



White Paper

Performance Optimization of EXASolution with the Intel® Trace Analyzer and Collector Tool



Introduction

The ever increasing complexity for worldwide trade and its related enterprise computing infrastructures and business environments demand sophisticated and advanced solutions for a faster and better decision making. This white paper describes the performance enhancement of such a Business Intelligence (BI) solution using the Intel® Trace Analyzer and Collector software development tool for distributed computing.

EXASolution enlarges the possible field of utilization for Business Intelligence applications for large data volumes. By using innovative technology, EXASolution can perform BI analyses up to 100 times faster than traditional databases. For the first time, users are able to perform large and complex analyses in real time. The integration of EXASolution in all established Business Intelligence tools is simple, thanks to standard interfaces.

With the resulting performance optimization, EXASOL is able to extend its leadership in the worldwide recognized TPC™ H Benchmark (TPC-H).

EXASolution - Parallel DataWarehouse Relational Database Management System (RDBMS)

Today, the business processes of modern enterprises are supported by multiple information-processing systems. Analysis requirements are continuing to rise and the speed and accuracy of Business Intelligence evaluations are increasingly crucial to success.

For this, the EXASOL concept is as follows (www.exasol.com):

Intelligent software instead of expensive hardware!

EXASolution is a highly scalable database management system that applies efficient algorithmic solutions from the field of high-performance cluster computing to the analytical challenges in the BI environment. EXASolution shatters existing data warehousing performance barriers via the unique combination of in-memory processing technology and a shared-nothing architecture using innovative compression concepts and advanced cluster technologies.

Intelligent algorithms distribute the data within a cluster and automatically process the necessary optimization steps on the fly. With its self-developed communication library and highly adaptive data distribution algorithms, EXASolution achieves such high data throughput rates

that expensive storage area networks (SAN) become redundant. If the data volume, analysis requirements or number of users increases, the cluster can simply be extended by adding additional server system nodes.

Intel® Trace Analyzer and Collector

Efficient optimization always has to be started with an analysis of the optimization potential. Without such an analysis it is very likely that optimization is done in a sub-optimal way and time and efforts are potentially wasted. After an optimization is done, the difference to the starting point (base line) due to the optimization has to be evaluated. This is certainly true for single processor and core performance which can be monitored for example by the Intel® VTune™ performance analysis tool. It is also true for multi-threaded applications to be analyzed by the Intel® Threading tools like the Intel® Thread Profiler. Hence, it is especially true for distributed applications where a simple profiling can only show hotspots but give no clues how these hotspots are generated and inter-related.

The Intel® Trace Analyzer and Collector is the appropriate tool for this (distributed messaging) task. It consists of the following two parts: (1) the Trace Collector library and (2) the Trace Analyzer graphical user interface (GUI). Intel® Trace Analyzer and Collector is capable of monitoring, collecting, and analyzing the temporal behavior of distributed messaging applications employing hundreds or thousands compute nodes and cores. It has a history of more than 15 years of development and successful optimization for MPI (Message Passing Interface) based applications. However, as we will see below, it is also possible to use Intel® Trace Analyzer and Collector for distributed non-MPI applications like the RDBMS EXASolution.

One of the key factors for performance in distributed applications is *scalability*. Ideal scalability means that doubling the *resources* for a program will result in doubling the *outcome*. In our case *resource* is the number of processors/cores utilized and *outcome* is some performance measure like operations or transactions per second delivered.

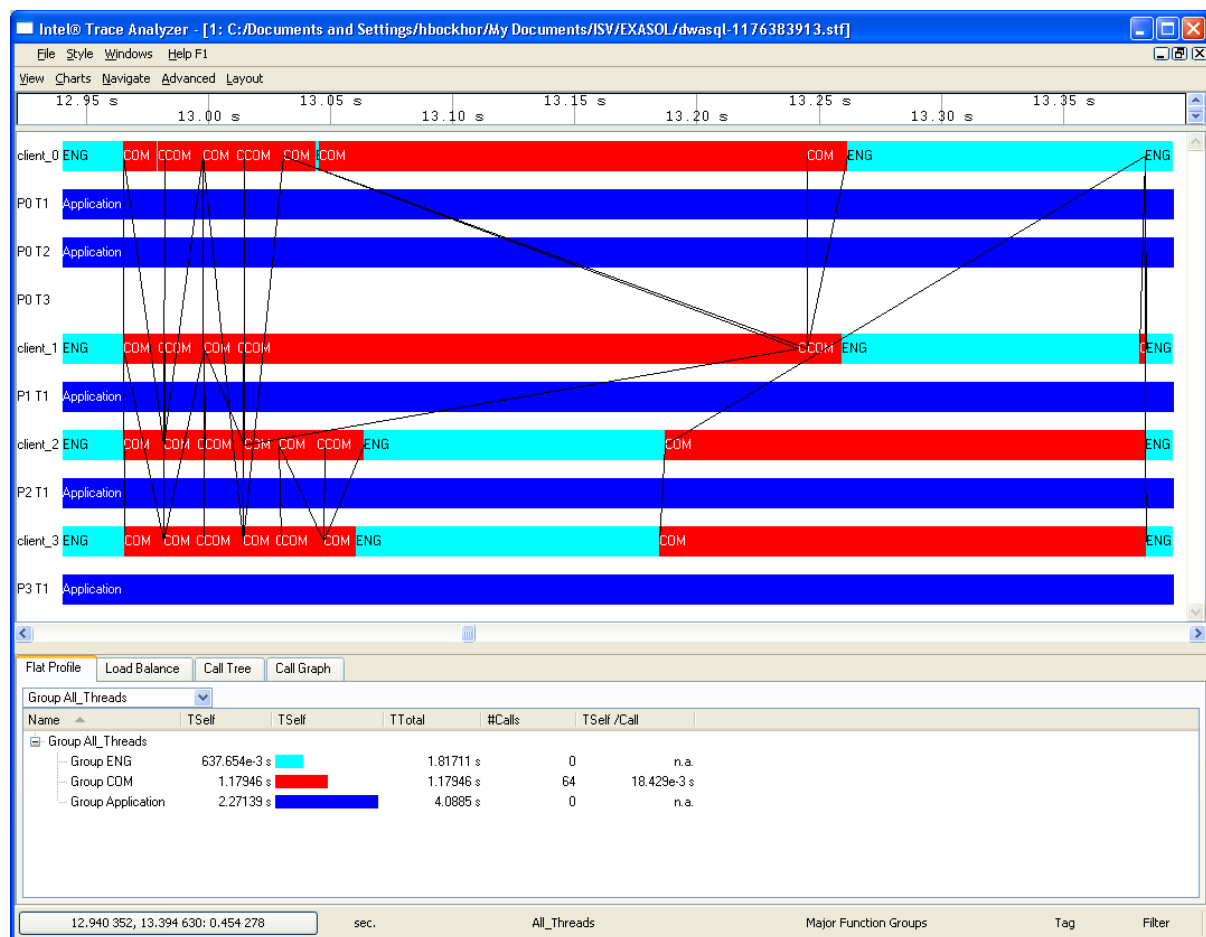


Figure 1: Intel® Trace Analyze and Collector Visualization Window

Any serial parts of the application inhibit scalability described and formulated by the famous Amdahl's law. For example, a 10% serial execution time limits the maximum theoretical possible speedup of the application to a factor of 10 - even with an infinite number of processors being infinite fast. It is also worthwhile to mention that any single processor optimization in the 90% parallel part of the program will have no effect on the maximal speedup of 10x.

While parallelizing the serial parts of the program is an obvious task, other factors inhibiting scalability are harder to find. Two well known factors are *load imbalance* and *dependencies*. In the first case the work is not evenly distributed on the processors, and in the second case process A has to wait on process B until it can deliver an intermediate result. Especially dependencies can not be detected by simple profiling, but Intel® Trace Analyzer and Collector will clearly point to these effects. Besides these factors depending on the parallelization algorithm, the network hardware and its characteristics influences the choice of the best communication strategy. Other obvious characteristics are latency and throughput, as well as e.g. interconnect/Ethernet network congestions when too many messages travel over the same network interface card (NIC) or through the network-switch at the same time.

The first step of an Intel® Trace Analyzer and Collector analysis is to generate a trace with instrumented events like entries and exits into and out of functions and message sends and receives. The Trace Analyzer GUI is quite intuitive and the basic elements of two of the key charts (event timeline and function profile) in Intel® Trace Analyzer and Collector can be explained best by using the following snapshot taken from one of the first tracings at EXASOL (Figure 1).

In Figure 1, on top of the event timeline chart the wall clock execution time is displayed (timeline). Below the timeline, several colored bars are visible. These bars represent the used threads of the traced application. The threads are traced automatically because the Trace Collector-API routines check if they run on a newly generated thread. The notation *PxTy* on the left side of a bar means process #x and thread #y. The names *client_0*, ..., *client_3* are defined by the user (see below). These names are displayed for the master thread *PxT0*. Two different groups of function symbols are defined in this trace: *ENG* and *COM*, and where given (defined) by the user. *COM* is used for the communication routines and *ENG* for database engine routines. The identifier *Application* is used as default group for all the rest of the application. The black lines connecting the thread bars symbolize (socket-) communication between processes. More information on functions and messages can be revealed within Intel® Trace Analyzer and Collector by right-clicking on the corresponding symbol. At the bottom of the window view the function profile chart is displayed.

In traces for applications using MPI, the communication routines will be colored red automatically. For non-MPI tracing, the definition of communication routines has to be done explicitly by the user. In our case, the EXASOL example displays communication *COM* in red as well. Hence, looking at accumulations of "red" routines shows where potential communication problems might be located.

Other important elements are the message lines. The Intel® Trace Analyzer and Collector tool allows investigation of each message by right-clicking on the black message line. Attributes shown are size and volume, start time, end time and rate. Comparing the speed of a message with the maximal possible speed provided by the interconnect gives an idea if a long transfer time is due to a slow interconnect or if it is due to a sub-optimal algorithm.

Intel® Trace Analyzer and Collector provides several other charts that characterize the problem from different view points, but the event timeline may be seen as a key chart for starting an analysis and navigating through the collected trace.

The example above shows a snapshot of two communication routines. On the left side an *all-to-all* communication is shown. Each process sends a message of roughly the same size to each

other process. While the algorithm is completely symmetric, the *COM* routines on *client_0* and *client_1* show substantially longer execution times. Looking at a longer part of the trace we see that this happens many times so it is not just a fluctuation in the interconnect fabric. By investigating the pattern, it is seen that two long running messages have to be received by *client_1* at the same time. This is a case of Ethernet congestion. The total communication time would be much less when we force the messages to be send one after the other. There is also a dependency visible because *client_0* has to wait until *client_1* has finished receiving and sends its final message to *client_0*. A better solution for this issue may be to insert barrier like operations for avoiding simultaneous sends to the same client. On other fabrics like Infiniband this effect may not be seen at all. The communication routine on the right side of the picture is a barrier operation for synchronization. The program execution can only resume when all processes have entered the barrier. The local load imbalance will result in waiting time inside the barrier for *client_2* and *client_3*.

After having shown how an analysis of a sub-optimal algorithm can look like, the question of how to get a trace must be answered. When using MPI the tracing happens automatically for the communication routines. In the distributed non-MPI case this must be done manually and the most difficult part of it is the initialization. Using the Intel® Trace Analyzer and Collector-API becomes straight forward when the program uses a communication library. If this library contains an initialization routine that is called by all processes, all Trace Collector initialization can be done within this routine. In the case of EXASOL this was the case; in other programs with less clean structured communication this task might need more attention.

The first step for initialization is to define which processes are Trace Collector clients to be traced and which process will host the Trace Collector server that collects all the data and writes out the Intel® Trace Analyzer and Collector trace file. For convenience one of the clients may take the server role as well. It is also possible to use an external program called VTserver contained inside the Intel® Trace Analyzer and Collector package. This program will then do the server part of Trace Collector after internal communication with the clients is established.

The second step is to initialize Trace Collector on the clients by calling *VT_clientinit*. This routine needs an integer process id and the name of the client (*client_0*, ..., *client_3* in our case) as input. It returns a so called contact string containing all the communication data (internet address, port and process id). This string has to be communicated by the user from all client processes to the server process of Trace Collector.

The third step is to receive all contact strings by the Trace Collector sever process and put the contact strings into an array. The server process takes this array as input for its initialization routine *VT_serverinit*.

The last step of initialization is that server and clients call the VT_initialize routine for establishing Trace Collector internal communication. The Intel® Trace Analyzer and Collector tracing starts after VT_initialize has been called by all processes. The methodology is slightly different when using the VTserver program instead the API call because VTserver needs the contact strings already as program arguments.

After initializing Trace Collector events like function calls or execution of code, individual user defined blocks can be traced by using the VT_begin and VT_end calls. It is also possible to include source code locations. Message lines can be generated by using VT_log_sendmsg and VT_log_recvmsg. The send call should be placed at the beginning of the send routine and the receive call is to be placed at the end of the corresponding receive routine. Intel® Trace Analyzer and Collector tracing will be finished when all processes have called VT_finalize. The clients will send their data to the server and the server will write out the trace file.

Experience shows that the exact methodology may be understood best by having an easy example. This example and further support will be made available by sending mail to a contact address given below.

Instrumentation of the EXASolution code

Tracing Communication

EXASOL's communication library was instrumented quite simply by adding tracing in the functions „send” and “receive”. Global operations like broadcasts were traced just like functions. The direct tracing of global operations caused some problems since EXASOL's communication library works buffered (e.g. an array is send as one block and then received value by value).

General Tracing

Functional blocks were instrumented very explicitly. Since the source code of EXASolution is rather large and communication is done in many places, it would otherwise be very hard to find out the source of the communication traces showed in the Trace Analyzer. As spin-off, the deep functional instrumentation was able to show the amount of code parallelization by the use of multithreading.

In Figure 2 one can see the summary of a query execution on two parallel servers (*client_0* and *client_1*). There are three sections with high communication (vertical lines). Threading (*T1*, *T2*, ...) looks pretty good since the worker threads are in idle-mode (red areas) only during communication steps and at the end. Thanks to extended instrumentation, the time

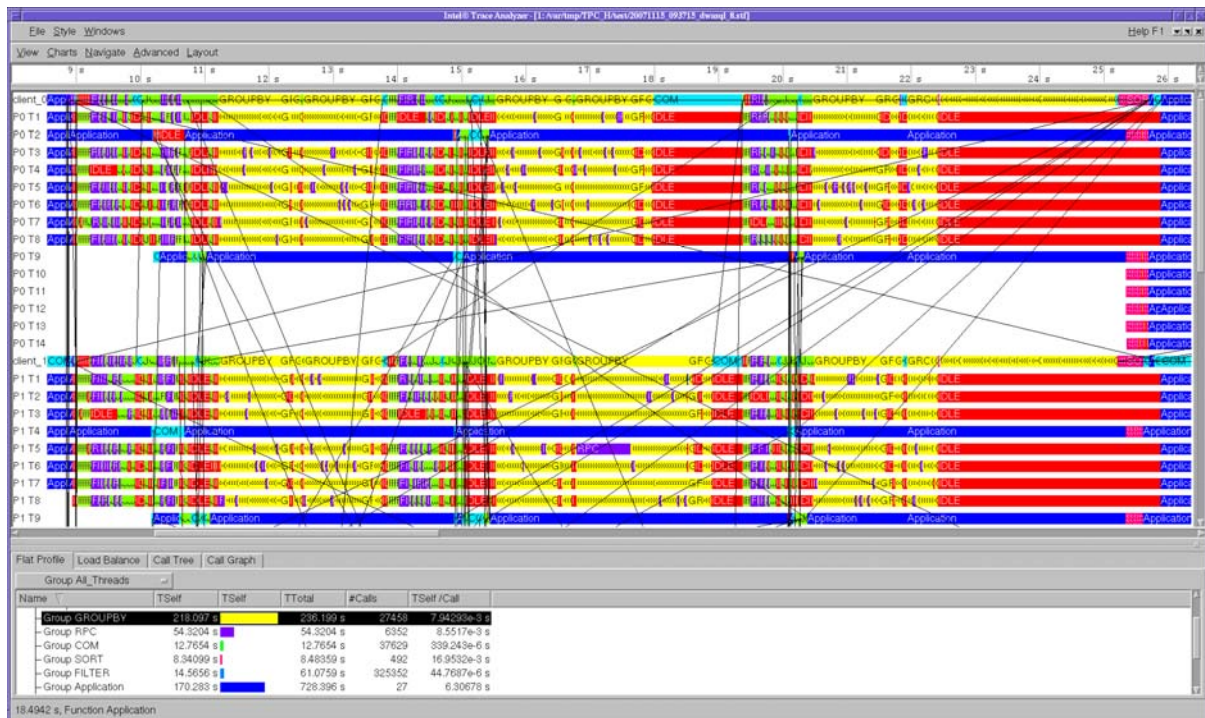


Figure 2: Trace-File for a complete SQL query

of each function is broken down very accurately. In this example one can see how fast one can get an overview about the program activity and optimization potential.

Performance Optimization of EXASolution

During the work with Trace Analyzer, many runtime characteristics were identified. Some examples are as follows:

- Identification of code which was not parallelized enough;
- At some points, only „half-duplex“ communication was used (see example);
- Communication through multiple processes (not threads) on one server caused problems since the operating system bandwidth is not shared fairly;
- An unexpected high amount of disk accesses revealed optimization potential inside the memory management library.

Figures 3 and 4 show a good example of optimization. One can see a global data exchange sequence between four servers (*client_0* - *client_3*). The purple areas mark functions of the communication library of EXASOL, whereas the green areas show computational steps (note that the coloring used in Intel® Trace Analyzer and Collector can be defined by the user).

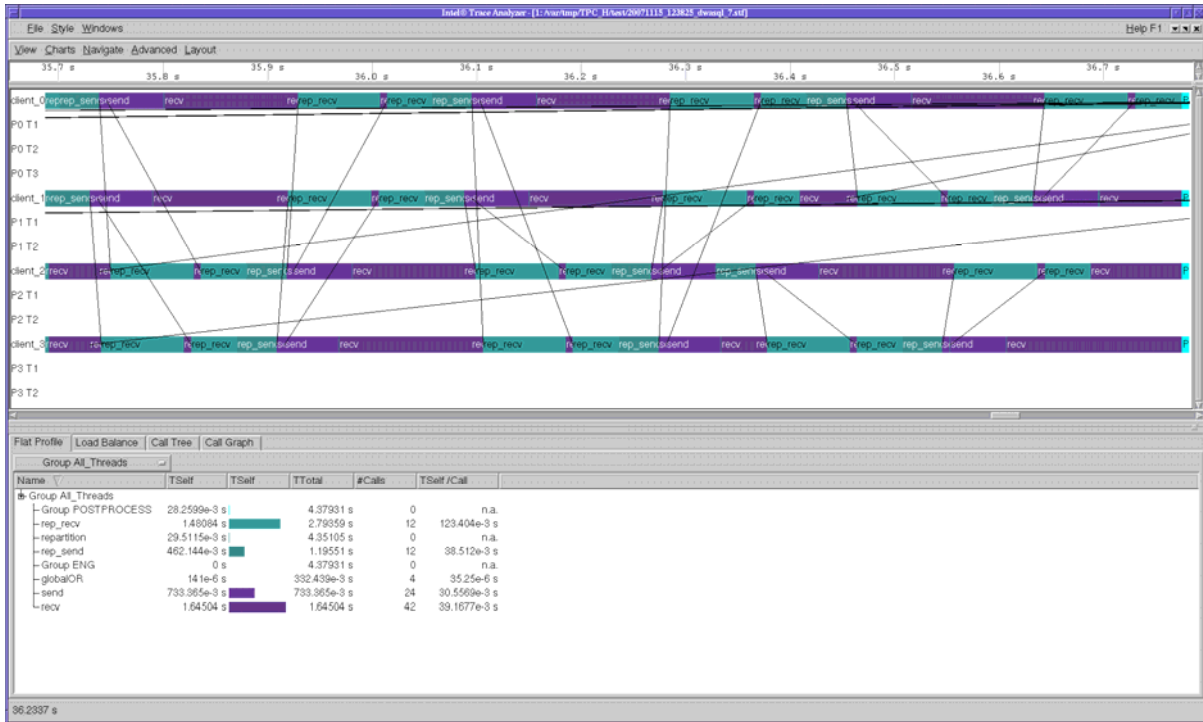


Figure 3: Communication behavior of four servers before optimization

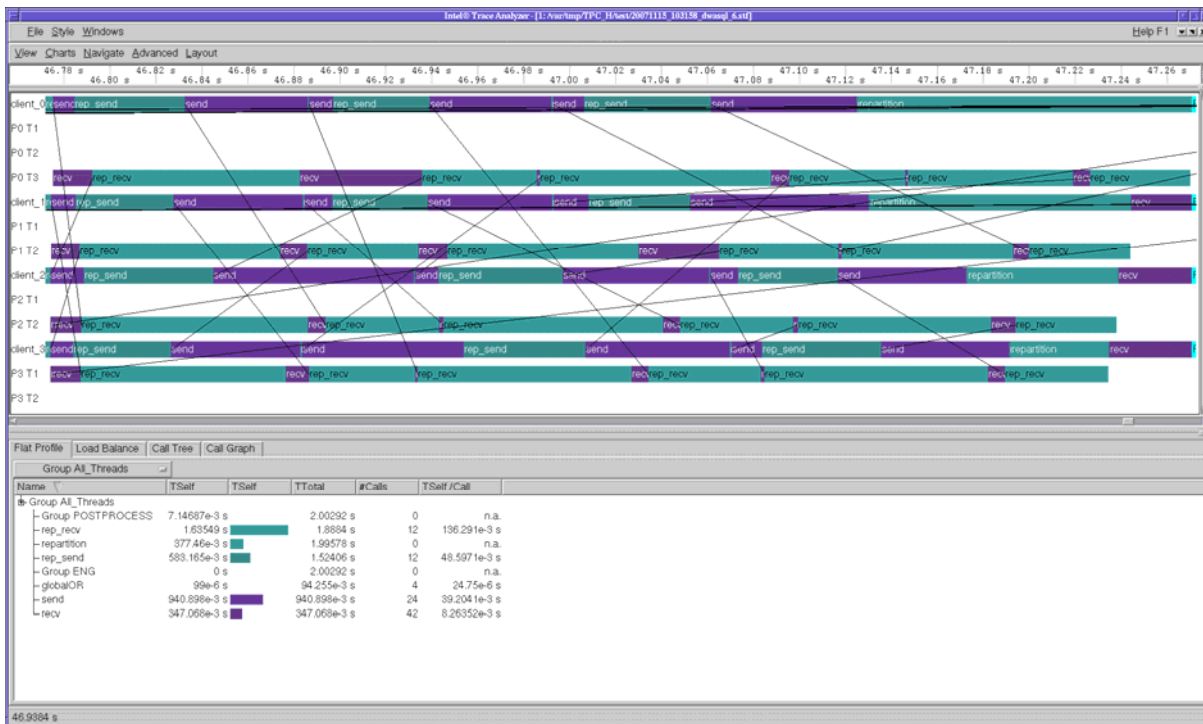


Figure 4: Communication behavior of four servers after optimization

Figure 3 looks relatively well arranged. The following activities can be identified:

- The sender computes some data and then sends two packets to the receiver;
- The receiver receives the first packet, does some computation, receives the second packet and computes something again;
- After that sender and receiver exchange roles and do the same the other way around;
- Afterwards new pairs of sender/receiver are built and the whole action is done again. Since there are four servers, three rounds of exchange are executed.

In Figure 4 one can see the same steps after doing some optimizations. Sending and receiving are now done in different threads which can send and receive messages simultaneously. Besides that, the computations of the sender were moved between sending the two data packets, since they were not needed for the first packet. Since the receiver does computations after receiving the first packet itself, this approach is rather good.

This simple optimization has accelerated one query from 2.65 to 1.97 seconds, and another one from 4.93 to 4.49 seconds.

Benefit for EXASOL: Extending the lead in TPC-H benchmark

The Intel® Trace Analyzer and Collector software tool was used by EXASOL in course of an optimization project to extend its leadership in the worldwide recognized TPC™ H Benchmark (TPC-H).

The TPC-H is a performance test of the independent Transaction Processing Performance Council (TPC) for data warehouse solutions. The benchmark simulates data warehouse analyses requiring considerable processing power and measures both the speed and capacity of database solutions. For further information, please see: www.tpc.org.

During this project, EXASOL was able to analyze the runtime behavior very efficiently by using the Intel® Cluster Tools. As a result, some parts of the application performance were improved considerably. In addition, EXASOL estimates that the development time and development efficiency has improved up to 30% using these tools.

References

- EXASOL AG: www.exasol.com
- Intel Corporation: www.intel.com
- Intel® Software Network: www.intel.com/software
- Intel® Software Development Products: www.intel.com/software/products
 - Intel® Cluster Tools: www.intel.com/go/clustertools
 - Intel® MPI Library: www.intel.com/go/mpl
 - Intel® Trace Analyzer and Collector / Correctness Checking: www.intel.com/traceanalyzer
 - Intel® Compilers
 - Intel® MKL Library
 - Intel® Threading Tools
- Intel® Cluster Ready: www.intel.com/go/cluster



For product and purchase information visit:
www.intel.com/software/products

Intel, the Intel logo, Intel. Leap ahead. and Intel. Leap ahead. logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Intel does not control or audit the design or implementation of third party benchmarks or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmarks are reported and confirm whether the referenced benchmarks are accurate and reflect performance of systems available for purchase.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel products are not intended for use in medical, life saving, life sustaining applications. Intel may make changes to specifications and product descriptions at any time, without notice.

Copyright © 2008, Intel Corporation. All Rights Reserved.