Accurate Performance Evaluation, Modelling and Prediction of a Message Passing Simulation Code based on Middleware

Michela Taufer and Thomas Stricker

Laboratory for Computer Systems
Swiss Institute of Technology (ETH)
CH-8092 Zuerich, Switzerland
taufer@inf.ethz.ch, tomstr@computer.org

July 29, 1998

Abstract

In distributed and vectorized computing there is a large number of highly different supercomputing platforms an application could run on. Therefore most traditional parallel codes are ill equipped to collect data about their resource usage or their behavior at run time and the corresponding data are rarely published and few scientists attack the planning of an application and its platform systematically. As an improvement over the current state of the art, we propose an integrated approach to performance evaluation, modeling and prediction for different platforms. Our approach uses a combination of analytical modeling and systematically designed experimentation with full application runs, reduced application kernels and some benchmarks. We studied our methodology of performance assessment with Opal, an example code in molecular biology, developed at our institution to run on our four Cray J90 "Classic" Vector SMPs. Besides a detailed assessment of performance achieved on the J90s, the primary goal of our study was to find the most suitable and most cost effective hardware platform for the application, in particular to check the suitability of this application for slow CoPs, SMP CoPs and fast CoPs, three flavors of Clusters of PCs built with off-the-shelf Intel Pentium processors. A performance assessment based on our model is much easier than porting and parallelizing the application for a new target machine and so we could easily obtain and include performance estimates for a T3E-900, a high end MPP system. The predicted execution times and speedup figures indicate that a well designed cluster of PCs achieves similar if not better performance than the J90 vector processors currently used and that the computational efficiency compares favorably to the T3E-900 for that particular application code.

1 Introduction

A large variety of different parallel computing platforms makes it quite difficult to pick the the best suited and most cost effective parallel computer as a target platform for a new application code to run on. The question of the best platform for an application should be addressed early in the design process as soon as the characteristics of the application code become known during early prototyping. Typically the scientists in the application field are only interested in the results of the computation itself and rarely in how fast and how efficiently they were computed. Therefore most traditional parallel codes are ill equipped to collect data about their resource usage or their behavior at run time and the corresponding data are rarely published. There are a few exceptions to this rule - like [14], a paper devoted entirely to the characteristics of a large FEM application.
Most machines in high performance computing and even todays PCs have good hardware instrumentation to collect all the necessary data, but most system vendors don't promote direct access to them. Instead they provide high-level performance tuning and advisory tools with little information about their resolution, their accuracy and their theory of operation. Furthermore those tools often interfere with the tasking support of the parallelization and communication tools (i.e. the middleware for parallelization). Common other problems are also the client/server paradigm with full overlap of computation and communication and many latency hiding mechanisms, that make accurate and detailed performance measurements almost impossible.

As an improvement over the current state of the art, we propose and demonstrate an integrated approach to application design, parallelization and performance analysis using a combination of analytical modeling and measurements. We studied this integration with Opal [13], an example code in molecular biology, developed at our institution to run on our four Cray J90 "Classic" Vector SMPs, with 8-16 processors each. Besides a detailed assessment of performance achieved on the J90s, the primary goal of our study was to find the most suitable and most cost effective hardware platform for the application, in particular to check its suitability for fast, slow and SMP CoPs, two flavors of Clusters of Pentium PCs, built by our computer architecture group.
In Chapter 2 we develop and present an analytical complexity model to predict the execution time for Opal simulations with different input parameters. We calibrate the model with a systematic experimental design and show what we learned from detailed analysis of computation and communication performance. In Chapter 3 we discuss the integration of performance monitoring into middleware packages and despite overlapped communication and computation. In Chapter 4 we use the model together with some architectural key data to predict the efficiency of Opal on alternative platforms including Clusters of PCs.

2 Case Study: Instrumenting a Molecular Biology Code

2.1 A brief description of Opal

Opal is a software package to perform the simulation of the molecular dynamics of proteins and nucleic acids in vacuum or in water through energy minimization. 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:

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

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:

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

A typical function V has the form:

$$\begin{aligned} V(\vec{r}_1, \ldots, \vec{r}_n) = & \sum_{allbonds} \frac{1}{2} K_b (b - b_0)^2 + \sum_{allbondangles} \frac{1}{2} K_\Theta (\theta - \theta_0)^2 + \\ & \sum_{improperdihedrals} \frac{1}{2} K_\xi (\xi - \xi_0)^2 + \sum_{dihedrals} K_\varphi \big(1 + \cos(n\varphi - \delta)\big) + \\ & \sum_{allpairs(i,j)} \Big(\frac{C_{12}(i,j)}{r_{ij}^{12}} - \frac{C_6(i,j)}{r_{ij}^{6}} + \frac{q_i q_j}{4\pi\varepsilon_0 \varepsilon_r r_{ij}} \Big). \end{aligned}$$

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}.
A first serial version of Opal, Opal-2.6, was developed at the Institute of Molecular Biology and Biophysics at ETH Zürich [12]. It was written in standard FORTRAN-77 and optimized for vector supercomputers through a few vectorizable loops. In the serial code of Opal-2.6 a single processor runs the whole computation. Opal-2.6 spends most of the computing time during a simulation evaluating the non-bonded interactions over all pairs of atoms of the molecular system (the last term of the atomic interaction function V). Fortunately, these calculations also offer a high degree of parallelism in addition to the vectorizable inner loops.

The parallel version of Opal

The parallel version of Opal [17, 2] distributes its work among multiple processors in a client-server setting: multiple servers share the computation of the Van der Waals and Coulomb forces while one client computes the few remaining interactions and coordinates the work. The computation repeats for every time step.
For a molecular complex [1] of n atoms, the number of non-bonded interactions between atoms, which must be evaluated, is of the order of n^2. In the new version of Opal, this sequential complexity of the molecular energies evaluation is reduced by neglecting many of the non-bonded interactions from the molecular energy computation: only the pairs of atoms, whose distance is less than a *cut-off* parameter, are taken into account. At first, the data describing the non-bonding interaction parameters between the solute-solute, solute-solvent, solvent-solvent atoms pairs are replicated on all the servers. This global information, whose volume depends on the problem size and does not scale with the number of processors, allows each server to achieve a large independence. Through this data, each server runs its tasks into the simulation requesting no more parameters at each step from the client than the atom coordinates.
The simulation consists in repeating the same computation tasks at the end of which the information about the total energy, volume, pressure and temperature of the molecular complex is displayed. In the first stage of each simulation step, which we call *update phase*, each server selects a distinct subset of the atom pairs, checks the distance among the atoms of each pair and adds the pair to its own *list of all active pairs* when the atoms are not beyond the given distance *cut-off*. In the second stage of the simulation step, the servers compute partial non-bonded energies (Van der Waals energy and Coulomb energy) using the *list of all active pairs*. At the end of this step each server sends its partial results to the client which gathers them and sums the total molecular energy of the molecular complex as well as its volume, pressure and temperature.
The data in each list are updated periodically. The interval between successive updates can be selected by the user through the setting of an Opal parameter called *update*. The value of the *update* parameter expresses the number of interaction steps after which the *lists of all active pairs* are updated.
The distribution of the atom pairs for the evaluation of the energies due to the non-bonded interactions is done using a *pseudo-random strategy*. Randomization should help to balance the workload among the servers and to avoid duplication of work.
Moreover, with a slight change of the molecular simulation model, i.e.,the use of water molecules as single units centered in the oxygen atoms in the solvent instead of three individual atoms, we accomplished:

- a reduced workload of the servers,
- a reduction in size of the list (memory usage) and even
- an increase in accuracy for the molecular energy calculations with small cut-off radii.

Parallelization Alternatives

The parallelization of the non-bonded pairwise computation through the distribution of the mass centers amongst several processors used for Opal is not the only parallelization approach. There are three main approaches to a parallelizing: the *replicated-data (RD)* method used for Opal, in which the mass centers (i.e. atoms) are distributed among the processors, the *geometric- or space-decomposition (SD)* method, in which each processor considers the mass centers into its sub-domain during the simulation and the *force- decomposition (FD*) method in which the force matrix $F_{i,j}$ ($F_{i,j}$ is the force on mass center i due to mass center j) is partitioned by blocks among the processors [15, 1].

Comparable packages in molecular biology

With its parallelization the parallel version of Opal has become similar to Amber [19]. Both the codes allow the user to carry out energy minimization and molecular dynamics using the same analytical function (see 2.1). Moreover, both the codes use molecular components interaction lists, which need periodical updates, and allow to evaluate the interactions of each atom with the rest of the molecular complex into a cut-off distance. For the parallel version of Amber, a generalized MPI interface [7] is used for message-passing, while the parallel version of Opal relies on the PVM interface, and the Sciddle RPC middleware package [4, 18]. Still both codes are explicitly parallel and well suitable for distributed memory machines with a Msg Pass API.

[1] A molecular complex is the combination of a solute such as proteins or nucleic acids and a solvent, e.g. water.

2.2 A time complexity model for Opal

During the design and parallelization of Opal we derived an analytical time complexity model that captures all essential parameters of the real application. The predicted outcome of the model is the execution time of Opal in seconds, written as a sum of several partial result variables which are computed separately and also measured separately during validation:

$$t_{OPAL} = t_{tot\_par\_comp} + t_{tot\_seq\_comp} + t_{tot\_comm} + t_{tot\_sync}$$

- $t_{tot\_par\_comp}$, the total parallel computation time, is the computation time spent by the servers servicing the request for the computation. The servers run two routines as parallel work: *the update routine* that updates the lists of atom pairs, and the *energy evaluation routine* that evaluates the partial energies of the non-bonded interactions (Van der Waals energy and Coulomb energy).

  $$t_{tot\_par\_comp} = t_{update} + t_{nbint} \tag{2}$$

  The computation time of the update routine always grows quadratic with problem size because each time the servers update their own list, all the pairs of atoms must be checked. At the same time, the update time decreases linearly with the increase of the time interval between two list updates.

  $$t_{update}(n,\gamma) \approx a_2 \frac{s\,u}{p} \frac{(1-2\,\gamma)^2\,n^2 - (1-2\,\gamma)\,n}{2} \tag{3}$$

  where:

  - s is the number of simulation steps.
  - p is the number of servers on which run the Opal application.
  - u is the frequency of the list updates (update parameter).
  - n represents the number of mass centers (atoms and water molecules) of the whole molecular complex.
  - γ (gamma) is the ratio of number of water molecules to the total number of mass centers.
  - a_2 represents the computation time spent to generate a pair of atoms and calculate the distance between them.

  On the other hand, the time for the energy evaluation routine is subject to the effects of the cut-off parameter: the dimension of the lists, on which pairs the partial energies are evaluated, increases drastically with the increase of the cut-off distance. The energy-evaluation routine grows quadratic up to the number of atoms within the cut-off radius and linear beyond that.

  $$t_{nbint}(n,\tilde{n}) \approx \begin{cases} a_3 \frac{s}{p} \frac{n(n-1)}{2} & \text{when } n < \tilde{n} \\ a_3 \frac{s}{p} \tilde{n}\,n & \text{when } n > \tilde{n} \end{cases} \tag{4}$$

  where:

  - $\tilde{n}$ is a function of the cut-off radius and the volume density of the molecular complex. The meaning of $\tilde{n}$ is is the average number of neighboring atoms considered for their total energy calculation.
  - a_3 is the time needed to compute the non-bonded energy contribution of a single pair of atoms.

  The energy evaluation entirely dominates the parallel computation time when $n < \tilde{n}$:

  $$t_{nbint} >> t_{update}$$

  The introduction of the cut-off parameter reduces the amount of the computation spent into the energy evaluation routine when $n > \tilde{n}$. But despite its lower asymptotic complexity due to the easy introduction of the cut-off parameter, the energy evaluation routine dominates the update process in this second case for all practical problem sizes. The crossover point in n for which the update time equals the energy evaluation time depends on the

molecular complex volume as well as the cut-off parameter. For our simulations (see 2.5) crossover happens for for unrealistic numbers of water molecules or protein atoms to achieve the high values of n. Furthermore a reduction of the update frequency is possible to reduce the amount of update computation arbitrarily and restore the assumed relation:

$$t_{nbint} > t_{update}$$

We summarize the total parallel time as:

$$t_{tot\_par\_comp} \approx \begin{cases} s\left(\frac{1}{2p}\left(a_2\, u\, (1-2\,\gamma)^2 + a_3\right) n^2 - \frac{1}{2p}\left(a_2\, u\, (1-2\,\gamma) + a_3\right) n\right) & \text{when } n < \tilde{n} \\ s\left(\frac{1}{2p}\left(a_2\, u\, (1-2\,\gamma)^2\right) n^2 + \frac{1}{p}\left(a_3\, \tilde{n} - a_2\, u\, \frac{(1-2\,\gamma)}{2}\right) n\right) & \text{when } n > \tilde{n} \end{cases}$$

- $t_{tot\_seq\_comp}$, the total sequential computation time, is the total time spent by the client to compute the energy-, pressure-, volume-, and temperature values of the molecular complex from the partial energies and forces computed in the parallel step. It depends on the number of steps of the simulation and on the number of atoms of the molecular complex:

$$t_{tot\_seq\_comp} = a_4\, s\, n \tag{5}$$

  where a_4 is the time needed for each atom of the molecular complex to evaluate its bonded interactions.

- t_{tot_comm}, the total communication time, is the time spent by the communication processes between the client and the servers during the entire simulation. The client calls two different kinds of routines that are run on the servers: the list update routine and the energy evaluation routine. We enhanced the communication environment with some synchronization tools that allow us to separate the communication times properly from other computation and idle times and therefor permit to explain the communication components precisely. More details about these synchronization tools and their underlying model are explained in [17]. The resulting communication time of the client's RPCs can be decomposed into:

$$t_{tot\_comm} = t_{call\_upd} + t_{return\_upd} + t_{call\_nbi} + t_{return\_nbi} \tag{6}$$

  In addition to a constant overhead, the time spend by the client to send the data (the atom coordinates), which are used by the servers for the list update phase and the energy computation, is linear in the problem size n.

$$t_{call\_upd} = t_{call\_nbi} = \frac{\alpha}{a_1}\, n + b_1 \tag{7}$$

  besides to the quantities defined above we define:

  - α (alpha) is the number of bytes used to represent the coordinates of a single atom.
  - a_1 is the communication rate including the overhead in the communication environment (Sciddle and PVM)
  - b_1 is the communication overhead, in seconds, used to transfer an empty block from the sender to the receiver

  For the update RPC, the client does not retrieve any data from the servers when they arrive at the end of the update routine: the client just waits for a result message which assures the end of the server tasks.

$$t_{return\_upd} = b_1 \tag{8}$$

  On the other hand, the amount of data, sent by each server to the client from the energy evaluation routine when it exits, comprises the Van der Waals and Coulomb energies, and the gradients of the atomic interaction potential.

$$t_{return\_nbi} = 2\,\frac{\alpha}{a_1} + \frac{\alpha}{a_1}\, n + b_1 \approx \frac{\alpha}{a_1}\, n + b_1 \tag{9}$$

  All together, the total communication time of equation (6), can be rewritten as:

$$t_{tot\_comm} = s\left(p\,\frac{\alpha}{a_1}\,(u+2)n + 2\,p\,b_1\,(u+1)\right)$$

- t_{tot_sync}, total synchronization time, is the time to synchronize the processes with each other. t_{tot_sync} is the sum of four terms:
  - t_{str_upd}, the total time to synchronize the client and the servers when the update routines are called,
  - t_{end_upd}, the total time to synchronize the client and the servers when the update routines finish,
  - t_{str_nbi}, the total time to synchronize the client and the servers when the energy evaluation routines are called,
  - t_{end_nbi}, the total time to synchronize the client and the servers when the energy evaluation routines finish.

  We assume that the four synchronization time components do not depend on the number of servers as well as on the problem size while we define that each term increases linearly in the number of simulation steps and moreover, t_{str_upd} and t_{end_upd} decrease as the update parameter decreases. We assume that each synchronization process takes a constant time b_5.

$$t_{tot\_sync} = s\,u\,b_5 + s\,u\,b_5 + s\,b_5 + s\,b_5 = 2\,s\,(u+1)\,b_5 \tag{10}$$

  Due to available space, we do not introduce more details about the synchronization tools and their development here but we refer the reader to [17].

Model parameters

To conclude our explanation of the model we summarize all the used parameters and categorize them into application parameters and platform parameters.
The *application parameters* are: the number of simulation steps s, the number of servers on which the Opal application p runs, the frequency of the list updates u, the number of mass centers (atoms and water molecules) of the whole molecular complex n, the average number of neighboring atoms considered for their total energy calculation $\tilde{n}$ which is a function of the cut-off radius c and the volume density of the molecule and last but not least the the ratio of number of water molecules to the total number of mass centers γ. The parameters relevant to the platforms (i.e. the parallel machines) are:

- the communication rate a_1 measured in MByte/sec,
- the communication overhead b_1 measured in seconds,
- the computation rate in MFlop/sec which is indirectly obtained by a weighted sum of:
  - the computation time spent to generate a pair of atoms and calculate the distance between them, a_2, in seconds,
  - the time spent to compute the non-bonded energy contribution of a single pair of atoms, a_3, in seconds,
  - the time needed to each atom of the molecular complex to evaluate its bonded interactions, a_4, in seconds,

  over the number of floating point operations counted by our hardware monitors,
- the time to synchronize processes b_5 in seconds.

2.3 Determining the application parameters

A series of measurements is used to determine all model parameters for the reference platform (the Cray J90). The experimentation with the code follows a systematic, full factorial experimental design according to Chapter 16 in [11]. The performance of the code depends mainly on four factors whose effects have been isolated to obtain the maximum information with the minimum number of experiments:

- the number of servers (parallelism),
- the number of atoms in the molecular complex (problem size),

- the cut-off parameter (approximation properties),
- the update frequency parameter (communication-computation balance).

For the calculation we consider one to seven servers, small, medium and large problems, full- vs. partial- updates and two different cut-off radii - a small, effective one at 10 Å vs. a large, ineffective one at 60 Å. The response variables measured are the communication, parallel computation, sequential computation, idle time and synchronization time as listed in the composite formula. The experiments always run on a dedicated system and therefore there is no overhead on the measurements due to a timesharing environment. In a few preliminary tests, every measurement has been repeated several times. The tests have confirmed a low variability and a good reproducibility of the execution times, and we therefore have concluded that ten simulation steps suffice to assure an accurate and meaningful timing of an entire simulation of the protein folding process, which we call an experiment or case.

2.4 Understanding the Execution of Opal with our model

We investigate the performance of the Opal code by measurements of simulation execution times of two molecular complexes with different sizes: the parallel computation time, the sequential computation time, the communication time, the synchronization time, and the idle time. We measure the detailed breakdown of the wall clock execution time for ten simulation steps.
The first molecular complex is a medium size problem for a simulation that Opal can handle: it is the complex between the Antennapedia homeodomain from Drosophila and DNA [8], composed by 1575 atoms and immersed in 2714 water molecules or a total of 4289 mass centers (*medium problem size*). The second molecular complex has a large size: it is the NMR structure of the LFB homeodomain, composed by 1655 atoms and immersed in 4634 water molecules, a total of 6289 mass centers (*large problem size*).
We run the code for different levels of parallelism: the number of servers ranges from one to seven. At the same time we measure the execution times when the simulation is fully accurate and the computation complexity is quadratic in the protein size (i.e. *no cut-off*) and when the simulation is approximate and consequentially the computation complexity becomes linear (i.e. *with cut-off*). Finally, we investigate the role of the lists update: we run the simulation ether with an update of our lists upon every iteration (*full update*) or with a partial update every 10 iterations (*partial update*). The comparison of the different cases allows the study of the scalability, i.e. execution time, with increasing parallelism and to investigate the impact of the problem size, the frequency of the lists update, the cut-off parameter on the performance.
Figures 1a)-d) display a detailed breakdown of the wall clock execution time for 10 simulation steps in the medium size molecular complex with different choices for the number of servers, the cut-off and the update parameters. The chart in Figure 1a) shows that without cut-off, the time in parallel computation is the largest fraction of the execution time and that it decreases as expected when more servers are added. At the same time the communication time increases about linear with the number of servers, but its overall contribution remains small, even for seven servers. The synchronization time and the sequential computation time remain insignificant to the overall execution time. However to the surprise of the Opal implementors, our instrumentation reveals a load balancing problem for runs with an even numbers of processors. Figure 1b) shows an Opal execution with reduced updates. As expected, the lower update frequency does not affect the overall performance on simulations much because the large amount of parallel computation dominates the execution time. Figure 1c) shows a simulation with an effective cut-off parameter (at 10 Å). The cut-off parameter determines the asymptotic computational complexity: the amount of the parallel computation is smaller than in the cases above and its overall contribution becomes comparable to the other measured execution times. The sequential computation time, the synchronization time and the communication time gain a high importance for the overall performance. Figure 1d) displays a run with both the cut-off and the partial update option in effect. The frequency of the list updates leads to a notable difference in the performance of simulations with small cut-off radii.
The problem size itself, the number of atoms of the whole molecular complex, has a varying impact on the different components of the execution time. The time components, the parallel computation time, the communication time, the idle time, the sequential computation time and the synchronization time, increase each one in a different way with the number of atoms: while the size of the problem has a super-linear impact on the parallel computation time and the idle time, it has only a linear moderate impact on the sequential computation time and the communication time. Figure 2a)-d) show the detailed breakdown of the wall clock execution time for 10 simulation steps in the large size molecular complex. While the order of the measured execution time doubles when we increase the problem size from

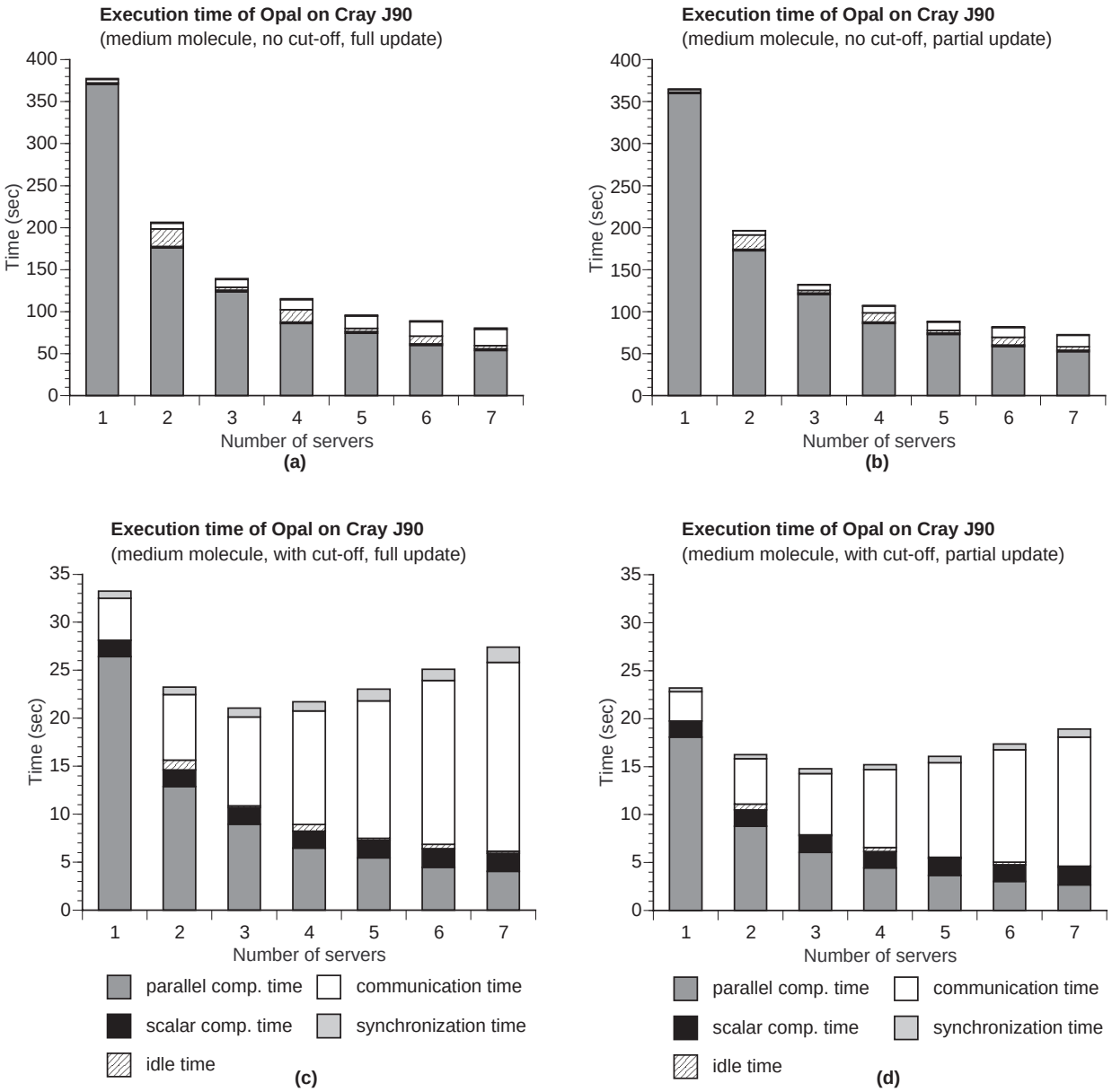

Figure 1: Detailed breakdown of the measured execution times for 10 iterations of an Opal simulation with a medium molecule.

a medium amount of mass centers to a large amount of mass centers, the behavior of the computation time components remains almost the same.

2.5 Calibrating Model with our Implementation

In order to verify the analytical model against the implementation, we compare the measured execution time on the Cray J90 with the computed values of the model for the same machine.

Figure 3 displays the parameter space that we have considered during calibration of our model. We have chosen three different molecular complexes, each one with a different problem size: small, medium, large size. Furthermore we have run the simulation of each molecular complex picking the update frequency and the cut-off parameter at the two extreme values: a lists update each ten simulation steps (partial update) versus a lists update at each simulation

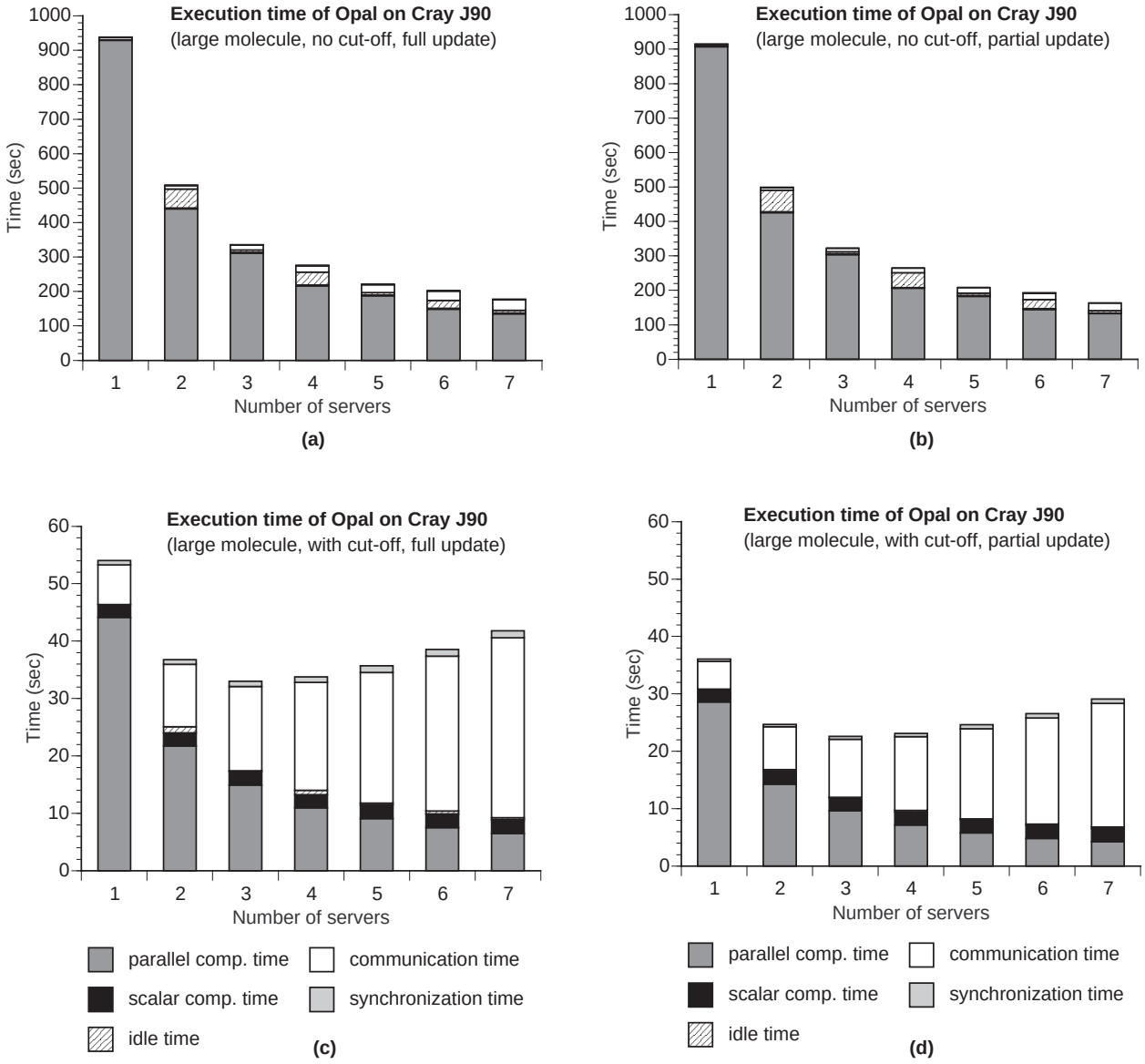

Figure 2: Detailed breakdown of the measured execution times for 10 iterations of an Opal simulation with a large molecule

step (full update) and a simulation without cut-off parameter (i.e. all the atoms interactions are considered) versus a simulation with cut-off parameter (i.e. for each atom only interactions within the range of 10 Å are considered). At the same time we have computed the outcome of the simulation through the analytical model for each one of these cases and adjusted the parameters for a last square fit to the corresponding measurements.
Figures 4a)-d) show the comparison of the wall clock times measured against the times predicted by the analytical model for a different number of servers, different cut-off, update frequency and large or medium molecular complex. For brevity in the paper, we list only the data of a reduced $(7 \cdot 2^{3-1})$-design although the data was achieved with a full factorial design of 84 experiments. During the calibration the differences between model and measurement have been investigated and plotted for each case. The overall fit of the model to the measurement for the cases in Figures 4a)-d) and for the remaining cases is excellent. The full data is listed in [17].

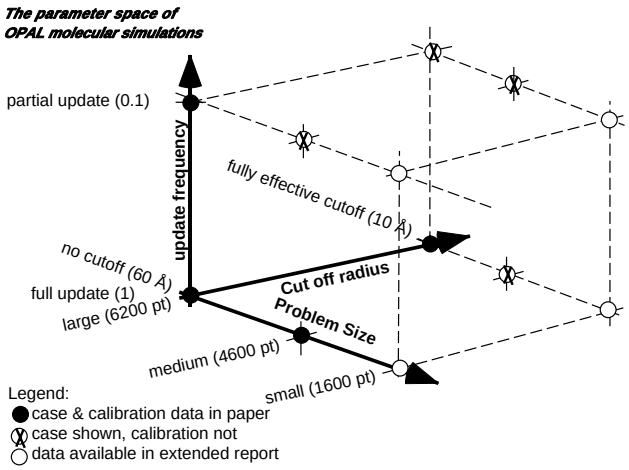

Figure 3: Parameter space model of Opal molecular simulation.

2.6 Other Complexity Measures

Execution time is not the only complexity measure that can be treated in this manner. A space complexity model for memory issue is largely orthogonal to the execution time model.
Memory management remains a highly complex issue in the parallelization. The parallel Opal has been designed to use memory in the most economical way: each server has only a part of the non-bonded interactions pairs of atoms. The dimension of the data lists on each server scales down linearly with the number of processors.
On the other hand, each server needs the same global data (information about the solution-solution, solution-solvent, and solvent-solvent non-bonded interactions) whose amount of data depends on the problem size, i.e, the number of water and molecular atoms. This global information does not scale with the number of processors. The computation of these global data structures on each server involves a duplication of work but saves some communication. Furthermore, this computation of these constants takes place at the beginning of the simulation, and its effort is amortized throughout the steps of the simulation phase. After having initialized the global data, each server assumes a larger independence during the computation because it evaluates its partial non-bonded interaction terms without requesting further parameters from the client with the exception of the atom coordinates.
The size of the data structures in Opal grows as follows with the problem size:

	Order	Constant	Large Example 6290 mass centers
		c [Bytes]	[Bytes]
pair list	$c(1-2\gamma)n^2$	2*4	160'000'000
atom coordinates	cn^2	3*8	1'000'000
atom gradients	cn^2	3*8	1'000'000
atom interactions	$c(1-3\gamma)n^2$	2*8	3'000'000
energy values	c	2*8	16

A more accurate space complexity model would only be useful when there are interesting tradeoffs between space and time complexity for the predicted execution time. We found no interesting time-space tradeoffs, except for the size of working-sets that influence execution speed through the memory hierarchy, virtual memory with swapping, physical core memory (DRAM) and two levels of caches.
We ran some tests with the single processor version of Opal on our Pentium PC platforms to investigate the computational performance of the most significant loop (*comp_nbint*) in main memory, out-of-core (with unix system swap) and all in cache. On the Cray J90 systems a similar study could be made by turning vectorization off and on. However unlike in the PC world where different cache configurations are on the market, vectorization is no real system design option, since every J90 CPU can vectorize. It would be stupid to turn it off.
The absolute and relative computational performances are as follows:

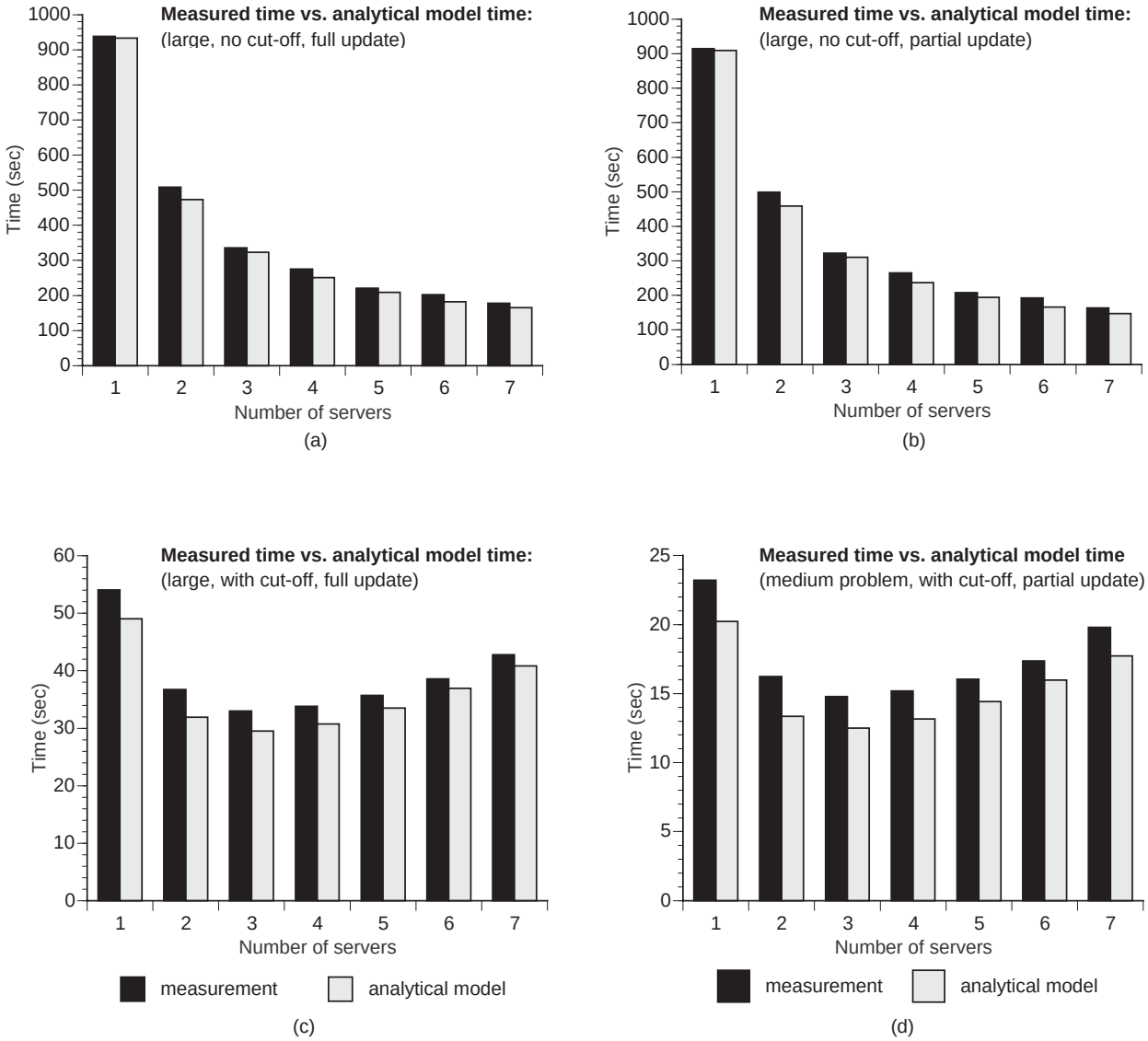

Figure 4: The difference between the wall clock times measured and the times predicted by the analytical model.

	Working Set MByte	Computational Rate on Pentium200 [MFlop/s] [2]	Relative
in cache [3]	50K	35	1.09
in core	8M	32	1.00
out of core	120M	8	0.25

After these first trials with the memory hierarchy, it appears to us that blocking Opal for the caches is not beneficial for enhancing the performance. It appears that the inner loop of Opal is CPU and not memory limited. This observation is also confirmed by the good speedup of SMP nodes (see upcoming sections). The performance breakdown for the out of core case is so drastic, that out-of-core problem sizes would push the execution time beyond the limit for acceptable turnaround for one simulation.

On most systems we could use the hardware performance monitoring hardware to account more accurately for cache misses and strongly confirm the validity of our experiments above. Again this underlines the need to instrument mid-

dleware for performance monitoring right from the start when a parallel application code is designed and implemented. Although the authors of this paper can think of further code blocking transformation that would enhance locality for better use of the caches, we stopped our investigation after these trials. We are currently lacking the control over the Opal production code distribution and therefore we have no way to dispatch our improvements into the real world.

3 Integrating Performance Instrumentation with Application Design

3.1 Overheads and loss of transparency due to middleware

The client server structure of parallel Opal is ideally suited for Sciddle [4], a remote procedure call (RPC) system extension to the PVM communication library. Sciddle comprises a stub generator (the Sciddle compiler) and a run-time library. The stub generator reads the remote interface specification, i.e., the description of the subroutines exported by the servers, and generates the corresponding communication stubs. The stubs take care of translating an RPC into the necessary PVM message passing primitives. The application does not need to use PVM directly during an RPC: the client simply calls a subroutine (provided by the client stub), and the Sciddle run-time system invokes the corresponding server subroutine (via the server stub). It was, however, a deliberate decision in Sciddle to use PVM directly for process management (starting and terminating of servers). Thus a Sciddle application still needs to use a few PVM calls at the beginning and the end of a run.

Why to use middleware like Sciddle

Sciddle is a highly portable communication library. It has been ported to Linux PCs, UNIX workstations, the Intel Paragon, and supercomputers like the Cray J90 and the NEC SX-4. In particular, Sciddle supports both PVM systems on the Cray J90, the network PVM and the shared memory PVM. The Sciddle/PVM combination might sound like a very suboptimal solution for a single J90 Classic system, that also supports coherent shared memory. However at the time Opal was in development, our site was operating four Cray J90s interconnected by HIPPI and the developers had certainly plans to use Parallel Opal on a Cluster of J90 SMPs. For such a platform, message passing is a must and shared memory would not do. For most application codes, the additional overhead of Sciddle is very small [3], but Sciddle causes a lack of control over the PVM option flags and PVM internal operations (i.e. the proper use of data in place, shared memory flags). With a specific synthetic RPC test, Sciddle runs communication at about 7 MByte/s which is just about as much as the Sciddle developers got out of a PVM ping pong on the J90 [3]. Therefore we attribute the disastrously low communication performance for the J90 (an SMP machine with a fast crossbar) to the unpredictable performance of the Cray PVM implementation and the unfortunate interaction between middleware and communication library.
We understand that due to its internal architecture and its API PVM is far away from a zero-copy message passing system. Therefore we suggested to the developers that Opal is rewritten in a clean “post-in-advance” style of MPI programming.

3.2 A plea for including of Hardware Performance Instrumentation into Middleware

The tasking facilities of PVM and Sciddle interfere with the normal use of performance monitoring tools such as the HPM command on the Cray J90 systems or the corresponding tools on the T3E or Intel platform. We worked with the implementors of Sciddle to integrate queries to the low overhead counter device (e.g. /dev/hpm) into the Sciddle code and undertake the necessary accounting for the number of floating point operations executed and for the clock cycles used in the client and in the servers. In a setting with a high level abstraction RPC model the performance instrumentation must adhere to the same high level abstractions and therefore be integrated into the middleware as well as the application code. The question of a good API standard for performance monitoring instrumentation is still an open one. Debuggers pose a very similar software engineering problem for the parallel programming world.
Sampling based tools give a direct estimate for the compute rate in MFlop/s and are easy to use, but they are extremely complex to understand. Sampled computation rates are no substitute for the simple ratio of operations counted divided by the cycles used. The characteristic performance of different machine for Opal runs in Table 1 in Section 4 shows how difficult it is to measure MFlop counts. The number of floating point operations required to compute exactly the same application results differs significantly, because of vectorizing transformations and the different implementations

for intrinsic functions like sqrt() and exponentiate(). With a standalong performance monitoring tool we had possibly never learned that fact, but wondered forever about some MFlop/s numbers that could simply not make sense.

3.3 Overlap of Communication and Computation

There is no doubt that the Sciddle package accelerated the development of the application and that its additional overhead stays well within acceptable limits compared to the PVM message passing system. However packages like Sciddle support and encourage the overlap of computation and communication preventing a detailed quantification and correct accounting of the elapsed time for local computation, communication and idle waits due to load imbalance.
In the parallel programming framework Sciddle was conceived for, it might be easy to measure and accumulate high level metrics like *server computation rate*, *client computation rate* for the entire application program, but low level indicators like *communication efficiency*, *idle times*, and *load imbalance* of single parts are much harder to get. The latter metrics are more relevant in the performance analysis. In order to find a solution to the difficulties to measure and quantify overheads in Sciddle, we propose a modification to the timing synchronization behavior to fix this problem. The Sciddle environment does not provide explicit synchronization tools, but it allows a direct communication with the underlying PVM environment and therefore permits explicit synchronization. We introduce additional PVM barriers to separate the communication clearly from the computation: with these changes to the Sciddle communication environment, it is possible to measure or compute all these metrics directly. The barrier function lets the servers synchronize themselves explicitly with each other.
Many papers have been written to show how to eliminate barriers and permit more overlap of communication and computation, but the potential benefit of overlap is often overestimated because of memory system bottlenecks in most machines. For the optimal accounting of times among the client and the servers we are forced to give up some of the overlap. To us, the accuracy, predictability and tight control of performance appears more important and we happily accept a small slowdown (less than 5%) over the overlapped application for the sake of a solid understanding what is going on with performance in the code.
The use of a shared communication channel between servers and client introduces contention due to limited resource at the end of a client compute phase. This contention is application specific and it is likely on the original Sciddle run when all the servers perform the same amount of work (without large server idle time values). The barriers in the modified Sciddle framework do not actually cause, but merely expose the contention of single client multiple server communication in all cases.

4 Performance Prediction for Alternative Platforms

In this last part of our paper, we use our analytic model together with some standard performance data of alternative computer platforms to predict the performance of Opal in the case that we could port the code to that platform. Two different classes of MPP (Massively Parallel Multi-Processors) are considered for our study in addition to the real Opal platform, the Cray J90: the Cray T3E "big iron" MPP and three flavors of PC Clusters: slow CoPs (Cluster of PCs), SMP CoPs and fast CoPs. We call the first PCs Cluster *slow* since it is optimized for lowest cost; connected with shared 100BaseT Ethernet its processors are just single 200MHz Intel Pentium Pro. The SMP CoPs platform is based on some twin 200MHz Intel Pentium Pro processors and SCI shared memory interconnects, while the *fast* PCs Cluster has single 400MHz Intel Pentium Pro processor per node and features fast communication with fully switched Gigabit/s Myrinet Interconnects. Comparable Clusters of PCs installations are described in [5, 6, 16].

4.1 Extraction of Model Parameters for Alternatives

As shown in Section 2.2, the parameters of our analytic model have been intentionally chosen in a way to include all major technical data usually published for parallel machines. This includes among others: message overhead, message throughput, computation rate for SAXPY inner loops and time to synchronize all participating processors. For each new platform we determine the key parameters by the execution of a few microbenchmarks, verified against published performance figures [9]. An overview of the data used is found in the Tables 1 and 2.
We use these figures together with the formulas of the analytical model to predict the execution time of Opal with a medium and large size molecular complex in varying configurations. Some model parameters, which we have called application parameter in Section 2.2, are intrinsic to Opal itself and invariant across the different machines. Those

MPP Node Type (clock speed)	Execution Time on single node [s]	Floating Point op. counted on single node [MFlOp]	Computation Rate on single node [MFlOp/s]	Relative Time [%]	Adjusted Computation Rate on single node [MFlOp/s]
Cray T3E-900 (450 MHz)	9.56	811.71	85	138	52
Cray J90 Classic (100 MHz)	6.18	497.55	80 [a]	100	80
Slow CoPs (200 MHz)	10.00	327.40	32	65	50
SMP CoPs (2*200 MHz)	5.00	327.40	65	65	100
Fast CoPs (400 MHz)	4.85	325.80	67	65	102

Table 1: Computation speed parameters used for performance prediction of Opal on all four different platforms. The numbers reflect the performance of the isolated Opal application kernel used as a microbenchmark.

[a] For fall 98 our J90 Classics are scheduled for an upgrade to the new vector processors that significantly enhance the computational throughput

MPP Node Type	MByte/s on single node (hw peak)	MByte/s on single node (observed)	Latency on single node (observed)
Cray T3E-900 (MPI)	350	100	12 μsec
Cray J90 (PVM/Sciddle)	2000/8	3	10 msec
Slow CoPs (Ethernet)	10	3	10 msec
SMP CoPs	50	15	25 μsec
Fast CoPs (Myrinet)	125	30	15 μsec

Table 2: Communication speed parameters used for performance prediction of Opal for all four different platforms. The numbers are obtained from microbenchmarks and verified against published values

parameters are simply kept at their level measured with the J90. On the other hand, we have had to select the most important platform parameters, which depend on the new machines features. The observed MBytes/s in Table 2 are relevant to the model parameters a_1 and b_1. The model parameters a_2 and a_3 are indirectly achieved using the data in Table 1 and computing the average time used to generate a pair of atoms and calculate the distance between them as well as to compute the non-bonded energy contribution of a single pair of atoms.

Since the model captures the parallelization and adjusts to different processor performance, we can calculate the estimated execution time and the relative speed-up achieved on each new platform and compare it with the measured speed-up achieved on a Cray J90.

As for many scientific codes one routine of Opal dominates the compute performance. This routine has been benchmarked on each platform using the most accurate cycle counters and floating point performance monitoring hardware that is actually present on all four machine types. The most important surprise has been a significant difference in floating point operations for the different platforms although the arithmetic was 64bit in all cases and the results were precisely identical (or within the floating point epsilon for comparisons between Cray and IEEE arithmetic). The differences are due to the different compilers or intrinsics functions. We eliminate this difference by assuming that the best compiler (i.e. the PGI compiler for the PCs [10]) is setting a lower bound for the computation: we adjust the local computation rate (MFlop/s) of other platforms accordingly.

The communication performance is even more difficult to compare in real applications. Some unfortunate interactions

between middleware and PVM library reduces the measured communication rate on the Cray J90 processor to about 3 MByte/s, despite its more than one GByte/s strong crossbar interconnect between the 8 processor boards and the memory banks. The authors of the Sciddle middleware claim that they measured up to 7 MByte/s for a synthetic Sciddle RPC example and that this matches the performance of PVM 3.0 on the machine [3]. It certainly remains below what this machine is capable of in shared memory mode.
We suspect that with the right configuration of PVM flags or at least with a rewrite of the middleware to use MPI in true zero copy mode, we could significantly improve the performance of Opal on the J90, but such work is outside the scope of this performance study. For the other platforms we assumed an MPI or PVM based re-implementation without Sciddle and deduced our performance numbers mainly from MPI microbenchmarks gathered by our students and from similar numbers published by independent researchers on the Internet (e.g. [9]).

4.2 Discussion

The complexity model incorporates the key technical data of most parallel machines as parameters. Therefore it is well suited for performance prediction.
In the first two graphs of Figures 5a)-d) we look at the predicted execution times for 10 Opal iterations with a medium size molecule. Platforms include the Cray T3E, the Cray J90 (reference) and three Clusters of PCs (fast, SMP and slow CoPs). Since we also list the absolute execution time in seconds we can directly compare the performance of all four platforms when 1-7 processors are used in Charts 5a) and 5c). The success or failure of the parallelization for this becomes most evident, when we plot a relative speed-up with 1-7 processors in the Charts 5b) and 5d). The well specified synchronization model guarantees that we are not subject to the pitfalls of a badly chosen uni-processor implementation.
In the upper Charts 5a) and 5b) no cut-off is effective and therefore the runs are largely compute bound. The execution time reflects the different compute performance of the different node processors with a slight edge for the SMP CoPs architecture which becomes slighter and slighter with the increase of the processors number. The compute bound operation also leads to a good speedup, as seen in Chart 5b).
The main users of Opal in molecular biology assured to us repeatedly, that for certain problems a simulation with a 10 Å cut-off parameter is accurate enough to give new insights into the proteins studied. Therefore we ran the second test case of the same molecule with a computation reduced by cut-off. In the lower two Charts 5c) and 5d) the computation is accelerated with the effective cut-off and therefore gradually becomes communication bound as the parallelism increases. In this case the communication performance of the machine does matter a lot. The Cray J90 and the slow CoPs (Ethernet) Cluster of PCs are severely limited by their slow communication hardware or by their bad software infrastructure for message passing. This is visible in predicted execution times: as the number of processors increases and becomes larger than three, the execution time of the Cray J90 and the slow CoPs (Ethernet) Cluster of PCs is increasing rather than decreasing. This increase of the execution time offsets the gain of the parallel execution and loads to an overall loss for the larger number of nodes. This aspect is displayed by some speed-up curves in Charts 5d) which actually turn into slow down curves when too many nodes are added. For these architectures we achieve no benefit in putting more than three processors at work.
The SMP CoPs and Fast CoPs architectures have already lower execution time than the big irons for a small number of processors. However with the increase of the processors number the speed of the Cray T3E catches up due to the good communication time. This trend is also evident in the speed-up curves where the Cray T3E architecture achieves better gain and speed-up. For the platforms with the better communication systems we can scale the application nicely to 7 processor with a speed-up of 4 or greater. As we can see in the lower Graphs 5c) and d), speed-up can not be interpreted without looking at the absolute execution times simultaneously; while the Cray T3E has by few the best speed-up, it still ends behind Fast and SMP CoPs for seven servers.
The same performance scalability relationships are reflected in the Figures 6a)-d) for a large size problem. The charts show predicted execution times and speed-ups for a large problem. A comparison between the Charts 6a)-d) and 5a)-d) shows how the behavior of the execution time remains quite similar to the medium size problem. At the same time we notice that the increase of the amount of the computation for a large size problem lead to slightly better speed-ups in Chart 6b). Still both charts indicate flat speed-up for more processors due to overhead in the communication systems. In Chart 6d) we do not have the extreme slow down seen in in Chart 5d), but we can conclude that the increase of the amount of the computation has just pushed the point of the break down further outwards on the curve. With a larger number of processors we would probably encounter the same saturation point at which adding processors would stop

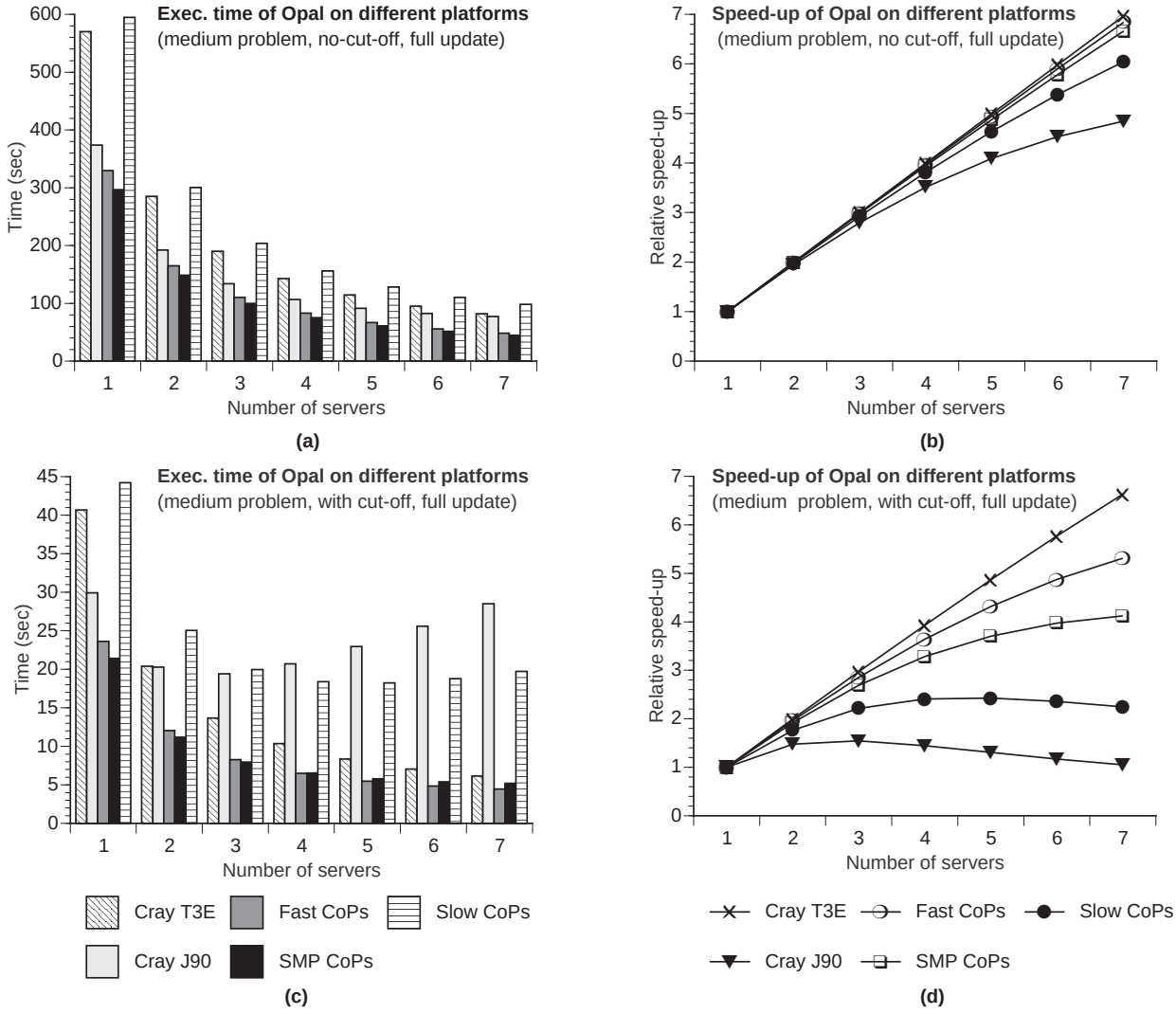

Figure 5: Predicted execution time for an Opal simulation of a medium problem size molecule.

to increase performance.

5 Conclusion

Our case study of Opal showed common problems with the performance instrumentation in an application setting with RPC middleware for parallelization and PVM communication libraries. Some middleware had to be instrumented with hooks for performance monitoring and the overlap of communication and computation had to be restricted slightly for a reliable accounting of execution times. We can state three potential benefits of the integrated approach for accurate performance evaluation, modeling and prediction in parallel programming: firstly, the analytic complexity model and a careful instrumentation for performance monitoring leads to a much better understanding of the resource demands of a parallel application. We realize that the basic application without cut-off is entirely compute bound and therefore parallelizes well regardless of the system. The optimization with an approximation algorithm using an effective cut-off radius changes the characteristics of the code into a communication critical application that requires a strong memory and communication system for good parallelization. Secondly, we discovered interesting anomalies in the implementation, e.g. the load imbalance for even number of servers and the differing number of floating point operations for different processors. Thirdly, we can use our model to predict with good certainty how the application

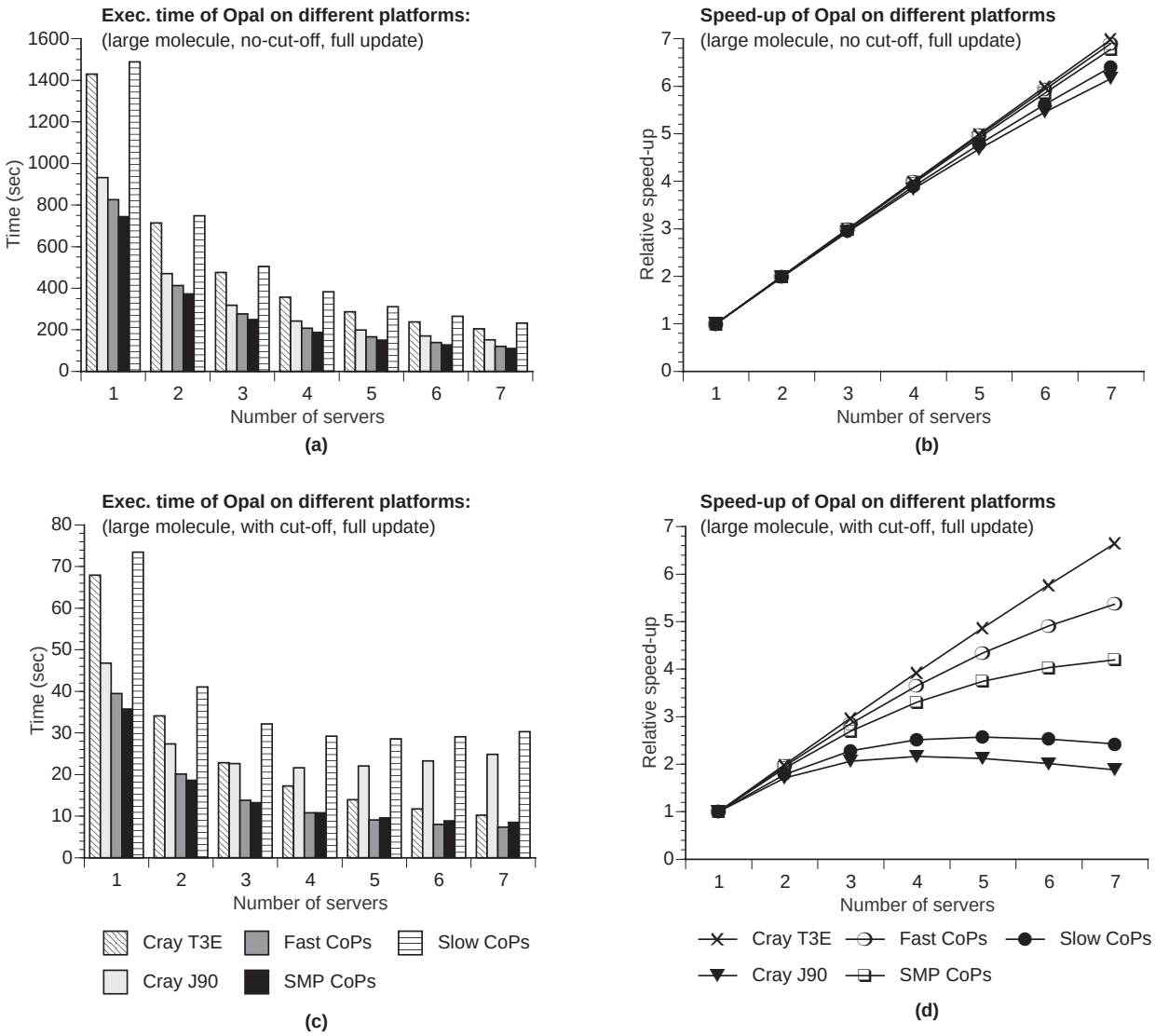

Figure 6: Predicted execution time for an Opal simulation of a large problem size molecule.

would run on *slow Cops*, *SMP CoPs* and *fast CoPs*, three low cost Cluster of PCs platforms connected by Gigabit Networks, like SCI or Myrinet. The migration of the Opal simulation code to the cluster of PC platform could potentially free our upgraded Cray J90 SMP vector machines for more complex and memory intensive computations with less regularity. The predicted execution times and speedup figures indicate that a well designed cluster of PCs achieves similar, if not better performance than the J90 Classic vector processors currently used for Opal and that the computational efficiency compares favorably even to the T3E-900 for this particular application code.

Acknowledgments

We would like to express our thanks to all the people who helped us during this work. We are very grateful to Peter Arbenz, Walter Gander, Hans Peter Lüthi, and Urs von Matt, who created Sciddle to parallelize Opal, for their help and particularly Peter Arbenz and Urs von Matt for reading carefully through several drafts of our work. We sincerely thank Martin Billeter, Peter Güntert, Peter Luginbühl and Kurt Wüthrich who created Opal and particularly Peter Güntert for his help and his chemistry advice. We thank Carol Beaty of the SGI/CRI and Bruno Löpfe of the ETH

Rechenzentrum who helped with questions on the Cray J90. We are also very grateful to Nick Nystrom and Sergiu Sanielevici of the Pittsburgh Supercomputer Center who sponsored our parameter extraction runs for the performance prediction of the Cray T3E-900.

References

[1] P. M. Alsing. N-body problem: Force decomposition method. 1995. http://www.phys.unm.edu/ phys500/-lecture4/force_decomp/force_decomp_background.html.

[2] P. Arbenz, M. Billeter, P. Güntert, P. Luginbühl, M. Taufer, and U. von Matt. Molecular dynamics simulations on cray clusters using the Sciddle-pvm environment. In *Proceedings of EuroPVM 96*, pages 142–149, Berlin, 1996. Springer.

[3] P. Arbenz, W. Gander, H. P. Lüthi, and U. von Matt. Sciddle 4.0: Remote procedure calls in PVM. In *Proceedings of HPCN 96, High Performance Computing and Networiing Conference*, pages 820–825, Berlin, 1996. Springer.

[4] P. Arbenz, C. Sprenger, H. P. Lüthi, and S. Vogel. Sciddle: A tool for large scale distributed computing. *Concurrency: Practice and Experience*, 7:121–146, 1995.

[5] D. J. Becker, D. Sterling, T.and Savarese, J. E. Dorband, U. A. Ranawake, and C. V. Packer. Beowulf: a parallel workstation for scientific computation. In *Proceedings of 1995 ICPP Workshop on Challenges for Parallel Processing*, Oconomowc, Wisconsin, U.S.A., August 1995. CRC Press.

[6] M. Blumrich, R. D. Alpert, Y. Chen, D. Clark, S. Damianakis, C. Dubnicki, E. W. Felten, L. Iftode, M. Martonosi, and R. A. . Shillener. Design choices in the SHRIMP system: An empirical study. In *Proc. 25th Intl. Symp. on Computer Architecture*, pages 330–341. ACM, June 1998.

[7] D. A. Case, D. A. Pearlman, J. W. Caldwell, T. E. Cheatham III, W. S. Ross, C. Simmerling, T. Darden, K. M. Merz, R. V. Stanton, A. Cheng, J. H. Vincent, M. Crowley, D. M. Ferguson, R. Radmer, G. L. Seibel, U. C. Singh, P. Weiner, and P. A. Kollman. *Amber*. Section of the manual still under development.

[8] W. J. Gehrin, Y. Q. Quian, M. Billeter, K. Furukubo-Tokunaga, A. F. Schier, D. Resendez-Perez, M. Affolter, G. Otting, and K. Wthrich. Hydration and dna recognition. *Cell*, pages 211–223, 1994.

[9] A. Geist. Pvm on pentium clusters communication spanning nt and unix. http:www.scl.ameslab.gov/workshops/-Talks/Geist/sld001.htm.

[10] Portland Group. *PGI Workstation User's Guide*. http://www.pgroup.com/ppro_docs, /pgiws_ug/pgiwsu.htm.

[11] R. Jain. *The art of computer systems performance analysis: techniques for experimental design, measurment, simualtion, and modeling*. John Wiley and Sons, Inc., New York, U.S.A, 1991.

[12] P. Luginbühl, P. Güntert, and M. Billeter. *OPAL: User's Manual Version 2.2*. ETH Zürich,Institut für Molekularbiologie and Biophysik, Zrich, Switzerland, 1995.

[13] 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. ETH-BIB P 820 203.

[14] D. O'Hallaron, J. Shewchuk, and T. Gross. Architectural implications of a family of irregular applications. In *Proc. 4nd Symp. on High Performance Computer Architecture*, pages ?–?, Las Vegas, Jan 1998. IEEE. Extended version appeared as Technical Report CMU-CS-97-198, Carnegie Mellon School of Computer Science.

[15] Plimpton S. and Hendrickson B. A new parallel method for molecular dynamics simulation of macromolecular systems. *Sandia Thechnical Report*, SAN94-1862, 1994.

[16] Sobalvarro, Pakin, Chien, and Weihl. Dynamic coscheduling on workstation clusters. Proceedings of the International Parallel Processing Symposium (IPPS '98), March 30-April 3 1998.

[17] M. Taufer. Parallelization of the software package opal for the simulation of molecula dynamics. Technical report, Swiss Federal Institute of Technology, Zurich, 1996.

[18] U. von Matt. Sciddle 4.0: User's guide. Technical report, Swiss Center for Scientific Computing, Zurich, 1996.

[19] P. K. Weiner and P A. Kollman. Amber: Assisted model building with energy refinement. a general program for modeling molecules and their interactions. *J. Comp. Chem.*, (2), 1981.