneck moved to another component. As pointed out by [3] [4] [5] the most notable is the processor versus memory size and bus ratio, both when it comes to latency as well as bandwidth.

Traditionally high performance computer designs have tried to balance three factors [6]: Compute power, memory and I/O-capacity. According to Vaughan et. al. [7] the new generation of COTS distributed parallel computers (clusters) [8] [9] has added inter-node communication capacity as a crucial factor. For Multicore processors Vaughan's addition should be split into two, namely intra-node and inter-node communication capacity due to the non-uniform inter-process communication architecture.

In this paper we present a, first study on the effects multicore based systems have on industrial computational fluid dynamics (CFD) simulations. Do multicore processors facilitate a decrease in execution time within the CFD area or will the already long execution times increase? During our investigation we found that for five of our nine models there were configurations in which the execution times increased by 6.5% to 63% when utilizing two cores in each processor instead of one. Six configurations paged to the hard drive due to lack of memory and were excluded.

2 Multicore Architecture and Issues

Typical modern computer architecture is based on a microprocessor, system memory, busses and various other components. Very much like the schematic overview in figure 1. All information that goes from the main memory to the processor has to pass through the system bus (Front Side Bus/FSB), the memory controller and the memory bus.

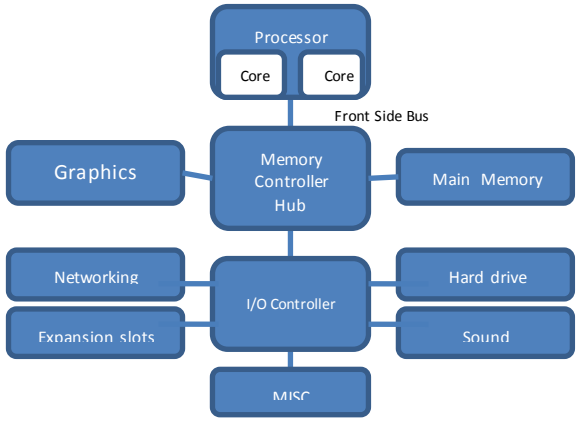

Figure 1: Architectural overview of a modern computer

In this section we go through the four most important systems in a computer; processor, memory, I/O-capacity and inter-process capacity. The three first are proposed by [6] with the addition of *inter-node communication* suggested by [7], although we find inter-process capacity to be a better description. Over the last 30 years the execution speed of a microprocessor has increased from 5 MHz to approximately 4 GHz. During this time many new techniques that improves the processors were introduced, such as pipelining and speculative execution as well as advanced instruction level parallelism. The old recipe of creating a faster processor by raising the clock frequency relied on the possibility to

manufacture smaller components inside the processors. Advances in this field are no longer possible due to the laws of physics.

The solution to higher speeds taken by almost all manufacturers is multiple processor cores on the same chip.

2.1.Memory Challenge

So far the memory architecture shown in figure 1 has not changed notably between the single and multicore processors. One of the major challenges of the multicore systems will be to design a system that can keep up with the tremendous computing power in a single multicore chip. This puts enormous pressure on, among other things, the FSB and the memory subsystem. At the moment the memory subsystem, on its own, is not fast enough to keep up with a single core processor; as a result it utilizes a two to four level memory hierarchy of increasingly faster memories closer to the processor. When extra cores are added some memory problems might be fixed by adding additional memory levels [3].

By comparing the historical capacity of processors using the SPEC-benchmark [10] with the FSB and memory bus [10] [11] we find that the difference has increased quite drastically over the years. Consider an application that was written for a system in 1993 and perfectly optimized for that "balance", would on a modern computer wait for memory approximately 50% of the time. It is a crude comparison and it is based on using one core in the modern multi core chip.

The problem of sustaining a high memory throughput to several processors is not a new problem for multiprocessor systems [12]. However, there is a difference between using multiple processors and multiple cores.

2.2.Inter Process Communication Challenge

All but the simplest parallel and distributed applications implement some level of inter-process communication. One problem that arises with the introduction of multicore clusters is that they have a non-uniform inter-process communication architecture. There are at least two different levels of communication costs involved when passing messages in a multicore cluster, either over the loopback interface or over the network.

Consider an application with four processes that implements message passing. If designed in 2003 it would be written to work well on Pentium 4 processors and gigabit Ethernet.

When executing a 16 process job on 4 computers each equipped with 2 dual core processors. The inter-process communication will be even more non-uniform with the additional two different levels of intra-node communications. Especially when compared to the 16 single core computers that the application was designed to be executed over. In that case the link capacity would still be $1/16^{th}$ of gigabit Ethernet, since it scales linearly.

2.3. I/O Utilization

The fourth important subsystem in regards to system balance is the I/O-subsystem. However the CFD applications that we focus in this investigation are very light on the I/O side. They only read the input files in the beginning of the execution and write the output at the end. Therefore we do not have any focus on the I/O utilization.

3 Computing Platform and Benchmark Applications

The aim of this experiment is to investigate the effect multicore processors have on applications in the Computational Fluid Dynamics (CFD) domain. Or more precisely, to determine if the efficiency of a simulation increases when two simulation processes are executed on each dual core node compared to one process per node.

3.1. High Performance Computing Platform

The Intel Core 2 is a common processor in many high performance environments. The cluster used in this investigation consists of 180 identical computers based on the Core 2 E6550 processor. The E6550 is manufactured using 65nm technology and has 291 million transistors.

The Intel Core 2 processor has two processor cores, each core has 2*32KB of private L1 cache (data and instructions), at the next level the cores share a 4MB L2 cache. Having a shared L2 cache has its benefits as well as drawbacks. While it allow cache sharing between cores and allowing the entire cache to be used by one of the cores as well as decreasing the cache impact of moving processes between cores it make the cores compete for the cache resources the hit latency is longer than for separate caches [12].

The memory controller and front side bus (FSB) of the Core 2 system are located off-chip, i.e. the architecture mimics that in figure 1. The transfer rate of the FSB has a maximum capacity of 10.6 gigabyte/s while the memory sustains 5.33 gigabyte/s per channel. The upper limit for fetching data from memory is limited to 10.6 gigabyte/s. However, for this to be achieved there has to be no other transfers affecting the busses. If the data is saved on the circuits in the same memory bank, the maximum theoretical transfer speed is 5.33 gigabyte/s.

The STREAM benchmark [13] [14] was used to measure the sustainable bandwidth of the memory system for reference purposes. During several executions with different sized matrixes of a size between 48 and 1800 megabyte, the sustained bandwidth never exceeded 4080 megabyte/s. When the matrix size was increased to span both memory channels the total sustained bandwidth was even decreased to 3900 megabyte/s.

The cluster is connected by a hierarchy of gigabit switches. During the experiments the computers were connected by a single gigabit Ethernet switch. A more detailed description of the hardware can be found in table 1.

Table 1: Cluster computer and network equipment specifications.

Processor	Intel Core 2, E6550
Clock Frequency	2.33GHz
# of cores	2
Hyper threading	No
Technology	65nm
Transistors	291 Millions
L1 Cache	32 KB code and 32KB data
L2 Cache	4MB cache shared between cores, non-

	inclusive with L1 cache
Front side bus	Q35
Frequency	1333MT/s
Bandwidth	10.6 gigabyte/s
Memory	DDR2
Size	2GB
Specification	PC2-5300
# of channels	2, dual channel
Frequency	667Mhz
Bandwidth	2 * 5.33 gigabyte/s
Measured Bandwidth	4.08 gigabyte/s (STREAM [13])

3.2.Applications and Models Used

Three applications were used in this investigation. Two commercial CFD applications; CFX [15] and Fluent [16] and the open source package Openfoam [17]. Nine CFD models, three per application, were used in the experiments. Since the behavior of these applications are dependent of the type of flow simulated so are the solution times. Three representative "real world" models were selected from two multinational high-tech companies for each of the two commercial applications. The CFX models ith Openfoam one real world and two research models from scientists were used together with Openfoam. Table 2 gives an overview of the models which have bearing on the aerospace domain.

Table 2: Sizes and names of the nine models used in the evaluation. The Number of elements gives a first order approximation of the size of the model.

Name	CFX	Fluent	Openfoam
M1	817 460	817 460	702 000
M2	1 811 036	743 223	1 980 900
M3	2 935 295	5 529 223	5 379 520

The sizes of the models as well as the run-time differ between models. As an example the Fluent M1 require approximately 4 hours to execute all the way through over eight nodes while Fluent model 2 require somewhere around 12 hours to complete and the M3 models need several days.

Those knowledgeable in the area of computational fluid dynamics know that there are two basic ways to determine when a simulation should end, either by specifying the number of calculations (iterations) to run or by specifying some convergence criteria (transient) [18].

When dealing with real world industrial applications and models one problem is that they are not well defined small units that execute in a minute. Real world CFD calculations are huge problems that often require tens of gigabytes of memory and take weeks to execute on a single computer. If all models in this investigation were to be executed all the way though it would require several years of runtime. In [19], Yang, et. al. however showed that a partly executed simulation fairly well matches a full execution of the code. Their empirical performance prediction results are constrained to cases where the problem and input sizes do not vary.

To be able to perform this investigation we decided to limit the execution time of the experiments. It was decided that we would use the iteration method (mentioned above) to limit the execution time. Each model was executed, in parallel, over eight computer nodes, with one process per node. After 1000 seconds of execution, the executing iteration was extracted from the application log file. This iteration was then used as the stop criteria for all experiments with that particular model, ensuring the exact same number of iterations and execution behavior in all comparable runs for that model-application combination. In this way only the changes in the number of processes and hardware could be responsible for the changes in run-time.

4 Experiment Setup

To determine the usefulness of multicore processors within the engineering simulation domain, the nine (9) fluid simulations in table 2 and their corresponding applications were executed in two different setups, Single and Dual.

Although utilizing the same hardware the difference between the two setups was that in the single setup each process is executed on one dual core node, as if it were a single core system, although the second core will execute various other system tasks. The dual setup on the other hand refers to when two processes are placed on each dual core computer node.

This means that when executing a eight processes job the Single setup will use eight dual core nodes and place one process on each node, effectively only utilizing one of the cores for simulation purposes. The dual setup, however, would employ four computers and utilize both cores in all machines: using a total of eight cores.

The models (table 2) were pre-partitioned into [2 4 6 8 16 32] partitions to be run in parallel by the same number of processes. This partitioning was done using the applications' default mechanisms.

For the simulations where the amount of memory needed for the execution was greater than the available system memory (figure 3 and table 6), thus forcing the systems to page to disk, was removed from the study. The reason behind this was that their execution times increase by several orders of magnitude. This threshold was reached earlier with the Dual setup since the node memory is shared between two processes. When adding more computer nodes and processes the amount of global memory effectively increases which in turn allow the applications to run entirely in memory.

During the execution of the applications the run time was measured and the processor, memory and network loads were monitored. To calculate the runtime a timestamp were made before and after the execution of the pre-partitioned simulations. The timestamps were taken with the gettimeofday() system call which has a relatively high resolution (ms). Although compared with a runtime between 253 and 6388 seconds the error is negligible.

The processor, memory and network loads were calculated as averages over 180 second intervals. These measurements were all made through the /proc kernel interface. One measurement set consumed ~8 ms of processor time, according to measurements done by strace [20].

5 Results

In order to investigate the effect multicore processors have on our three applications and the nine models that were executed in a

Single and Dual core setups, see section 4. Figure 3 a-i) contain graphs depicting the results from the experiments. The black lines show the execution time of the Single setup, i.e. where one process is placed on each node, and the dashed line represents the Dual setup where two processes are placed on each dual core node.

At a first glance the results look as one would expect them too, the Single setup for the same number of processes out performs the Dual setup, see figure 4. This was expected since the Single setup use two times the number of computers and effectively has twice the amount of memory, twice the level 2 cache (not applicable to all multicore processors) and an extra core to use for running operating system and other tasks, etc.

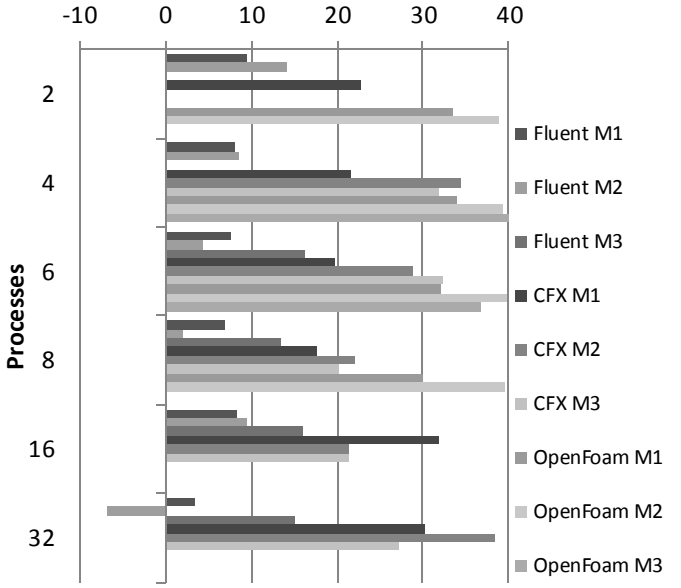

Figure 4: The number of percent by which the Single setup outperforms the Dual setup for the same number of processes.

This was the case for all models, apart from Fluent model 2, when divided into 32 processes as can be seen in figure 3 b). The dual setup outperformed the single with 22 seconds out of 341, so by ~6.5%. Fluent model 1 is not much larger and shows the same behavior, although the single setup still outperforms the dual with 3.5%, at 32 processes.

In table 4 the speedups (scalability) of the different applications and models are listed. To compare both single and dual setups with each other the speedup of the dual setup uses the same baseline as the single. Thus all values for each model are normalized after the single setup with the least amount of processes.

The speedup obtained when adding processes to the computation does not follow the theoretically best possible of $\mathbf{T_s/P}$ (time to execute simulation/number of processes) but for software in the CFD area scalability in this range is considered comparatively well [18] [21].

In table 3 we can see that when going from two Single to two Dual processes, i.e. from two cores on different computers to two on the same, Openfoam does not perform as good as the other two applications. Openfoam M2 models also shows a strange plateau, at 4 to 6 nodes as well as 8 to 16, where there is little to none scalability. Although for eight nodes it shows a decent scalability. Fluent M3 shows a similar, but not at all as bad, pattern between eight and sixteen nodes.

Quoting the execution times and speedups based on the amount of processes the problem is divided into is a straightforward approach. However, when dealing with computer clusters and multicore processors the speedups and execution times depending on the amount of computers is of interest.

Table 3: The execution time (T) of the different setups normalized to the equivalent lowest number single setup execution, T_N. The norm was given the value 100.

Model	Setup	Processes					
		2	4	6	8	16	32
Fluent M1	Single	100	53	39	32	17	13
	Dual	110	58	42	34	19	13
Fluent M2	Single	100	57	48	34	22	18
	Dual	116	62	50	35	24	17
Fluent M3	Single	-	100	72	63	57	24
	Dual	-	-	86	73	68	28
CFX M1	Single	100	54	39	34	22	17
	Dual	130	69	49	41	32	24
CFX M2	Single	100	56	40	34	21	15
	Dual	-	85	57	44	27	24
CFX M3	Single	100	55	41	35	23	18
	Dual	-	80	60	44	29	24
OFoam M1	Single	100	47	31	24	-	-
	Dual	150	72	46	34	18	-
OFoam M2	Single	100	52	53	26	-	-
	Dual	163	85	88	44	43	-
OFoam M3	Single	-	100	93	59	-	-
	Dual	-	269	147	59	41	-

5.1.Dual core Processor Efficiency

Turning to the less obvious result, for some models/setups the problem is actually solved faster when only one process instead of two is executed on each dual core computer node. This occurred at the higher number of computers and the speedup was between 6.5% and 64%, see table 4 for details. For the lower number of computer nodes the Dual setup always outperform the Single – given that there is enough resources to execute at all. E.g. the Fluent M2 model required more than 4 gigabyte of ram and would there for not execute on 2 machines or less. Then as more nodes are added the time gap between the Single and Dual setups close and for 5 of the 9 models the curves actually switch place, leaving the Single setup to outperform the Dual for the higher number of nodes.

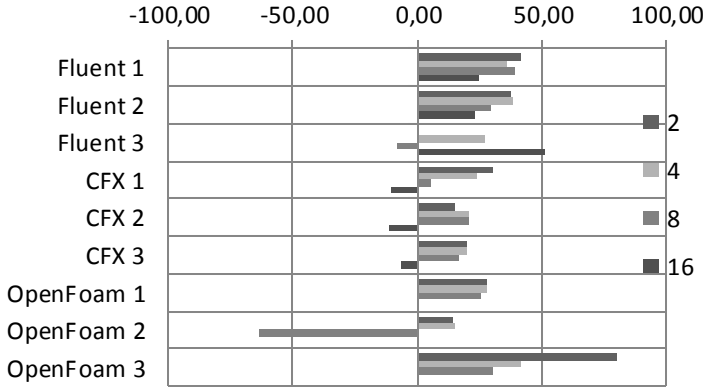

Figure 5: The number of percent by which the dual setup outperformed the single setup for the same number of computers [2,4,8,16].

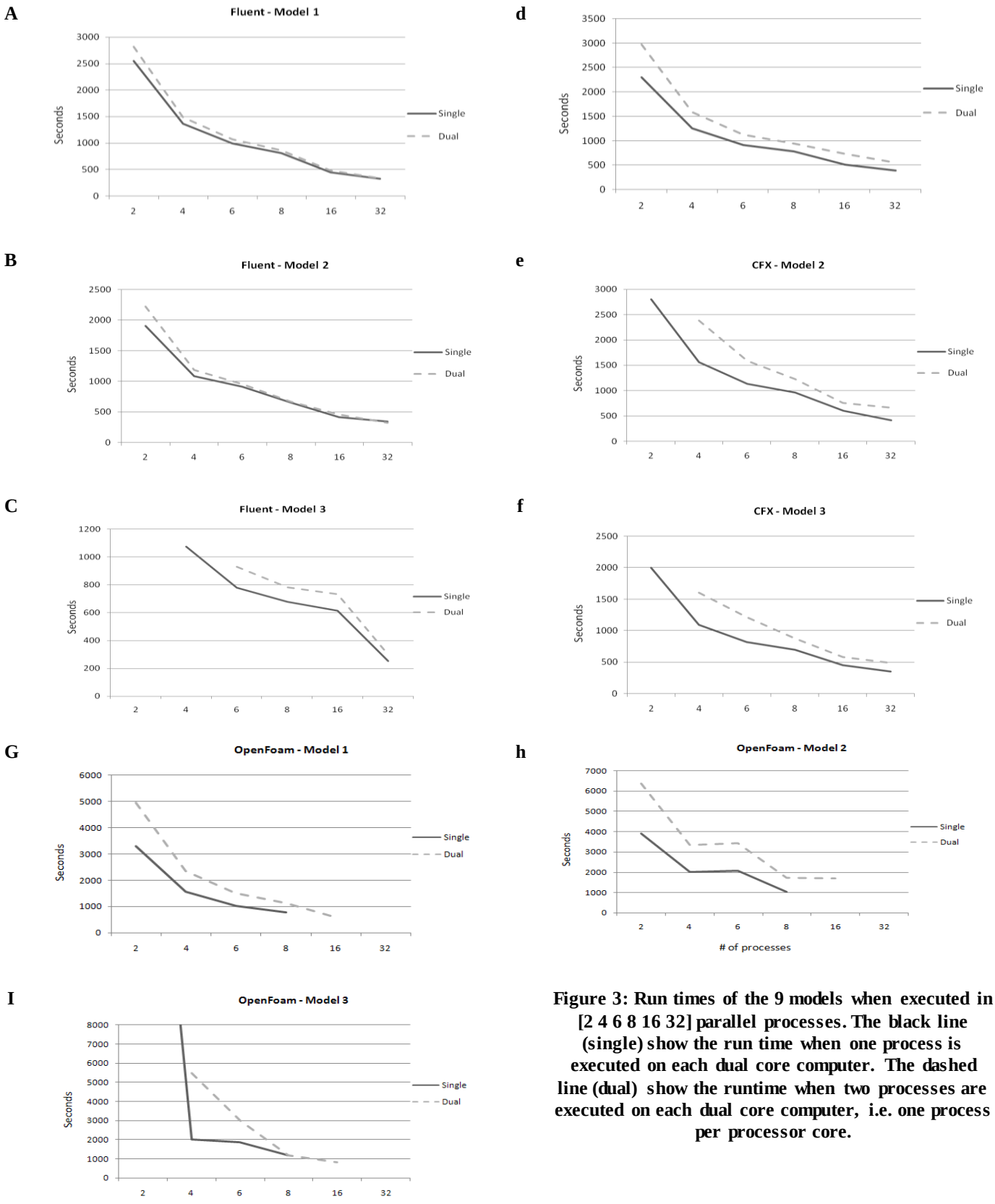

Figure 3: Run times of the 9 models when executed in [2 4 6 8 16 32] parallel processes. The black line (single) show the run time when one process is executed on each dual core computer. The dashed line (dual) show the runtime when two processes are executed on each dual core computer, i.e. one process per processor core.

The Single node executions that outperformed their Dual counterparts are listed in table 4 together with the speedup gained. Using Fluent model 3 as an example, the application actually solves the problem faster if you divide it into 8 parts and run it on 8 dual core processors compared to dividing it into 16 parts and running one process on each core on 8 dual core nodes. Hence, if there are 8 dual core nodes it will take 8% longer time to execute the problem if you utilize both cores instead of just one. The same applies to all CFX models when executed on 16 computers as well as for Openfoam model M2 when executed on 8 computer nodes. For the Openfoam model M2 the speedup gained, when executing one process per computer instead of two was as high as 63%.

Table 4: The following models execute faster when executed with one process on each dual core computer than when executed with two processes on each computer.

Model	# of computers	Speedup for single
Fluent M3	8	8%
CFX M1	16	10%
CFX M2	16	11%
CFX M3	16	6,5%
OF M2	8	64%

Based on the results presented one can draw the conclusion that the generally most time efficient solution is to execute the CFD simulations according to the Single scheme. E.g. executing only one process on each computer; leaving the second core virtually unused. For the lower amount of processes the Single setup require two times the computers that the Dual setup. This effectively doubles the hardware investments, but using two times the processes also doubles the software license costs. This issue was investigated in [22] [23] where the conclusion was that the cost of adding hardware is preferred to the cost of the added software licenses.

For Fluent and CFX the advantage seems to decrease as the number of processes increase but then raise again at 16+ processes, this statement is true for all but Fluent M2. As Openfoam goes the difference is even larger, here the Single setup outperformed the Dual for all number of processes with between 29% and 40% (on average 32%), although this does not include the Openfoam M3 Single 2 and Dual 4 simulations since they did not fit completely into memory.

6 Analysis and Discussion

The most notable results from the investigation are those presented in figure 5 and table 4 – In these cases the application and model combinations execute faster with one core inactive on each compute node, i.e. when one process (P) is executed on each one of the dual core computer nodes (N) compared to when two processes are executed on the same number of nodes.

The main issue is to determine if this behavior is dominated by hardware bottlenecks or the software scalability. If an underlying mathematical or programming intricacy causes behavior, the application should behave *the same as long as the number of processes are constant*. If all resource demands are satisfied the Dual(P) and Single(P) runs would execute in the same amount of time. Since this is not the case to try to identify the resource(s) that Dual(P) uses more of than Single (P). We now continue with a more detailed analysis looking the three candidate resources suggested in section 2 above.

6.1. Network Usage

The investigated applications at hand communicate on a process to process basis. All processes can communicate with all other processes as well as carry out collective MPI operations, such as broadcasts and scatter-collect operations. A process in a Single setup that sends a message to another process will always send it thru the switch (1). Where Switch represents the amount of traffic traversing the switch, N is the number of nodes, P is the number of processes and M is the average number of Megabytes transmitted by each process to each other process.

$$Switch_{Single} = (N^2-P) * M \quad \text{(eq. 1)}$$

A process in a Dual setup, however, will carry out a portion of its communication over the loopback interface but the larger part still goes through the switch.

$$Switch_{Dual} = (N^2-2P) * M \quad \text{(eq. 2)}$$

When comparing Eq. 1 and 2 it is obvious that the switch is subject to a higher utilization in the Single setup then in the Dual. However, in the Dual setup both processes on a computer node share the same network connection as well as the internal busses. The average load on each computer node is calculated as (switch/N)*(P/N). Nevertheless, consumed bandwidth is quite low. The Fluent M2 in 16 process Single configuration had the highest measured computer node communication 7.77 megabyte/s. It is one out of ten configurations that had a node communication of more than 5 megabyte/s i.e. 40 megabit/s. All CFX and Openfoam models had a per node communication less than 2.5 Megabyte/s.

Based on these numbers it is highly unlikely that the single process per machine performs faster than two processes per machine depends on the network resources. The increase in network utilization might impact the execution time in a negative way but not at all on the scale observed.

6.2. Memory Footprint

The investigated applications use a technique called bulk synchronous parallel [24], as do the majority of the industrial simulation codes in the CFD and FEM areas. This means that these applications at the global level hold a large matrix in memory of #E number of elements, where each element has a size of E_{size} in memory. The total memory size of the matrix is then $\#E * E_{size}$. Considering the inherent memory footprint M of the application, one could naively expect the memory usage per process to scale according to eq. 3 below, where P is the total number of processes.

$$M + (\#E * E_{size}/P) \quad \text{(eq. 3)}$$

However, all three applications use some optimization schemes to obtain higher performance. To avoid using the network too much some boundary data is stored with several processes. Instead of communicating, which in best case results in the introduction of network latencies and in the worst case bandwidth depletion, some elements are calculated and stored locally by several processes. The memory demands can thus more correctly be described by eq. 4, where Δ represents the amount of redundant information stored for optimization reasons.

$$M + (\#E * E_{size}/P+\Delta)$$

(eq. 4)

As can be seen in figure 9 the amount of memory does not decrease linearly with the number of processes. For many of the cases the per computer memory usage actually increases with the number of nodes. Evidently RAM-memory shortage cannot explain why the applications execute faster when only one process is placed on each dual core computer instead of two.

6.3.Memory Bandwidth

Despite that each process is identical in both execution and memory demands regardless of it being executed by itself on a compute node or together with an additional process, there is an important difference in technical context. Even if the memory allocated for two processes still easily can be fitted into the main memory (see above), the number of memory accesses will be (more than) doubled.

At the same time there is a major difference between the single and dual setup due to the shared L2 cache of the Intel Core2duo architecture. In the Single setup each process has a 4 megabyte L2 cache while in the Dual setup it has to be shared between the two processes, and the processes works on two different datasets. Thus, while the number of memory accesses *increases* by a factor of two or more (inter process communication among other things will create some extra load), the available L2 cache will in practice *decrease* by a factor of two. It is consequently reasonable to suspect that this change will give raise to additional delays not present in the single execution setup. When the STREAM benchmark was used to measure the sustained bandwidth it never exceeded 4080 megabyte/s, see section 3.1 for more details.

7 Conclusion

In this paper we investigated the impact multicore processor based clusters has on the execution of industrial simulations, namely computational fluent dynamics (CFD). Almost all new cluster installations are equipped with multicore processors, but according to our results it is far from trivial to determine if this is in fact a blessing or a curse.

The most significant finding was that five of the models, ran faster when only one process (P) were executed on each multicore node (N). When two processes were executed on each node the execution time was increased by between 6.5 and 64% with a median increase of 10% in these cases. In general the trend is that at a low number of nodes two processes per computer, i.e. one process per core, outperforms once process per node, but as the number of nodes and processes increase the gap closes and eventually the single placement strategy wins.

We evaluated four different possibilities, application scalability as well as processor, memory and inter process communication capacity as suggested by [6] as well as [7]. Application scalability was ruled out early since there were no scalability issues with the number of processes that were executed with one per computer node instead of two. The inter process communication capacity were also ruled out as a major contributor. Turning to the memory and memory bandwidth utilization it is obvious that the shared FSB and L2 cache of the Intel Core2duo architecture affects the performance. In the Single setup each process has a 4 megabyte L2 cache which in the Dual setup is shared and they work on two different datasets.

Ultimately we draw the conclusion that it is not obvious that utilizing more than one core of a dual core processor is beneficial. Furthermore we know that there are several cases where it is a great deal better to only use one core, especially when factoring in the per process license costs. The reason behind this (technical) behavior is still unknown although it is likely that it is linked to the memory hierarchy, and it is highly unlikely that it depends on the inter-process communication or the processor utilization.

8 References

[1] N. Singer, ”More chip cores can mean slower supercomputing,” *Sandia National Laboratories*, January 2009.

[2] J. Held, J. Bautista och S. Koehl, ”From a Few Cores to Many: A Tera-scale Computing Research Overview,” Intel white paper, 2006.

[3] L. A. Polka, H. Kalyanam, G. Hu och S. Krishnamoorthy, ”Package Technology to Address the Memory Bandwidth Challenge for Tera-scale Computing,” *Intel Technology Journal*, vol. 3, nr 11, 2007.

[4] L. Liu, Z. Li och A. H. Sameh, ”Analyzing memory access intensity in parallel programs on multicore,” i *International Conference on Supercomputing*, Island of Kos, Greece., 2008.

[5] S. K. Moore, ”Multicore is bad news for supercomputers,” *IEEE Spectrum*, November 2008.

[6] R. Buyya, High Performance Cluster Computing, New Jersey, USA: Prentice Hall, 1999.

[7] F. Vaughan, D. Grove och P. Coddington, ”Communication Performance Issues for Two Cluster Computers,” i *Proceedings of the Twenty-Sixth Australasian Computer Science Conference*, Adelaide, Australia, 2003.

[8] T. Sterling, D. Becker, D. Savarese och J. Dorband, ”Beowulf: A Parallel Workstation For Scientific Computation,” i *Proceedings of the 24th International Conference on Parallel Processing*, Oconomowoc, US, 1995.

[9] A. Boklund, C. Jiresjö och S. Mankefors, ”The Story Behind Midnight, a Part Time High Performance Cluster,” i *Proceedings of the 2003 international conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, 2003.

[10] J. L. Hennessy och D. A. Patterson, Computer Architecture: A Quantitative Approach, 4 red., US: Morgan Kaufman Publishers, 2006.

[11] Intel, ”Intel Museum,” [Online]. Available: www.intel.com/museum/.

[12] L. Peng, J.-K. Peir, T. K. Prakash, Y.-K. Chen och D. Koppelman, ”Memory Performance and Scalability of Intel's and AMD's Dual-Core Processors: A Case Study,” i *Performance, Computing, and Communications Conference*, New Orleans, USA, 2007.

[13] J. D. McCalpin, ”Sustainable Memory Bandwidth in Current High Performance Computers,” 1995. [Online]. Available: http://www.cs.virginia.edu/~mccalpin/papers/bandwidth/bandwidth.html.

[14] C. Hristea, D. Lenoski och J. Keen, ”Measuring memory hierarchy performance of cache-coherent multiprocessors using micro benchmarks,” i *ACM/IEEE conference on Supercomputing*, San Jose, USA, 1997.

[15] Ansys, ”CFX application,” [Online]. Available: http://www.medeso.se/software/ansyscfx.html.

[16] Ansys, ”Fluent application,” [Online]. Available: http://www.fluent.com/software/fluent.

[17] OpenCFD, ”Openfoam Application,” [Online]. Available: http://www.opencfd.co.uk/openfoam/.

[18] S. Perzon och L. Davidson, ”On CFD and transient flow in vehicle aerodynamics,” SAE Technical paper, 2000.

[19] L. T. Yang, X. Ma och F. Mueller, ”Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution,” i *Proceedings of the ACM/IEEE Super Computing Conference*, 2005.

[20] M. Frye, ”Debugging code with strace,” *RedHat Magazine,* August 2005.

[21] A. Boklund, C. Jiresjö, S. Mankefors-Christiernin, N. Namaki, L. Gustavsson-Christiernin och M. Ebbmar, ”Performance of Network Subsystems for Technical Simulation on Linux Clusters”, *Conference on Parallel and Distributed Computing and Systems*, Phoenix, USA, 2005.

[22] Boklund, N. Namaki, S. Mankefors-Christiernin, J. Gustafsson och M. Lingbrand, ”Dual Core Efficiency for Engineering Simulation Applications”, *Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, 2008.

[23] A. de Blanche, S. Mankefors-Christiernin, ”Minimizing Total Cost and Maximizing Throughput - A Metric for Node versus Core Usage in Multi-Core Clusters,” i *International conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, 2010.

[24] R. H. Bisseling, Parallel Scientific Computation: A Structured Approach using BSP and MPI, Oxford: Oxford University Press, 2004.