Performance Evaluation of Parallel Processing Environment for Molecular Dynamics

KENJI SATOU[1], KENRI KONNO[2], OSAMU OHTA[3], KAZUNORI MIKAMI[4], KEITA TERANISHI[5], YOICHI YAMADA[1], SHIN-YA OHKI[6]

1 Graduate School of Natural Science and Technology, Kanazawa University
Kakuma-machi, Kanazawa 920-1192, JAPAN
ken@t.kanazawa-u.ac.jp   http://bioinfo.ec.t.kanazawa-u.ac.jp/~ken/

2 Graduate School of Materials Science
3 Graduate School of Information Science
6 Center for Nano Materials and Technology
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa 923-1292, JAPAN

4 Cray Japan Inc.,
1-2-2 Uchisaiwaichou, Chiyoda-ku, Tokyo 100-0011, JAPAN

5 Cray Inc.
Mendota Heights, Minnesota 55120, U.S.A.

Abstract: - Molecular dynamics (MD) is one of the popular applications in the research field of high performance computing. Since it requires large amount of CPU time basically proportional to the square of the number of atoms simulated, acceleration of MD is essential to simulation of large biomolecules like proteins. Therefore, parallelization of MD has been actively studied long time. However, most of the studies of parallel MD report modified or newly developed algorithms specialized to some computer architectures like vector-parallel supercomputer, and an end-user of MD software cannot implement them to popular MD software developed by other ones. In this study, we evaluated performance of four kinds of computer architectures: 1) vector-parallel supercomputer, 2) multi-processor machine with shared memory, 3) multi-processor machine with distributed memory, and 4) PC cluster. Various compiler options for parallelization and optimization were tested. Experimental results revealed that if MD software is not parallelized nor vectorized in source level, use of normal PC cluster with maximum use of optimization options in compilation is the best way.

Key-Words: - Molecular dynamics software, Computer architecture, Parallel processing, Optimization

1 Introduction

As the success of Folding@Home project [1] demonstrates, there is a great demand of biomolecule analysis through molecular dynamics (MD) and efforts have been concentrated on the development of improved algorithm and software [2]. There exist many MD tools: AMBER [3] and CHARMM [4] are the most famous software suites, Tinker [5] and Gromacs [6] are relatively more simple and easy-to-use, myPresto [7] and Peach [8] were developed in Japan, and so on. These software tools are useful for both of commercial development of new pharmaceuticals and academic research in structure and function of biomolecules [9]. Except CHARMm, the tools above are free of charge or distributed at fairly low cost for the purpose of academic research. So, it is popular to personally install one of them and use it also personally or share it in a laboratory. However, even with today's computers dramatically improved in performance, it is still tough computation to solve the structure of large biomolecule like protein with huge amount of water molecules as solvent surrounding it. Therefore, acceleration techniques for MD have been actively studied.

There are many previous works on the acceleration of MD computation. They can be roughly classified into two categories: reduction of computation and parallel computation. In general, application of a technique in the former is limited since there must be a trade-off between reduction and precision of computation. The latter can be divided into finer categories: 1) parallel computing by vector processor, 2) parallel computing on a computer with multiple CPUs, 3) parallel computing on multiple computers connected with LAN (i.e. PC cluster), and 4) parallel computing on multiple computers connected with WAN (i.e. Grid computing).These are also in the historical order of trends in research and development of MD acceleration techniques. Once a PC was too poor to perform MD, and it was studied to make the best use of a supercomputer with one or a few vector processors for this purpose. After that, a multi-processor machine which has shared or distributed memory and multiple scalar processors connected with high-speed channel and switch became common. As a result, MD acceleration techniques by multiprocessing and/or multithreading were actively studied. Though a programming completely different from vector-parallel processing is required, this approach achieved considerable success by the high-speed communication mechanism and large memory capacity. Utilization of PC cluster can be a natural extension of this approach in significantly lower cost. To hide the latency of LAN, it is popular to use Myrinet instead of Ethernet and high-performance network communication library like SCore. Parallel processing on PC cluster via MPI library is also popular in other application domains [10,11].

Though there are various previous works, it is difficult to compare experimental results to each other since they were measured on different computer architectures. In addition, most of the acceleration techniques reported in papers require source-level modification of MD software tools, and unable to be reproduced without deep understanding of source code and parallel programming. Therefore, there is no clear guideline for a biochemist to choose best computer architecture for MD computation. Furthermore, in case of a MD software tool provided as source code (e.g. AMBER), choice of optimization options in compilation of the source code might greatly affect to the speed of MD computation.

Based on the above backgrounds, in this study we measured and compared performances of MD computation with various combinations of machine architectures, parallelization techniques, and optimization options. Except an architecture which definitely requires minimum modification to run the code, the same MD software tool was used without source code modification in the experiment. By avoiding source code modification as much as possible, the experimental results in this paper revealed a guideline for a biochemist to choose the best machine architecture for MD.

2 myPresto and cosgene

In this study, we adopted myPresto Version 3 as MD software tool for performance measurement. myPresto is distributed free of charge for non-commercial use at University. Among programs in myPresto, cosgene performs MD computation. myPresto is provided as source code and executables precompiled in some platforms. To compile cosgene from source code, Fortran 90 is needed. From one source code, an executable for serial computation or an executable for parallel computation via MPI library can be generated depending on configuration parameter. Hereinafter, we call the executables for serial and parallel computations cosgene_serial and cosgene_MPI, respectively.

About vectorization, it was reported that Presto, the predecessor of myPresto, was originally vectorized and achieved high performance on supercomputers like NEC SX series and Fujitsu VP series. However, source code of myPresto is basically independent from Presto and does not include vectorized codes.

3 Computer Platforms

We used the following four computer platforms with different architectures and operating systems.

NEC SX-8

This machine (Fig.1) is a descendant of SX-5 which share almost the same vector processors with the Earth Simulator [12]. SX-8 realizes peak vector performance of 16Gflops per CPU (vector processor). In the experiment, we used a model of SX-8 with 8 CPUs and 64GB memory. In case of interactive use, all the 8 CPUs are available, while 6 CPUs in batch processing via a queueing system NQSII. Operating system is SUPER-UX.

Fig.1. SX-8

SGI Altix 3700

This machine (Fig.2) is classified as shared memory multi-processor computer. The model we used contains 32 C-blicks connected with NUMAlink3, where each C-blick has four Itanium2 processors (1.6GHz) and 24GB memory. In total, this model provides 128 CPUs and 768GB shared memory.

Fig.2. Altix 3700

Cray XT3

This machine (Fig.3) is classified as distributed memory multi-processor computer. The model we used contains 90 nodes connected in 3D torus link, where each node has four Opteron 150 processors (2.4GHz) and 32GB memory.

Fig.3. XT3

Appro HyperBlade Mid-Cluster

This machine (Fig.4) is classified as PC cluster. The model we used contains 32 PCs connected with Gigabit Ethernet, where each PC has two Opteron DP Model 250 processors (2.4GHz) and 4GB memory.

Fig.4. HyperBlade Mid-Cluster

4 Parallel Computation Types

We tried the following types of parallel computation for the platforms described in the previous section.

NEC SX-8

- vector-parallel processing through automatic vectorization by compiler with -Chopt option.
- process- or thread-parallel processing through automatic parallelization by compiler with -Pauto option.
- process- or thread-parallel processing conducted by cosgene_MPI.
- combination of these types.

SGI Altix 3700

- process- or thread-parallel processing through automatic parallelization by compiler with –parallel option.
- process- or thread-parallel processing conducted by cosgene_MPI.
- combination of these types.

Cray XT3

- process- or thread-parallel processing conducted by cosgene_MPI (minimum modification is applied to source code of cosgene to run it on XT3).

Appro HyperBlade Mid-Cluster

- process- or thread-parallel processing conducted by cosgene_MPI without compilation (i.e. provided executable was used as is).

5 Compilers and Options

In the configuration of cosgene, we typically specified the following compilers and options for each platform, where FC and FC_MPI denote the name of Fortran 90 compiler for cosgene_serial and cosgene_MPI, respectively, and OPT denotes optimization options passed to compiler. PP=fpp is a special option only for ifort to invoke preprocessor. For more details about options, see the manual of each compiler.

NEC SX-8

- FC = f90
- FC_MPI = mpi90
- OPT = -C debug -D_SMALL_SYSTEM
- combinations of -g (debug), -Chopt (full use of optimization and vectorization upper limits), -Cnoopt (no vectorization and optimization), -Cvsafe (very safe use of optimization and vectorization without side effect), -EP (C preprocessor activation), -pi auto (automatic inline expansion), and -Pauto (automatic parallelization) were tried as additional options.

SGI Altix 3700

- FC = ifort
- FC_MPI = ifort
- OPT = -O2 -static
- PP = -fpp
- -parallel (automatic parallelization) was tried as an additional option.

Cray XT3

- FC = ftn
- FC_MPI = ftn
- OPT = -fast -fastsse -O3 -mcmodel=medium

Appro HyperBlade Mid-Cluster

- FC = pgf95
- FC_MPI = mpif90
- OPT = -fast -fastsse -O3

6 Protein Molecule for MD

For MD computation of biomolecule, we adopted a protein called myosin phosphatase inhibitor CPI-17 with Thr38 replaced with Asp [13]. 1j2m is the PDB code of this protein containing 99 residues (Fig.5). After energy minimization, a new conformation 1j2m_min was prepared and input to cosgene_serial and cosgene_MPI (Fig.6). In MD computation, a force field parameter C99_aa.tpl was adopted, which contains topology information for all amino acid monomers for the AMBER96 force field. 100ps MD simulation was performed in each experiment. Fig.7 shows an example of conformation after 100ps simulation.

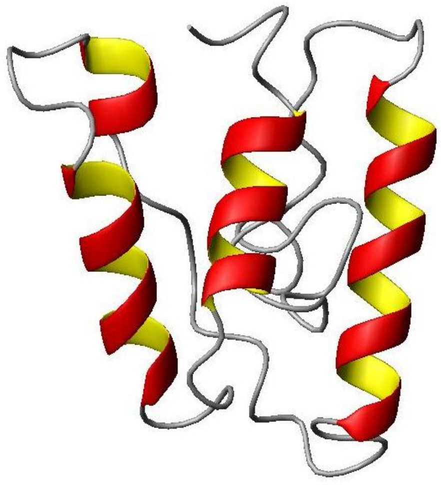

Fig.5. Conformation of 1j2m

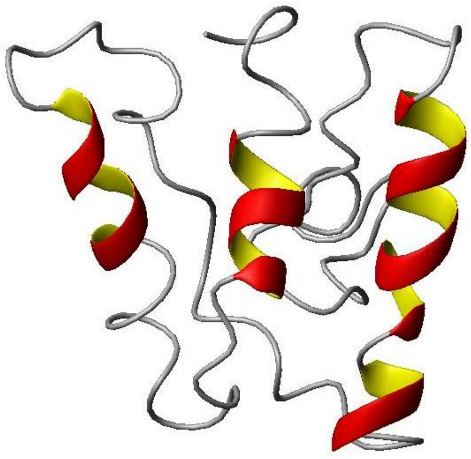

Fig.6. Conformation of 1j2m_min

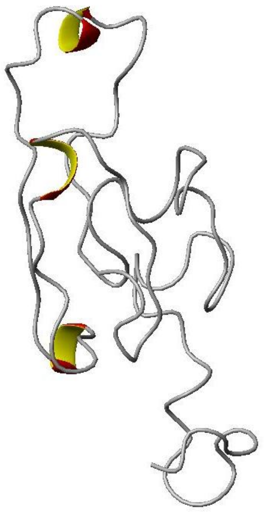

Fig.7. Conformation of 1j2m_min after 100ps MD simulation

7 Experimental Results

As a control point, in each platform we measured computation time of cosgene_serial compiled with only the typical options listed in section 5 (i.e. without any additional options for parallelization and optimization). The number of CPUs allocated for computation was one. Table 1 shows the result of computation, where HBMC denotes Appro HyperBlade Mid-Cluster. In Table 1, HBMC is significantly faster than others. In contrast, SX-8 is very slow, however, it is not surprising since a vector-processor has only a poor scalar performance in general. Though Itanium2 and Opteron are CPUs with different characteristics, computation times in Altix and XT3 can be explained in terms of CPU clocks (1.6GHz and 2.4GHz).

Table 1. Computation time of typical cosgene_serial with 1CPU

	Computation time (second) with 1 CPU
SX-8 cosgene_serial	135823
Altix cosgene_serial	20452
XT3 cosgene_serial	14141
HBMC cosgene_serial	4357

Next, various combinations of parallel processing and options were tested on each platform. Fig.8 illustrates the effect of –parallel option and/or MPI parallel processing in Altix. From this figure, we can see the following:

- –parallel option causes indispensable overhead. From 1 to 3 CPUs, only the overhead was observed. In case of 4 CPUs, it causes some speed-up in comparison with 1~3 CPUs, however, still slower than 1CPU without –parallel option.
- MPI parallel processing seems to work well, however, -parallel option cannot achieve further acceleration in combination with it.

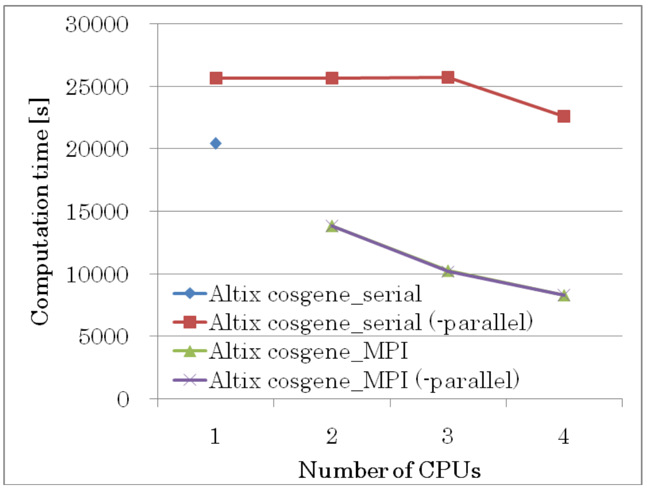

Fig.8. Effect of MPI and –parallel option in Altix

For SX-8, also various options were tested but cosgene_serial and cosgene_MPI were executed only with 1CPU and 4CPUs, respectively (Fig.9). From this figure, we can see the following:

- –Chopt and –Cvsafe are effective to reduce the computation time. However, since these options perform both of optimization and vectorization, ratio of their contribution is unclear.
- MPI parallel processing seems to work well. Moreover, –Cvsafe is also effective with MPI.
- In contrast to –parallel that is not effective in Altix, -Pauto in SX8 is significantly effective. However, it does not accelerate cosgene_MPI.
- –Pauto and –Chopt seems to be interfering to each other.
- Optimization by automatic inline expantion (-pi auto) is quite effective in both of cosgene_serial and cosgene_MPI though it does not perform any vectorization.

Unlike Altix and SX-8, we used only typical options for XT3 and HBMC since there were no vectorization and parallelization options for them.

Besides compiler options, Fig.10~13 illustrate scalability of MPI parallel processing in four platforms. In all platforms, as the number of CPUs increases, acceleration effect by adding CPU decreases. It shows that at least this version of cosgene_MPI cannot achieve linear speed-up.

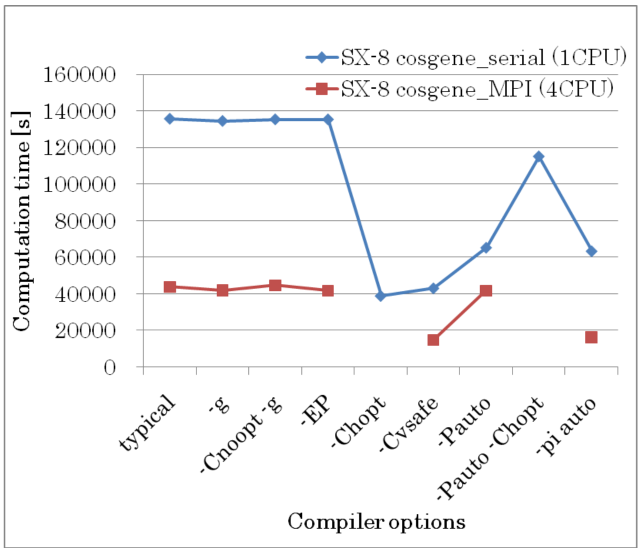

Fig.9. Effect of MPI and various options in SX-8

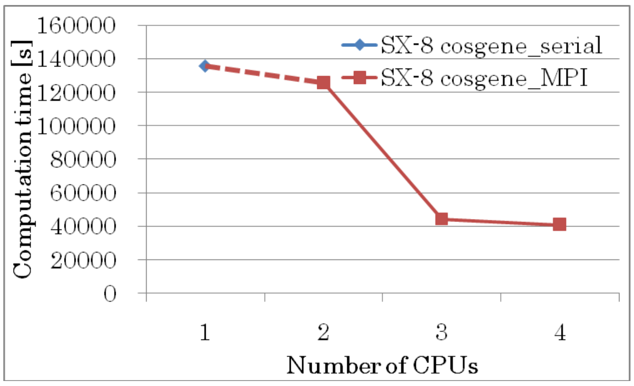

Fig.10. Effect of MPI in SX-8 (with typical parameters only)

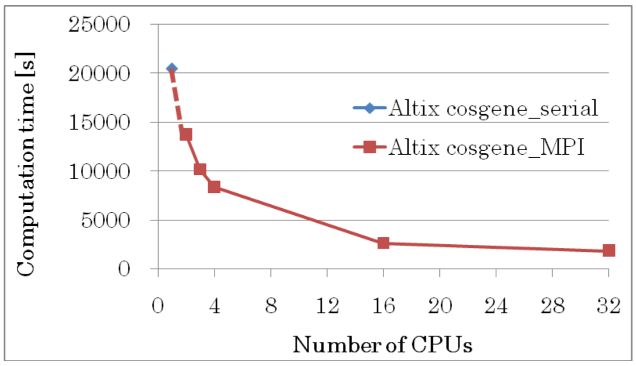

Fig.11. Effect of MPI in Altix

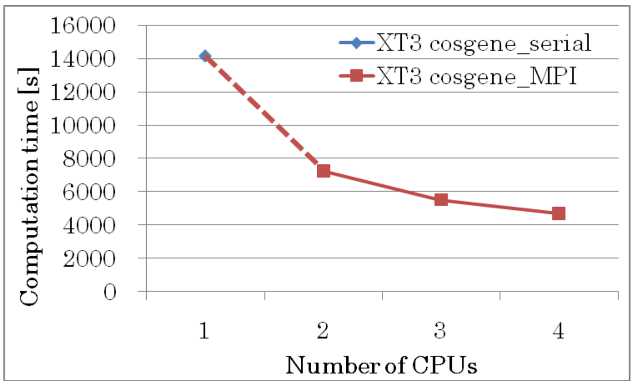

Fig.12. Effect of MPI in XT3

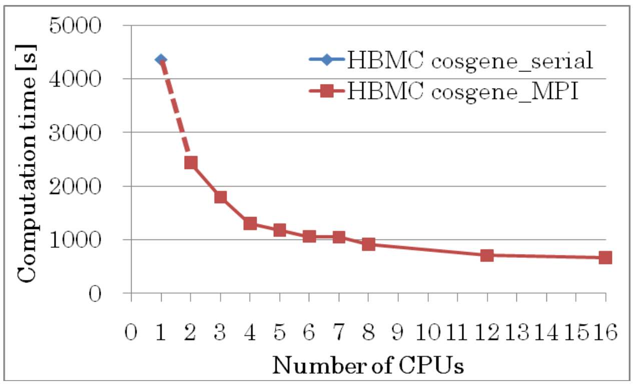

Fig.13. Effect of MPI in HBMC

Finally, Table 2 summarizes ratio of control point (1CPU, cosgene_serial, typical options only) and best performance (4CPUs, cosgene_MPI, additional options allowed). Here we see that acceleration was possible in SX-8, Altix, and XT3, however their best performances were lower than the control point of HBMC.

Table 2. Accerelation ratio

	Control (second)	Best (second)	ratio
SX-8	135823	14803 (-Cvsafe)	9.18
Altix	20452	8301	2.46
XT3	14141	4700	3.01
HBMC	4357	1306	3.34

8 Computation Characteristics in XT3

Among four architectures studied in this paper, a more detailed analysis was conducted on XT3. In this analysis, we used sample2 included in the source distribution of myPresto. The protein used in this MD simulation is 1lza, a hen egg-white lysozyme containing 129 residues (Fig.14). After energy minimization, a new conformation lys_1_min was prepared and input to cosgene_serial and cosgene_MPI (Fig.15). The topology file lys_1.tpl is also provided and used in MD computation. Fig.16 shows an example of conformation after MD simulation.

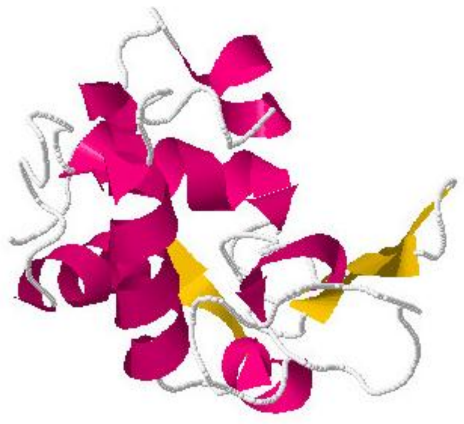
Fig.14. Conformation of 1lza

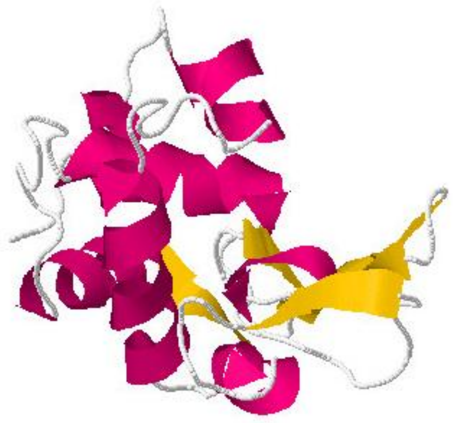
Fig.15. Conformation of lys_1_min

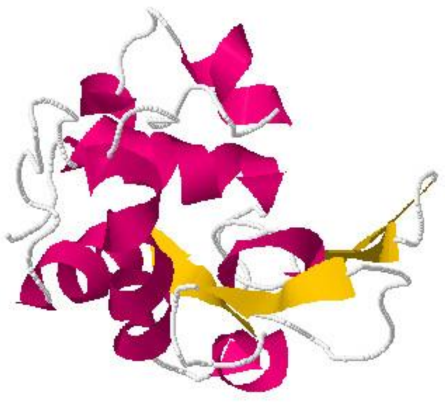
Fig.16. Conformation of lys_1_min after MD simulation

First, hardware performance counter statistics was measured by using CrayPat tool. Tables 3 and 4 show the statistics in the execution of cosgene_serial by 1 CPU and cosgene_MPI by 8 CPUs, respectively. In Table 3, it can be seen that the power of single Opteron processor in XT3 is well utilized since MIPS around 1390, MFLOPS 574.16, 12% peak, and D1 cache hit ratio 99.3% are substantially high. In Table 4, execution time was improved from around 52 seconds to 14. However, MFLOPS per CPU was decreased to 309.16. It implies that due to some bottlenecks in parallel processing, the power of each CPU was not fully utilized. Possible reasons are as follows: 1) load of communication for parallel processing, 2) load of MPI data transfer is higher than that of computation in MD algorithm, and 3) synchronization wait caused by imbalance of computation load in each CPU.

Table 3. Hardware counter statistics of cosgene_serial

Attribute	Value
Time%	100.0%
Time	51.910235
Calls	404
PAPI_TOT_INS	1389.963M/sec, 72151638848
PAPI_L1_DCA	491.549M/sec, 25515840364
PAPI_FP_OPS	574.160M/sec, 29804104404
DC_MISS	3.439M/sec, 178489608 misses
User time	51.909 secs, 124581673855
Utilization rate	100.0%
Instr per cycle	0.58 inst/cycle
HW FP Ops / Cycles	0.24 ops/cycle
HW FP Ops / User time	574.160M/sec, 29804104404 ops, 12.0%peak
HW FP Ops / WCT	574.147M/sec
HW FP Ops / Inst	41.3%
Computation intensity	1.17 ops/ref
MIPS	1389.963M/sec
Instructions per LD	2.83 inst/ref
LD & ST per D1 miss	142.95 refs/miss
D1 cache hit ratio	99.3%
LD ST per	35.4%

Table 4. Hardware counter statistics of cosgene_MPI (8CPUs)

Attribute	Value
Time%	100.0%
Time	14.120283
Imb.Time	0.022806
Imb.Time%	0.2%
Calls	404
PAPI_TOT_INS	2174.807M/sec, 30708193764
PAPI_L1_DCA	883.885M/sec, 12480427080
PAPI_FP_OPS	309.160M/sec, 4365321986 ops
DC_MISS	10.265M/sec, 144941315
User time	14.120 secs, 33887906726
Utilization rate	100.0%
Instr per cycle	0.91 inst/cycle
HW FP Ops / Cycles	0.13 ops/cycle
HW FP Ops / User time	309.160M/sec, 4365321986 ops, 6.4%peak
HW FP Ops / WCT	309.153M/sec
HW FP Ops / Inst	14.2%
Computation intensity	0.35 ops/ref
MIPS	2174.807M/sec
Instructions per LD	2.46 inst/ref
LD & ST per D1 miss	86.11 refs/miss
D1 cache hit ratio	98.8%
LD ST per	40.6%

Then, we examined hot spots in cosgene program. Table 5 shows the result of level 0~1 analysis of time consumption. It clearly shows that more than half of execution time is consumed by MPI data transfer and MPI synchronization wait.

Besides overheads in MPI data transfer and synchronization, the result of detailed analysis on the subroutines in USER category for MD computation is shown in Table 6. In this table, time consumption of level 2 subroutines is also reported. Subroutines with Time% lower than 1% are omitted. The most time consuming subroutine is written in bold face and occupies 40.9% of MD computation. Hardware counter statistics of the subroutine is shown in Table 7.

Table 5. Breakdown of total hardware counter statistics into three categories (USER, MPI_SYNC, and MPI)

Level	Time %	Time	Imb. (load imbalance	FLOPs	MFLOPS	Calls	Group
0	100.0%	19.252263	--	4348227192	225.83	1611002	Total
1	48.2%	9.288503	--	4348172210	468.02	1599209	USER
1	37.0%	7.119652	--	0	0	2511	MPI_SYN
1	14.8%	2.844108	--	54982	0.02	9282	MPI

Table 6. Hardware counter statistics of time consuming subroutines for MD computation

Level	Time %	Time	Imb. (load imbalance)	FLOPs	MFLOPS	Calls	Group or Function
0	100.0%	19.2523	--	4348227192	225.83	1611002	Total
1	48.2%	9.2885	--	4348172210	468.02	1599209	USER
2	**40.9%**	**3.794393**	**14.8%**	**3785336107**	**997.69**	**1156**	**fast_nonbonded_calc_vdwhyddependelecutoff**
2	23.3%	2.168139	4.9%	0	0	3	communicate_method_broadcast_mddata_
2	9.3%	0.866093	17.9%	438123097	505.7	1	child_process_exec_minloop_
2	4.2%	0.39278	18.6%	0	0	845643	communicate_method_trans_bufferint_
2	4.2%	0.38561	100.0%	105769	0.27	0	input_common_input_commonfile_
2	3.4%	0.31959	26.0%	133958	0.42	222762	interacttable_method_register_residueatoms_
2	3.0%	0.27452	9.5%	1508	0.01	1156	fast_nonbonded_calc_cutoff15interactenergy_
2	2.4%	0.22192	19.4%	31062159	139.37	59	interacttable_method_update_residuesurfacecutoff_
2	1.9%	0.1761	100.0%	78700396	446.5	0	minimize_method_exec_conjugategradient_
2	1.6%	0.14903	100.0%	7656092	51.37	29	monitoring_monitor_minimize_
2	1.6%	0.14787	34.9%	0.75	0	222762	interacttable_method_reset_flagfixatomorbonded_
2	1.3%	0.12116	20.9%	0	0	240069	communicate_method_trans_bufferreal_

Table 7. Hardware counter statistics of the function fast_nonbonded_calc_vdwhyddependelecutoff_ in cosgene_MPI (8CPUs)

Attribute	Value
Time%	40.9%
Time	3.794393
Imb.Time	0.562844
Imb.Time%	14.8%
Calls	1156
PAPI_TLB_DM	42.011M/sec, 159395253
PAPI_L1_DCA	622.910M/sec, 2363388747
PAPI_FP_OPS	997.687M/sec, 3785336107
DC_MISS	3.738M/sec, 14183954 misses
User time	3.794 secs, 9105864953 cycles
Utilization rate	100.0%
HW FP Ops / Cycles	0.42 ops/cycle
HW FP Ops / User time	997.687M/sec, 3785336107 ops, 20.8%peak
HW FP Ops / WCT	997.613M/sec
Computation intensity	1.60 ops/ref
LD & ST per TLB	14.83 refs/miss
LD & ST per D1 miss	166.62 refs/miss
D1 cache hit ratio	99.4%
% TLB misses / cycle	1.8%

About the most time consuming subroutine shown in Table 6, we confirmed in practice that 5% acceleration in this subroutine is possible by source-level modification. Since the subroutine occupies around 20% of total execution time (40.9% of 48.2% equals to 19.7%), this improvement is around 1% of total execution time. It means that the original source code of cosgene is sufficiently tuned and hard to drastically improve.

9 Conclusion

In this study, we tested various combinations of parallel processing and optimization options on four different computer architectures, i.e. a vector supercomputer, multi-processor supercomputer with shared and distributed memories, and a PC cluster. Experimental results revealed superiority of PC cluster against other expensive supercomputers. However, scalability of MPI parallel was not so promising. Similarly, automatic vectorization was not so effective since in comparison with acceleration by -Chopt, around 80% of it can also be achieved by a simple optimization, i.e. inline expansion by -pi auto. It implies that percentage of vectorization by compiler might be low. In other words, though a supercomputer with huge memory is still needed to solve a fine structure of extremely large biomolecules, a common PC with a dual- or quad-core processor and large memory (4GB or more) is one of the competitive alternatives to solve a structure of relatively smaller biomolecules by using a popular MD software tool like myPresto.

References:

[1] S.M. Larson, C.D. Snow, M. Shirts and V.S. Pande., Folding@Home and Genome@Home: Using distributed computing to tackle previously intractible problems in computational biology, Computational Genomics, Horizon Press, 2002.

[2] S.A. Adcock and J.A. McCammon, Molecular Dynamics: Survey of Methods for Simulating the Activity of Proteins, Chem. Rev., Vol.106, No.5, 2006, pp.1589-1615.

[3] D.A. Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham III, S. DeBolt and D. Ferguson, G. Seibel and P. Kollman, AMBER, a Package of Computer Programs for Applying Molecular Mechanics, Normal Mode Analysis, Molecular Dynamics and Free Energy Calculations to Simulate the Structural and Energetic Properties of Molecules, *Comp. Phys. Commun.*, Vol.91, 1995, pp.1-41.

[4] B.R. Brooks, R.E. Bruccoleri, B.D. Olafson, D.J. States, S. Swaminathan and M. Karplus, CHARMM: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations, *J. Comput. Chem.*, Vol.4, 1983, pp.187-217.

[5] J.W. Ponder and F.M. Richards, An efficient Newton-like method for molecular mechanics energy minimization of large molecules, *J. Comput. Chem.*, Vol.8, 1987, pp.1016–1024.

[6] B. Hess, C. Kutzner, D. van der Spoel and E. Lindahl, GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation, *J. Chem. Theory Comput.*, Vol.4, No.3, 2008, pp.435-447.

[7] Y. Fukunishi, Y. Mikami and H. Nakamura, The filling potential method: A method for estimating the free energy surface for protein-ligand docking, *J. Phys. Chem. B.*, Vol.107, 2003, pp.13201-13210.

[8] Y. Komeiji, M. Uebayasi, R. Takata, A. Shimizu, K. Itsukashi and M. Taiji, Fast and accurate molecular dynamics simulation of a protein using a special-purpose computer, *Journal of Computational Chemistry*, Vol.18, Issue 12, 1998, pp.1546-1563.

[9] T. Schröder, A. Quintilla, J. Setzler, E. Birtalan, W. Wenzel and S. Bräse, Joint experimental and theoretical investigation of the propensity of peptoids as drug carriers, *WSEAS Transactions on Biology and Biomedicine*, Vol.4, Issue 10, 2007, pp.145-148.

[10] I. Aziz, N. Haron, L.T. Jung and W.R.W. Dagang, Parallelization of Prime Number Generation Using Message Passing Interface, *WSEAS Transactions on Computers*, Vol.7, Issue 4, 2008, pp.291-303.

[11] K.L. Hsieh, W. Tsai and N.M. Shih, Applying PC-cluster into Clustering Analysis for Organism's Codon Usage Based on MPI Techniques, *WSEAS Transactions on Computers*, Vol.6, Issue 8, 2007, pp.1044-1049.

[12] S. Shingu, H. Takahara, H. Fuchigami, M. Yamada, Y. Tsuda, W. Ohfuchi, Y. Sasaki, K. Kobayashi, T. Hagiwara, S. Habata, M. Yokokawa, H. Itoh and K. Otsuka, A 26.58 Tflops Global Atmospheric Simulation with the Spectral Transform Method on the Earth Simulator, *Proc. of the ACE/IEEE SC2002 conference*, 2002.

[13] S. Ohki, M. Eto, M. Shimizu, R. Takada, D.L. Brautigan and M. Kainosho, Distinctive Solution Conformation of Phosphatase Inhibitor CPI-17 Substituted with Aspartate at the Phosphorylation-site Threonine Residue, J.*Mol.Biol.*, Vol.326, Issue 5, 2003, pp.1539-1547.