Interconnect Efficiency of Tyan PSC T-630 with Microsoft Compute Cluster Server 2003

Josef Pelikán
Charles University in Prague, KSVI Department, Josef.Pelikan@mff.cuni.cz

Abstract[1]

Interconnect quality is very important in distributed high-performance computing. We were interested in efficiency of Microsoft's implementation of Message-Passing Interface (MPI-2) included in Compute Cluster Server 2003. An experimental distributed application (master-slave model) was implemented and executed on a 20-core cluster equipped with Gigabit Ethernet interconnect. Using different configurations and setups we were able to estimate some parameters of real-world network traffic: overhead of single MPI-message, maximum network load on master node and speedup factors for our well-parallelizable problem.

Keywords: parallel programming, Compute Cluster Server, cluster interconnect, MPI, SHA-1, VUCAKO Bench

Introduction

Efficient parallel programming is very important objective today, when physical limits are limiting growth of single-core computing power. Therefore CPUs with more cores-on-die are manufactured and SIMD architectures or multi-node computer systems are used in high performance computing for a while..

We will be concentrating on multi-node (distributed) computing. Outline of the architecture is simple: there is number of processors (nodes), each of them has its own memory. Nodes are connected together by an "interconnect" (IC) network, grid or ring topologies (to mention the most important) are being used. Nodes need not have similar computing power or even the equal architecture. Google is one of well known users of massive heterogenous distributed computing systems.

Throughput of an IC appears to be most crucial attribute, influencing overall effectivity of the computation. Of course there are job classes more sensitive to IC efficiency (needing more inter-node communication), on the other hand many problems are well-parallelizable without intensive communication needs. Software architect should know technical parameters of distributed system and its IC in both cases.

There are several software libraries helping developers write programs for distributed systems efficiently and comfortably [1]. PVM (Parallel Virtual Machine) and MPI (Message-Passing Interface) are two main systems based on asynchronous message-passing between computing nodes. We will focus on MPI [2], especially on Microsoft's implementation of MPI-2 included in Compute Cluster Server 2003 [3].

One model for parallel programming of convenient problems is called "master-slave" (for details see any parallel-programming textbook or [4]). One node ("master") controls the computation and distributes work ("work units", WU) to identically operating "slave" nodes. There might be one type of WU, several WU types on the same level, or even complicated system of WU-s connected in a dependence graph. But this is not too important for our further research.

[1] Internal report describing technical research made in summer of 2007 on Tyan PSC T-630 cluster placed at Silicon Hill in Prague, 15 Nov 2007

A simple “master-slave” system must perform following actions in one WU-cycle ({m} stands for “master”, {s} for “slave”, {n} for “network”):

1. {m} assembly of input data (WU), concrete slave is defined
2. {n} WU is transferred to the slave
3. {s} WU is executed/computed by the slave
4. {n} results are transmitted back to the master
5. {m} result processing/merging, if applicable

Note that in case of asynchronous network transfer, master needs only to initiate the 2., other part of master's code will be waken at the end of the 4. Properly implemented master code should be busy in 1. and 5. only, slave code should be utilized in 3., then it initiates the 4. transfer. Details of one concrete problem will follow.

Distributed SHA-1 digest

We had chosen simple and well-parallelizable problem: distributed computing of SHA-1 digest (for hash functions see [5]). 32GB of randomly generated data were divided into 1 million working units, 32KB each. But input data volume (transferred over a network) is only 4MB, full input data is reconstructed on the slave's side. After decoding input data, slave computes SHA-1 digest using well known algorithm [6]. Result in form of 20-bytes long binary array is transferred back to the master, where it is accumulated (using binary XOR) into global result array. Thanks to commutativity of the XOR operation, order of result accumulation does not matter.

Latter method is the least effective one used in our measurements. SHA-1 digest of 32KB array takes only 200μs, which is comparable to message-passing overhead or network latency. Thus sending single WU in one MPI message is not very effective. We were introducing “working batches” containing 1, 2, 4, 10 or 100 WU-s. Input data size for one batch are 4, 8, 16, 40 or 400 bytes. Output (returning) data packet is always 20 bytes long, partial XOR is computed on the slave's side. Comparing computation efficiency of different batch sizes will allow us to estimate several IC parameters (single message overhead, latency).

Schematic code of the used master-worker system follows. Master pseudocode:

```
while ( anything-to-compute )
{
  MPI_Recv( slave, resultFromSlave );
  if ( resultFromSlave != GREETING_RESULT )
    mergeResult( resultFromSlave );
  prepareUnit( newUnit );
  MPI_Send( slave, newUnit );
}
for ( i = 1; i < numberOfNodes; i++ )
  MPI_Send( i, QUIT );
MPI_Finalize();
```

Slave pseudocode:

```
MPI_Send( master, GREETING_RESULT );
while ( true )
{
  MPI_Recv( master, unitToProcess );
  if ( unitToProcess == QUIT )
    break;
```

```
    result = computeUnit( unitToProcess );
    MPI_Send( master, result );
  }
  MPI_Finalize();
```

The actual C++ code was included into open-source VUCAKO Bench project [7]. Standalone test #64 can be compiled from "mpi64.cpp" source, binary executable is named "mpi64.exe" and should be deployed on Windows Compute Cluster Server [3] by command

```
job submit /numprocessors:12 /workdir:\\mscluster\<usr>$\
/stdout:out.txt mpiexec mpi64.exe -b <b>
```

where <usr> is actual login name and <b> batch size (1, 2, 4, 10, 100, ..). After job finish output text files will contain run-time statistics and timings.

The code of test #64 would probably run on other systems/MPI implementations as well. But we were tested it only on Windows CCS so far.

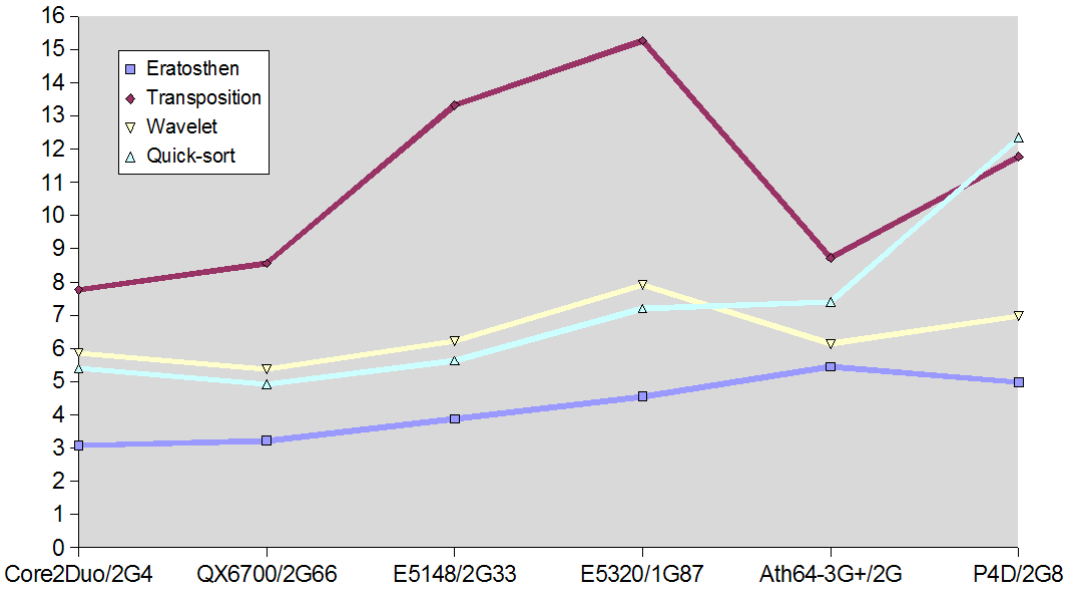
Test setup

We were using Tyan PSC T-630 cluster [8] installed at Silicon Hill (other details can be found in [9]). Some basic technical facts:

- 5 nodes (boards) connected together by dedicated Gigabyte Ethernet interconnect
- each board has two Intel XEON 5148 dual-core processors installed, clock: 2.33GHz
- each board has 2GB of DDR II/667MHz DRAM modules installed
- each board has a 80GB SATA-II hard drive attached (not used in our problem)
- operating system on head node: Microsoft Windows Compute Cluster Server 2003 [3], other nodes have regular Microsoft Windows Server Standard installed

Our experimental C++ code was compiled using Microsoft Visual Studio 2005 Professional, with help of additional libraries from the CCS installation (MPI communication). We had also performed many benchmark tests to rank single-core efficiency of the installed processors. VUCAKO Bench suite was used here too, detailed results can be found on its WWW pages [7].

For comparison purposes we will show a couple of benchmark results here (only CPU-related and memory-related ones, for details see [7]). Less numbers are better:



Distributed computing

Distributed SHA-1 digest was used for measuring practical efficiency of the interconnect. The "mpi64" task from VUCAKO Benchmark was compiled using Microsoft Visual Studio 2005 Professional and deployed several times in configurations with batch size 1 to 100 and utilizing 1 to 19 slave processes (100-batch configuration was considered to be "low-network-traffic" etalon, as it uses only hundreds of messages per second in average). Overall times in seconds were measured, averaged original times were:

	1-batch	2-batch	4-batch	10-batch	100-batch
1 slave (single chip)	212.5	209.1	206.5	205.5	**203.3**
3 slaves (single board)	70.8	69.7	68.9	68.4	**67.7**
7 slaves (two boards)	46.8	40.0	36.4	36.1	**29.5**
19 slaves (whole cluster)	27.3	20.3	17.1	15.4	**11.7**

100-batch configurations achieved parallel speedups: 3.00, 6.89, 17.43 (for 3, 7, 19 slaves) which is close to theoretical value for 4‰ of non-parallelizable code (Amdahl's law in [10]). Communication overhead is more pronounced in configurations with more than 3 slaves (when Ethernet interconnect has to be used).

Overview of network traffic for individual configurations (average number of MPI messages from master to all slaves per second : average incoming-message processing time on the master):

	1-batch	2-batch	4-batch	10-batch	100-batch
1 slave	4744:211µs	2410:415µs	1220:819µs	490:2039µs	50:20ms
3 slaves	14230:70µs	7233:138µs	3658:273µs	1473:679µs	149:7ms
7 slaves	**21522:46µs**	12594:79µs	6923:144µs	2789:359µs	342:3ms
19 slaves	**36958:27µs**	24775:40µs	14698:68µs	6563:152µs	864:1ms

Next table contains MPI-overhead estimates (total overhead in seconds : overhead per single MPI message in µs). Total overhead times were computed by comparing total computation time of a configuration to the "100-batch" standard:

	1-batch	2-batch	4-batch	10-batch	100-batch
1 slave	9.1 : 5µs	5.7 : 6µs	**3.2 : 6µs**	**2.2 : 11µs**	0.0 : 0µs
3 slaves	3.1 : 2µs	2.0 : 2µs	**1.2 : 2µs**	**0.7 : 4µs**	0.0 : 0µs
7 slaves	17.3 : 9µs	10.5 : 10µs	**6.9 : 14µs**	**6.6 : 33µs**	0.0 : 0µs
19 slaves	15.6 : 8µs	8.7 : 9µs	**5.5 : 11µs**	**3.7 : 18µs**	0.0 : 0µs

Comparing 4-batch and 10-batch to 100-batch case, we can see that the master is able to process effectively 10.000 of MPI messages per second (Mps) regardless of slaves' location (the same chip/board/different boards).

MPI overhead can be estimated to be less than 6µs per message for single-board configurations (≤3 slaves). Overhead ratio in such cases will be less than 4% of total computation time.

If interconnect had to be used (>3 slaves), one must expect much bigger communication overheads. Typical values measured in our experiments were between 8µs and 20µs per message. Overhead ratios can claim up to 50% of total computation time. Nevertheless we observed peak 37k Mps (27µs/message) throughput in this (least propitious) case.

Conclusions

Everyone recommends to keep number of MPI-messages as low as possible, but MPI implementation in CCS seems be quite effective even in non-ideal conditions (thousands of messages per second from single slave to the master). Intra-board communication seems to be quite effective as well, so one has no great need to optimize the application in a "hybrid-parallelization" way, except for special algorithms which can take big advantage from sharing memory between worker threads.

We can expect network overhead to be as low as 10μs per message even in cases with fully loaded master processes (thousands of MPI-messages per second per slave).

Future work: equivalent tests will be performed on a 80-processor blade server equipped with an InfiniBand interconnect [11]. After that we will be able to give more complete study of MPI efficiency and influence of interconnect technology to overall effectivity of a master-slave solution.

Acknowledgements

This work was supported by Microsoft (see [9]) and Silicon Hill (www.siliconhill.cz).

References

1. Peter Kacsuk, Ferenc Vajda. *Network-based Distributed Computing (Metacomputing).* ERCIM Hungary, 1999
2. Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface.* 15 Nov 2003, on-line version can be found at: http://www.mpi-forum.org/
3. *Microsoft Windows Compute Cluster Server 2003,* main WWW page, http://www.microsoft.com/windowsserver2003/ccs/
4. Gary Shao, Francine Berman. *Master/Slave Computing on the Grid.* Heterogeneous Computing Workshop, 2000
5. Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* Wiley, 2nd Edition, 1995, ISBN: 0471117099
6. Steve Reid's public domain implementation of SHA-1 digest algorithm, 27 Sep 1996, one of present URLs: http://svn.ghostscript.com/jbig2dec/tags/release_0_04/sha1.c
7. *VUCAKO Bench*, open-source benchmark suite for 32- and 64-bit computers, homepage, http://cgg.ms.mff.cuni.cz/~pepca/bench/
8. *Scalable Servers Corporation*, main WWW page, http://www.tyanpsc.com/
9. *Windows Compute Cluster Server 2003.* Silicon Hill WiKi: http://wiki.siliconhill.cz/Windows_Compute_Cluster_Server_2003
10. Amdahl, G.M. *Validity of the single-processor approach to achieving large scale computing capabilities.* In AFIPS Conference Proceedings vol. 30 (Atlantic City, N.J., Apr. 18-20). AFIPS Press, Reston, Va., 1967, pp. 483-485.
11. *InfiniBand Trade Association*, WWW pages, http://www.infinibandta.org/itinfo/