

Molecular Dynamics Simulations on Cray Clusters using the SCIDDLE-PVM environment

Peter Arbenz¹, Martin Billeter³, *PeterGüntert*³,
Peter Luginbühl³, *MichelaTaufner*^{1,2}, *Urs von Matt*¹

August 6, 1999

¹ Institute of Scientific Computing, ² Università degli Studi di Padova,
Swiss Federal Institute of Technology (ETH), Dipartimento di Elettronica ed Informatica,
8092 Zürich, Switzerland, Via Gradenigo 6/a, Padova, Italy,
email: [arbenz,vonmatt]@inf.ethz.ch email: taufer@dei.unipd.it

³ Institute of Molecular Biology and Biophysics,
Swiss Federal Institute of Technology (ETH),
8093 Zürich, Switzerland,
email: [billeter,guentert,lugi]@mol.biol.ethz.ch

1 Introduction

The computer simulation of the dynamical behavior of large molecules is an extremely time consuming task. Only with today's high-performance computers has it become possible to integrate over a realistic time frame of about a nanosecond for relatively small systems with hundreds or thousands of atoms. The simulation of the motion of very large atoms from biochemistry is still out of reach. Nevertheless, with every improvement of hardware or software computational chemists and biochemists are able to attack larger molecules.

In this paper we report on an effort to parallelize OPAL [6], a software package developed at the Institute of Molecular Biology and Biophysics at ETH Zürich to perform energy minimizations and molecular dynamics simulations of proteins and nucleic acids in vacuo and in water. OPAL uses classical mechanics, i.e., the Newtonian equations of motion, to compute the trajectories $\vec{r}_i(t)$ of n atoms as a function of time t . Newton's second law expresses the acceleration as

$$m_i \frac{d^2}{dt^2} \vec{r}_i(t) = \vec{F}_i(t), \quad (1)$$

where m_i denotes the mass of atom i . The force $\vec{F}_i(t)$ can be written as the negative gradient of the atomic interaction function V :

$$\vec{F}_i(t) = -\frac{\partial}{\partial \vec{r}_i(t)} V(\vec{r}_1(t), \dots, \vec{r}_n(t)).$$

A typical function V has the form [10]

$$\begin{aligned}
 V(\vec{r}_1, \dots, \vec{r}_n) = & \sum_{\text{all bonds}} \frac{1}{2} K_b (b - b_0)^2 + \sum_{\text{all bond angles}} \frac{1}{2} K_\Theta (\theta - \theta_0)^2 + \\
 & \sum_{\text{improper dihedrals}} \frac{1}{2} K_\xi (\xi - \xi_0)^2 + \sum_{\text{dihedrals}} K_\varphi (1 + \cos(n\varphi - \delta)) + \\
 & \sum_{\text{all pairs } (i, j)} \left(\frac{C_{12}(i, j)}{r_{ij}^{12}} - \frac{C_6(i, j)}{r_{ij}^6} + \frac{q_i q_j}{4\pi \epsilon_0 \epsilon_r r_{ij}} \right).
 \end{aligned}$$

The first term models the covalent bond-stretching interaction along bond b . The value of b_0 denotes the minimum-energy bond length, and the force constant K_b depends on the particular type of bond. The second term represents the bond-angle bending (three-body) interaction. The (four-body) dihedral-angle interactions consist of two terms: a harmonic term for dihedral angles ξ that are not allowed to make transitions, e.g., dihedral angles within aromatic rings or dihedral angles to maintain chirality, and a sinusoidal term for the other dihedral angles φ , which may make 360° turns. The last term captures the non-bonded interactions over all pairs of atoms. It is composed of the van der Waals and the Coulomb interactions between atoms i and j with charges q_i and q_j at a distance r_{ij} .

In a numerical simulation the equations (1) are integrated in small time steps Δt , typically 1–10 fs for molecular systems. Therefore a realistic simulation of 1000 ps requires up to 10^6 integration time steps.

The last term of V , i.e., the sum over all pairs of atoms, consumes most of the computing time during a simulation. Van Gunsteren and Mark report [10] that in 1992 the evaluation of the interactions between the pairs of atoms of a 1000-atom system required about 1 s. Therefore at least 300 h were needed to execute 10^6 time steps. These order-of-magnitude estimates are still applicable today.

Fortunately, these calculations also offer a high degree of parallelism. In OPAL we evaluate the non-bonded interactions in parallel on p processors. Each processor receives the coordinates $\vec{r}_i(t)$ of all the atoms, and it computes a subset of the interactions. Thus we can execute $O(n^2)$ arithmetic operations on each server, whereas only $O(n)$ data items need to be communicated. Consequently we may expect a significant speedup for a large number n of atoms.

2 SCIDDLE

The SCIDDLE environment [2, 3, 9] supports the parallelization of an application according to the client-server paradigm. It provides asynchronous remote procedure calls (RPCs) as its only communication primitive. In SCIDDLE, an application is decomposed into a client process and an arbitrary number of server processes. Servers are special processes that are waiting to execute RPC requests from their client. Servers can also start other servers themselves. Thus the topology of a SCIDDLE application can be described by a tree structure.

The interface between client and server processes is described by a SCIDDLE interface definition (cf. Sect. 3). A compiler translates this interface definition into communication stubs for the client and server processes. Error checking is performed both at compile-time and at runtime.

```

INTERFACE OPAL_Server;

CONST Max = 10000;

TYPE Matrix = ARRAY [3, Max] OF LONGREAL;

PROCEDURE nbint (IN nval                : INTEGER;
                 IN atcor [1:1:3, 1:1:nval] : Matrix;
                 OUT atfor [1:1:3, 1:1:nval]: Matrix): ASYNC;

END

```

Figure 1: Remote Interface Definition

SCIDDLE-PVM uses the PVM system to implement asynchronous RPCs. An application only needs to use PVM to start the server processes. No explicit message passing is necessary any more since all the communication is performed through SCIDDLE. Thus SCIDDLE applications benefit from the safety and ease of use of RPCs. They are also exceedingly portable as PVM becomes available on more and more platforms. In [1] it is shown that the overhead introduced by SCIDDLE is minimal and can be neglected for applications with large messages.

In recent years, PVM has become a de-facto standard for distributed applications. Its wide acceptance has lead numerous computer vendors to provide high-performance implementations. For the Intel Paragon and the IBM SP/2, say, PVM implementations exist that sit directly on top of the native message passing interface [5]. Cray Research offers a version of PVM tuned for the Cray J90 SuperCluster [8]. Applications based on SCIDDLE-PVM will be able to exploit these optimized PVM implementations.

3 Parallelization

SCIDDLE uses a simple declarative language to specify the RPCs exported by a server. In Fig. 1 we present a simplified version for the OPAL server that computes the non-bonded interactions (`nbint`).

Constants may be defined and used as symbolic array dimensions. User-defined types may be constructed from arrays and records. Procedures come in two flavors, synchronous and asynchronous. Each procedure parameter is tagged with a direction attribute. Parameters are always copied in the respective directions.

As SCIDDLE is designed in particular for parallel distributed numerical applications, it provides special array handling support for easy distribution of subarrays to multiple servers. A subarray can be selected by attributing array parameters with views. A view is, like an array section in MATLAB [7], a triple of the form `[begin-index:stride:end-index]` (cf. Fig. 1). The view components are either constants or the names of other integer parameters passed in the same call.

The SCIDDLE stub compiler translates a remote interface definition into the appropriate PVM communication primitives. In the example of Fig. 1 the procedure `nbint` is declared asynchronous. Thus the client stub provides the two subroutines

```
int invoke_nbint (int nval, Matrix atcor, int sid);
int claim_nbint (Matrix atfor, int cid);
```

A call of `invoke_nbint` initiates an asynchronous RPC on the server `sid`. The input parameters are sent to the server, and a call identifier `cid` is returned as the function result. `invoke_nbint` returns as soon as the parameters are safely on their way to the server.

As soon as the client is ready to receive the results from an asynchronous RPC it calls the subroutine `claim_nbint`. The call identifier obtained from `invoke_nbint` is consumed, and the output parameters are retrieved.

If the procedure `nbint` had been declared synchronous the server stub would only provide the subroutine

```
int call_nbint (int nval, Matrix atcor, Matrix atfor, int sid);
```

A call of `call_nbint` blocks the client until the results have been returned from the server. Such a blocking RPC may be useful for quick calculations, but no parallelism can be obtained in this way.

The server process must implement the procedure

```
void nbint (int nval, Matrix atcor, Matrix atfor);
```

The server stub receives the input parameters from the client, calls `nbint`, and sends the results back to the client.

The SCIDDLE runtime system also offers additional subroutines for starting and terminating server processes. Multiple ongoing asynchronous RPCs can be managed conveniently by means of call groups [9].

4 Results

In this section we present the results obtained from a first parallel implementation of OPAL. Our test case consisted of an abbreviated simulation of the complex between the Antennapedia homeodomain from *Drosophila* and DNA immersed in 2714 water molecules [4]. We executed 20 time steps of 2 fs, resulting in a total simulation time of 40 fs.

We conducted our experiments on one and two nodes of the Cray J90 SuperCluster at ETH Zürich. Each node features a shared memory of 2 Gigabytes and 8 vector processors. One Cray CPU delivers a peak performance of 200 Megaflops. The nodes are connected by a fast HIPPI network with a bandwidth of 100 Megabytes/sec.

Table 1 gives an overview of the execution times and speedups obtained from the parallel version of OPAL using various numbers of servers. The experiments with 6 and 8 servers were executed on two Cray J90 nodes, whereas all the other results were obtained on a single node. The column labelled “Execution Time” reports the wall clock time needed for a simulation run, including the

OPAL Version	Number of Servers	Execution Time	Speedup
sequential		432.6 sec	
parallel	1	547.4 sec	0.79
	2	302.9 sec	1.43
	3	241.3 sec	1.79
	4	174.6 sec	2.48
	6	146.8 sec	2.95
	8	144.0 sec	3.00

Table 1: Results on Cray J90 SuperCluster

OPAL Version	Number of Servers	Execution Time	Speedup
sequential		4669 sec	
parallel	1	5171 sec	0.90
	2	2751 sec	1.70
	3	1842 sec	2.53
	4	1413 sec	3.30
	5	1240 sec	3.77
	6	972 sec	4.80
	7	882 sec	5.29

Table 2: Results on Network of Silicon Graphics Workstations

time required for reading input files, pre-, and postprocessing. The speedups were computed as the ratio of the parallel execution time and the sequential execution time.

The experiments on the Cray J90 were performed during the normal operation of the system. Therefore the execution times include overhead due to the operating system and the timesharing environment. If the experiments were run on a dedicated system we would obtain faster and better reproducible execution times.

We also ran the same benchmarks on a set of Silicon Graphics Indy workstations at ETH in Zürich. These workstations are connected by an Ethernet, and they were also used by other jobs during our benchmarks. However we performed our tests during times when the machines were only lightly loaded. Table 2 summarizes the results for this environment.

The sequential version of OPAL performs extremely well on a vector computer like the Cray. The most time-consuming calculations during the evaluation of the non-bonded interactions can be entirely vectorized. This explains the performance advantage of the Cray over the Silicon Graphics workstations.

The slowdown of the parallel version with one server compared to the sequential version can be explained by the overhead of the parallelization. This includes the time necessary to start the server processes as well as the communication time between UNIX processes. Some of this overhead

only occurs during the initialization of the simulation, and thus is less significant for longer runs.

On the Cray the efficiency of the parallel version drops significantly if more than four servers are used. To a large extent this can be attributed to the timesharing operating system which does not offer gang scheduling for a set of PVM processes. But the Cray also loses some efficiency due to shorter vector lengths.

On the other hand the parallel implementation of OPAL scales very well on the network of workstations. This confirms that the computation is the main bottleneck of an OPAL simulation and not the communication between the client and its servers. We could expect an even higher performance if the server program were better tuned to the pipelined arithmetic and to the hierarchical memory structure of modern workstations.

The current version of OPAL uses a static load balancing scheme where each server receives tasks of equal size. This works well if all the processors in the network provide the same computing power. A moreScSc dynamic approach will be necessary if the machines are loaded very unevenly.

References

- [1] P. Arbenz, W. Gander, H. P. Lüthi, and U. von Matt: Sciddle 4.0, or, Remote Procedure Calls in PVM. In *High-Performance Computing and Networking, Proceedings of the International Conference and Exhibition*, ed. H. Liddell, A. Colbrook, B. Hertzberger and P. Sloot, Lecture Notes in Computer Science, Vol. 1067, Springer, Berlin, 1996, pp. 820–825.
- [2] P. Arbenz, H. P. Lüthi, J. E. Mertz, and W. Scott: Applied Distributed Supercomputing in Homogeneous Networks. *International Journal of High Speed Computing*, 4 (1992), pp. 87–108.
- [3] P. Arbenz, H. P. Lüthi, Ch. Sprenger, and S. Vogel: Sciddle: A Tool for Large Scale Distributed Computing. *Concurrency: Practice and Experience*, 7 (1995), pp. 121–146.
- [4] M. Billeter, P. Güntert, P. Luginbühl, and K. Wüthrich: Hydration and DNA recognition by homeodomains, *Cell*, 85 (1996), pp. 1057–1065.
- [5] H. Casanova, J. Dongarra, and W. Jiang: The Performance of PVM on Massively Parallel Processing Systems. Tech. Report CS-95-301, University of Tennessee, Computer Science Department, Knoxville, TN, August 1995, <http://www.cs.utk.edu/~library/TechReports/1995/ut-cs-95-301.ps.Z>.
- [6] P. Luginbühl, P. Güntert, M. Billeter, and K. Wüthrich: The new program OPAL for molecular dynamics simulations and energy refinements of biological macromolecules, *J. Biomol. NMR*, (1996), in press.
- [7] The MathWorks Inc., MATLAB, High-Performance Numeric Computation and Visualization Software, Natick, Massachusetts, 1992.
- [8] H. Poxon and L. Costello: Network PVM Performance. Cray Research Inc., Software Division, Eagan, MN, unpublished manuscript, June 1995.

- [9] Ch. Sprenger: User's Guide to Sciddle Version 3.0, Tech. Report 208, ETH Zürich, Computer Science Department, December 1993,
<ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/2xx/208.ps>.
- [10] W. F. van Gunsteren and A. E. Mark: On the interpretation of biochemical data by molecular dynamics computer simulation. *Eur. J. Biochem.*, 204 (1992), pp. 947–961.