

software controlled paging rather than dynamic caches [37, 38].

2.4. Statically Ordered Data Structure Access

The lack of side effects in dataflow actors makes it particularly difficult to support large, shared data structures [20]. Arvind *et al.* have therefore extended the dataflow model by introducing *I-structures*, a controlled form of global data structures [2]. *I-structures* are write-once data structures with nonstrict semantics, which in practice means that reads may be issued before data are available in the data structure. Support for *I-structures* requires an ability to queue read requests until they can be satisfied. This mechanism is the most promising available, but it does not come cheaply. One extra memory location is required for each read instruction that can be simultaneously pending. In addition, the *I-structure* memory needs a processor of a sort to tend to pending reads when a write finally occurs. A much simpler mechanism may be used when scheduling is static.

Consider for example an actor that emits an array. This array might be carried by a single token. Suppose that there are two actors that take this array as an argument. A pure dataflow model requires that the array be copied, or at least that an implementation behave as if the array had been copied. Using an *I-structure* avoids this copying. However, with ordered-memory accesses, the copying is not necessary, and neither is the *I-structure* memory. Since the scheduler is aware of all precedences, it will avoid scheduling reads before the data become available. If this cannot be avoided (a processor has nothing to do until the data become available), then the read is attempted before the bus is granted to the processor, so the processor halts. The bus will not be granted to the processor until the data are ready. There is no need to queue accesses.

When data passes through the shared memory from one actor to another actor, the scheduler can reclaim the token storage after scheduling the read by the destination. Write-once shared-data structures are only slightly more complicated, because there may be more than one destination actor. The scheduler can simply use reference counts (RCs) [26, 16] to determine when the memory can be reclaimed. For the above example, the RC associated with the array storage would be initialized to 2, the number of destinations, when the array is scheduled to be written. Each time a read is scheduled, the RC is decremented. When it reaches 0 the memory can be reclaimed. This works without any run-time overhead because the order of these transactions will be enforced at run time.

Many variations of this idea immediately come to mind; for example, reference counts could be used for each element of the array, instead of the whole array, thereby obtaining some of the advantages of the nonstrictness of *I-structures*. Specifically, the array does not have to be completely filled

before some of its elements can be read. Also, if the RC of a data structure is identically one, then an actor using it may modify it, instead of simply reading it, something not permitted in the write-once *I-structures*. An intelligent code generator can get considerable mileage out of this.

The reference count technique has been criticized for a number of reasons [2], most of which break down when the scheduling is static. Primarily, for ordered-memory architectures, the overhead of managing RCs is incurred *only at scheduling time*, not at run time.

3. STATICALLY SCHEDULED CONTROL

The ordered-memory architecture and the static shared data structures seem to provide a very clean solution to some vexing problems. However, they are only applicable when fully static or self-timed scheduling is possible. Although this imposes some serious constraints, the constraints are less serious than they may appear at first.

The programming environment called Gabriel [39], designed for signal processing applications, is based on graphical dataflow representations of algorithms. Although specialized to signal processing, this environment has permitted extensive experimentation with scheduling algorithms and target architectures and with a style of programming that matches the need for static scheduling. As mentioned before, we have implemented a software simulation of a four-processor ordered-memory architecture [5] using the Frigg hardware simulation environment [4] and have retargeted Gabriel to this architecture. Hence, we have been able to gain some experience compiling and running real programs on this architecture.

The granularity of the actors in Gabriel is arbitrary, varying from simple arithmetic operators up to high-level signal processing functions such as FFTs. Gabriel translates dataflow graphs into sequential assembly code for programmable DSPs, performing the scheduling statically for multiple processors. A typical signal processing application contains at most 100s of actors, so we can experiment with rather complex scheduling algorithms without getting bogged down.

To be able to schedule computations statically, Gabriel restricts the dataflow model to a subclass called synchronous dataflow (SDF) [35]. We begin this section with a review of the properties of this subclass and then continue by showing that it is not as limited as it might at first appear. In particular, we show that it supports recurrences, manifest iteration, and conditional assignment, but does not support true recursion, data-dependent iteration, or conditional evaluation.

3.1. Synchronous Dataflow

A subclass of dataflow graphs lacking data dependency is well suited to static scheduling. Precisely, the term "syn-

chronous dataflow" has been coined to describe graphs that have the following property [35]:

SDF Property. A *synchronous actor* produces and consumes a fixed number of tokens on each of a fixed number of input and output paths. An *SDF graph* consists only of synchronous actors.

The basic constraint is that the number of tokens produced or consumed cannot depend on the data. An immediate consequence is that SDF graphs cannot have data-dependent firing of actors, as one might find, for example, in an if-then-else construct. In exchange for this limitation, we gain some powerful analytical and practical properties [35, 36]:

(1) For SDF graphs, the number of firings of each actor can be easily determined at compile time. If the program is nonterminating, as, for example, in real-time DSP, then a periodic schedule is always possible, and the number of firings of actors within each cycle can be determined at compile time. In either case, knowing these numbers makes it possible to construct a deterministic acyclic precedence graph. If the execution time of each actor is deterministic and known, then the acyclic precedence graph can be used to construct optimal or near-optimal schedules.

(2) For nonterminating programs, it is important to verify that memory requirements are bounded. This can be done at compile time for SDF graphs.

(3) Starvation conditions, in which a program halts due to deadlock, may not be intentional. For any SDF graph, it can be analytically determined whether deadlock conditions exist.

(4) If the execution time of each actor is known, then the maximum execution speed of an SDF graph can be determined at compile time. For terminating programs, this means finding the minimum makespan of a schedule. For nonterminating programs, this means finding the minimum period of a periodic schedule.

(5) For any nonterminating SDF graph executing according to a periodic schedule, it is possible to buffer data between actors *statically*. Static buffering means loosely that neither FIFO queues nor dynamically allocated memory are required. More specifically, it means that the compiler can statically associate memory locations with actor firings. These memory locations contain the input data and provide a repository for the output data.

These properties are extremely useful for constructing parallelizing compilers, but they apply only to SDF graphs, and optimal schedules can be constructed only when the execution times of the actors are known. We have been developing techniques that weaken the SDF constraint, thus supporting more general dataflow graphs without resorting to fully dynamic control [23]. However, these techniques require modification of the MOMA controller of the ordered-memory architecture. There is still much work to be done

to find the best design parameters, so in this paper we retain the SDF constraint.

Optimal compile-time scheduling of precedence graphs derived from SDF graphs is one of the classic NP-complete scheduling problems. Many simple heuristics have been developed over time, with some very effective ones having complexity n^2 , where n is the number of actors (see for example [25]). However, even n^2 complexity can bog down a compiler. Fortunately, the granularity of dataflow actors in Gabriel and the small size of many signal processing applications mean that we can ignore this problem for now. To generalize these methods beyond signal processing applications, strategies will probably be needed to cluster sets of actors into macro actors, thus reducing the number of actors to be considered in constructing a schedule. For example, the clustering method proposed in [31] seems suitable.

Static scheduling promises low-cost architectures, at the expense of compile-time complexity. For many applications, this is a very attractive tradeoff. However, only some applications can be statically scheduled. The SDF model, which can be statically scheduled, may appear to lack control constructs because it does not permit data-dependent firing of actors. However, this is not entirely true. Some control structures are possible within SDF, notably recurrences, manifest iteration, and conditional assignment.

3.2. Recurrences

The dataflow community has recognized the importance of supporting *recursion*, or self-referential function calls. To some extent, this ability has become a litmus test for the utility of a dataflow model. The most common implementation, however, dynamically creates and destroys instances of actors. This is clearly going to be problematic for a static scheduler.

In imperative languages, recursion is used to implement recurrences and iteration, usually in combination. If we avoid the notion of "function calls," at least some recurrences can be simply represented as feedback paths in a dataflow program graph. This section studies the representation of recurrences using feedback. This representation poses no difficulty for static scheduling, although to some it lacks the elegance of recursion.

Recurrences depend on the notion of "delays." Once understood, this notion can be used to explain fundamental limits on the concurrency in SDF graphs. It can also be used to relate SDF to static dataflow [14]. This is done below.

3.2.1. Delays

A dataflow graph with recurrence is represented schematically in Fig. 5. This graph is assumed to fire repeatedly. In terminology borrowed from the signal processing community, the feedback path has a delay, indicated with a dia-

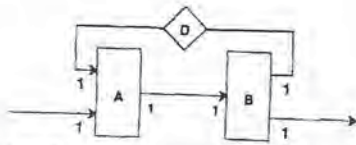


FIG. 5. A dataflow graph with a recurrence. Recurrences are expressed using direct loops and delays.

mond, which can be implemented simply as an initial token on the arc. A set of delays in a dataflow graph corresponds to a *marking* in Petri nets [45] or to the "D" tag manipulation operator in the U-interpretor [1]. In fact, the symbol D was selected to suggest delay [3]. A necessary (but not sufficient) condition for avoiding deadlock in an SDF graph is that any directed loop in the graph must have at least one delay.

A delay does not correspond to unit time delay, but rather to a single token offset. Such delays are sometimes called *logical delays* or *separators* to distinguish them from time delays [29]. For SDF graphs, a logical delay need not be a run-time operation. Consider for example the feedback arc in Fig. 5, which has a unit delay. The numbers adjacent to the arcs indicate the number of tokens produced or consumed when the corresponding actor fires. The initial token on the arc means that the corresponding input of actor A has sufficient data, so when a token arrives on its other input, it can fire. The *second* time it fires, it will consume data from the feedback arc that is produced by the *first* firing of actor B. In steady state, the n th firing of actor B will produce a token that will be consumed by actor A on its $(n + 1)$ th firing; hence the arc has unit token offset. The *value* of the initial token can be set by the programmer, so a delay can be used to initialize a recurrence. When the initial value is other than zero, we indicate it using the notion $D(\text{value})$. Since delays are simply initial conditions on the buffers, they require no run-time overhead. In Gabriel, a delay is a property of an arc in the dataflow graph, rather than an actor.

3.2.2. Bounds on Performance

Consider nonterminating algorithms, or algorithms that operate on a large data set. For these, directed loops are the only fundamental limitation on the parallelizability of the algorithm. This is intuitive because any algorithm without recurrences can be pipelined. A special case of SDF, called *homogeneous SDF*, is where every actor produces and consumes a single token on each input and output. For homogeneous SDF graphs, it is easy to compute the minimum period at which an actor can be fired. This is called the *iteration period bound* and is the reciprocal of the maximum computation rate. The iteration period bound may be much smaller than the time required to compute one pass through the dataflow graph. It is a limit on the time per pass if an infinite number of passes are computed.

Let $R(L)$ be the sum of the execution times of the actors in a directed loop L . The iteration period bound is the maximum over all directed loops L of $R(L)/D(L)$, where $D(L)$ is the number of delays in L [47, 10]. The directed loop L that yields this maximum is called the *critical loop*. General SDF graphs can be systematically converted to homogeneous SDF graphs for the purpose of computing the iteration period bound [34]. If there are no directed loops in the graph, then we define the iteration period bound to be zero, since in principle all firings of each node could occur simultaneously. It is important to realize that there is nothing fundamental in the following discussion that prevents this. Implementation considerations may make it impractical, however.

Another limitation on concurrency is the notion of state. Particularly in large- or medium-grain dataflow graphs, it is convenient to permit an actor to remember data from one invocation to the next. This is simply modeled as a self-loop with a unit delay. Such a self-loop precludes multiple simultaneous invocations of the actor, hence this self-loop may become the critical loop.

Once the iteration period bound is known, we can derive a bound on the performance of an ordered-memory architecture, on the basis of a set of (admittedly) unrealistic assumptions. First, assume that we have a completely deterministic dataflow graph, and assume that there are enough processors that a hypothetically optimal scheduler can meet the iteration period bound. The iteration period bound does not reflect bandwidth or latency limitations on interprocessor communication, however. For the ordered-memory architecture of Fig. 3, a memory transaction can occur in one cycle of the shared memory. If we assume that the shared-memory cycle time is the same as local-memory cycle time,¹ then latency adds nothing to the iteration period. Bandwidth limitations, however, may add to the iteration period. Each time the ideal scheduler schedules two simultaneous memory transactions, one of them must be delayed. If one of them is not in the critical path, then that one should be delayed, and there may again be no effect on the iteration period. If both are in the critical path, then the iteration period will be extended by one cycle. If three transactions are scheduled simultaneously, then one of the transactions has to be delayed two cycles, increasing the iteration period by at most two cycles. If M simultaneous transactions are scheduled, then the iteration period increases by at most $M - 1$ cycles. If the total number of transactions is T , then the very worst situation increases the iteration bound by at most $T - 1$ cycles.

Suppose now that 10 processors are each running a program that accesses shared memory 10% of the time. Then

¹ This is actually not a bad assumption for the architecture in Fig. 3, if the number of processors is modest. The main limitation on shared-memory cycle time is likely to be capacitive loading on the shared bus, and the price of the memory, of course.

this bound tells us that the ordered shared memory architecture can run this program in at most twice the time of the theoretical minimum. However, this result should not be taken very seriously because the performance will depend much more heavily on the scheduling heuristics used. The performance can be better (since simultaneous transactions can be studiously avoided [48]) but can also be worse (if, as is likely, a suboptimal scheduling algorithm is used). Furthermore, the use of gateways completely undermines this analysis. Thus, this is not a very useful bound.

3.2.3. Bounded Buffer Sizes

Although SDF actors cannot be created at run time, SDF is not the same as static dataflow [14]. For instance, in SDF, there is no impediment to having multiple instances of an actor fire simultaneously, as long as the actor does not have state. A particular implementation, however, may impose such a constraint. Consider for example an implementation that permits no more than one memory location to be associated with each arc. This is the key limitation in static dataflow [14]. It can be modeled with the recurrence in Fig. 5. The feedback arc begins with an initial token. This token represents a "space" on the output buffer of actor A. After A fires and consumes that token, it cannot fire again until after B has fired. Any memory limitation on any arc in an SDF graph can be modeled as a feedback path with a fixed number of delays. To avoid unnecessarily sacrificing concurrency, enough memory should be allocated to each arc that the corresponding feedback path does not become the critical loop.

Suppose that, in Fig. 5, actors A and B are scheduled onto different processors. In a conventional shared-memory architecture, any buffer size limitation implies handshaking at run time. In effect, the feedback path in Fig. 5 has to be implemented at run time, just to carry semaphores that indicate when it is safe to write to the feedforward buffer. For the ordered-memory architecture, however, buffer size limitations can be statically modeled by the scheduler. They imply no additional run-time overhead. In Fig. 5, the scheduler knows that the write from A and the read to B must alternate. Since the order of the transactions is enforced at run time without semaphores, no additional overhead is incurred.

3.3. Manifest Iteration

In manifest iteration, the number of repetitions of a computation is known at compile time and hence is independent

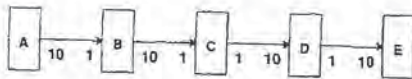


FIG. 6. An SDF graph that contains nested iteration.

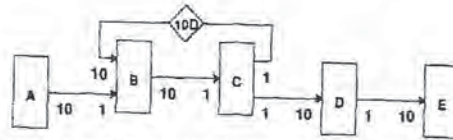


FIG. 7. A modification of Fig. 6 to model the effect of a buffer of length 10 between actors B and C.

of the data. It can be statically scheduled. Furthermore, it can be expressed in dataflow graphs by specifying the number of tokens produced and consumed each time an actor fires. For example, actor A in Fig. 6 produces 10 tokens each time it fires, as indicated by the "10" adjacent to its output. Actor B consumes 1 token each time it fires, so it will fire 10 times for every firing of actor A. In conventional programming languages, this would be expressed with a *for* loop. Nested *for* loops are easily conceived as shown in Fig. 6. If actors A and E fire once each, then B and D will fire 10 times, and C will fire 100 times. Techniques for automatically constructing static parallel schedules for such graphs are given in [35] and [48].

There is no fundamental limitation on the parallelism in Fig. 6 (there are no directed loops). Hence, this scheme solves the first open problem listed by Dennis in [13], providing the semantics of a "parallel-for" in dataflow.

3.3.1. Bounded Buffers

Although there is no fundamental limitation on the parallelism in Fig. 6 (there are no directed loops), there may be practical limitations. In Fig. 7, we model a buffer of length 10 between actors B and C. Again, the tokens on the feedback path represent empty locations in the buffer. Actor B must have 10 tokens on the feedback path (i.e., 10 empty locations in the buffer) before it fires. Whenever actor C fires, it consumes 1 token from the forward path, freeing a buffer location, and indicating the free buffer location by putting a token on the feedback path. The minimum buffer size that avoids deadlock is 10.

This nonhomogeneous SDF graph could be converted to a homogeneous SDF graph and the iteration period bound computed, but in this simple example the iteration period bound is easily seen by inspection. It is clear that after each firing of B, C must fire 10 times before B can fire again. The 10 firings can occur in parallel, so the minimum period of a periodic schedule is $R_B + R_C$, where R_x is the run time of actor x . In other words, successive firings of B cannot occur in parallel because of the buffer space limitations. By contrast if the buffer had length 100, then 10 invocations of B could fire simultaneously, assuming there are no other practical difficulties. Just as with unit length buffers, no additional synchronization overhead is required in the ordered-memory architecture to support these bounded buffers.

3.3.2. Static Buffers

A second limitation on the parallelism can arise from the addressing mechanism of the buffers. Each buffer can be implemented as a FIFO queue, as was done in Davis' DDM [12]. Delays are correctly handled, but then access to the buffer becomes a critical section of the parallel code. FIFO queues are economically implemented as circular buffers with pointers to the read and write locations. However, parallel access to the pointers becomes a problem. If successive invocations of an actor are to fire simultaneously on several processors, then great care must be taken to ensure the integrity of the pointers. A typical approach would be to lock the pointers while one processor has control of the FIFO queue, but this partially serializes the implementation. Furthermore, this requires that the hardware support an indivisible test-and-set operation.

In the ordered-memory architecture, the FIFO implementation can be made simpler than in a general shared-memory architecture, but a less expensive alternative is static buffering [36]. Static buffering is based on the observation that there is a periodicity in the buffer access that a compiler can exploit. It preserves the behavior of FIFO queues (namely, it correctly handles delays and ordering of tokens), but avoids read and write pointers. Specifically, suppose that all buffers are implemented with fixed-length circular buffers, implementing FIFO queues, where each length has been predetermined to be long enough to sustain the run without causing a deadlock. Then consider an input of any actor in an SDF graph. Every N firings, where N is to be determined, the actor will get its input token(s) from the same memory locations. The compiler can hard-code these memory locations into the implementation, bypassing the need for pointers to the buffer. Systematic methods for doing this, developed in [36], can be illustrated by example. Consider the graph in Fig. 7, which is a representation of Fig. 6 with the buffer between B and C assigned the length 10. A parallel implementation of this can be represented as follows:

```

FIRE A
DO ten times {
  FIRE B
  DO in parallel ten times {
    FIRE C
  }
  FIRE D
}
FIRE E

```

For each parallel firing of C, the compiler supplies a *specific* memory location for it to get its input tokens. Note that this would not be possible if the FIFO buffer had length 11, for example, because the second time the inner DO loop is executed the memory locations accessed by C would not be

the same as the first time. But with a FIFO buffer of length 10, invocations of C need not access the buffer through pointers, so there is no contention for access to the pointers. The buffer data can be supplied to all 10 firings in parallel, assuming the hardware has a mechanism for doing this. In the ordered-memory architecture, the 10 firings cannot be initiated simultaneously, because of bus bandwidth limitations. However, they can be initiated at intervals of one shared-bus cycle. If this cycle time is small compared to the execution time of the actors, then the concurrency in the parallel-for is adequately exploited.

An alternative to static buffering that also permits parallel firings of successive instances of the same actor is token matching [1]. However, even the relatively low cost of some implementations of token matching [44] would be hard to justify for SDF graphs, where static buffering can be used.

In Fig. 6 we use actors that produce more tokens than they consume, or consume more tokens than they produce. Proper design of these actors can lead to iteration constructs semantically similar to those encountered in conventional programming languages. In Fig. 8 we show three such actors that have proved useful in DSP applications. The first, Fig. 8a, simply emits the last of N tokens, where N is a parameter of the actor. The second, Fig. 8b, takes one input token and repeats it on the output. The third, Fig. 8c, takes one input token each time it fires and emits the last N tokens that arrived. It has a self-loop used to remember the past tokens (and initialize them). This can be viewed as the state of the actor; it effectively prevents multiple simultaneous invocations of the actor.

A complete iteration model must include the ability to nest recurrences within the iteration and to initialize the recurrences when the iteration begins. The SDF model can handle this. We illustrate this with a finite impulse response (FIR) digital filter because it is a simple example. An FIR filter computes the inner product of a vector of coefficients and a vector with the last N input tokens, where N is the order of the filter. It is usually assumed to repeat forever, firing each time a new input token arrives. Consider the possible implementations using a dataflow graph. A large-grain approach is to define an actor with the implementation details hidden inside. This is the preferred approach in Gabriel. An alternative is a fine-grain implementation with multiple adders and multipliers and a delay line. A third possibility is to use iteration and a single adder and multiplier. The first

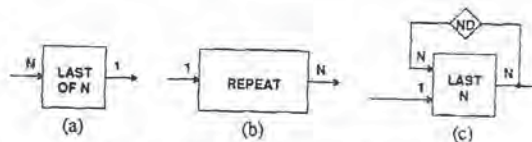


FIG. 8. Three SDF actors useful for iteration.

and last possibilities have the advantage that the complexity of the dataflow graph is independent of the order of the filter. A good compiler should be able to do as well with any of the three structures. One implementation of the last possibility is shown in Fig. 9. The iteration actors are drawn from Fig. 8. The coefficients actor simply outputs a stream of N coefficients: it produces one coefficient each time it fires and reverts to the beginning of the coefficient list after reaching the end. It could be implemented with a directed loop with the N delays, or a number of other ways. The product of the input data and the coefficients is accumulated by the adder with a feedback loop. The output of the filter is selected by the "last of N " actor.

The FIR filter in Fig. 9 has the advantage of exploitable concurrency combined with a graph complexity that is independent of the order of the filter. Note, however, that there is a difficulty with the feedback loop at the adder. Recall from the above that a delay is simply an initial token on the arc. If this initial token has value zero, then the first output of the FIR filter will be correct. However, after every N firings of the adder, we wish to *reset* the token on that arc to zero. This could be done with some extra actors, but a fundamental difficulty would remain. The presence of that feedback loop implies a limitation on the parallelism of the FIR filter, and that limitation would be an artifact of our implementation. Our solution is to introduce the notion of a *resetting delay*, indicated with a diamond containing an R.

3.3.3. Resetting Delays

A resetting delay is associated with a subgraph, which in Fig. 9 is surrounded by a dashed line. For each invocation of the subgraph, the delay token is reinitialized to zero. Furthermore, the scheduler knows that the precedence is broken when this occurs, and consequently it can schedule successive FIR output computations simultaneously on separate processors.

The resetting delay can be used in any SDF graph where we have nested iterations where the inner iterations involve recurrences that must be initialized. In other words, anything of the form

```
DO some number of times {
  Initialize X
```

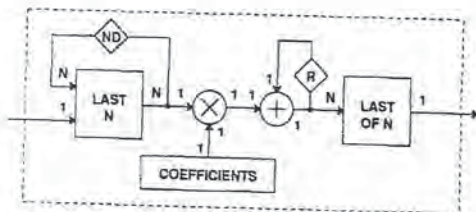


FIG. 9. An FIR filter implemented using a single multiplier and adder.

```
DO some number of times {
  new X = f(X)
}
```

The implementation of a resetting delay is simple and general. For the purposes of implementation, the scheduler first treats the delay as if it were an actor that consumes one token and produces one token each time it fires. Recall that in practice no run-time operation is required to implement a delay, so there actually is no such actor. However, by inserting this mythical actor, the scheduler can determine how many times it would fire (if it did exist) for each firing of the associated subgraph. The method for doing this is given in [35] and consists of solving a simple system of equations. For each resetting delay, the scheduler obtains a number N of invocations between resets; this number is used to break the precedence of the arc for every N th token and to insert an object code that reinitializes the delay value. The method works even if the subgraph is not invoked as a unit and even if it is scattered among the available processors. It is particularly simple when inline code is generated. However, when the iteration is implemented by the compiler using loops, then a small amount of run-time overhead may have to be associated with some delays in order to count invocations.

So far we have shown that neither manifest iteration nor recurrences present a fundamental problem for the SDF model. Resetting delays can be used to initialize recurrences within nested iterations. Hence corresponding programming constructs can be efficiently and automatically implemented on an ordered-memory architecture. Conditionals are a bit more problematic.

3.4. Conditional Assignment

Conditionals in dataflow graphs are harder to describe and schedule statically. One attractive solution is a mixed-mode programming environment, where the programmer can use dataflow at the highest level and conventional languages such as C at a lower level. Gabriel is precisely such an environment. Conditionals would be expressed in the conventional language. This is only a partial solution, however, because conditionals would be restricted to lie entirely within one large-grain actor, and concurrency within such actors is difficult to exploit. If the complexity of the operations that are performed conditionally is high, then this approach is not adequate. Furthermore, conditionals within an actor usually imply a nondeterministic execution time of the actor. If the variability of the possible execution times is high, the performance of the ordered-memory architecture will suffer.

A simple alternative that is sometimes suitable to replace *conditional evaluation* with *conditional assignment*. The functional expression

$$y \leftarrow \text{if}(c) \text{ then } f(x) \text{ else } g(x)$$

can be implemented as shown in Fig. 10. The MUX actor consumes a token on each of the T, F, and control inputs and copies either the T or the F token to the output. Hence, both $f(x)$ and $g(x)$ will be computed and only one of the results will be used. When these functions are simple, this approach is efficient; indeed it is commonly used in deeply pipelined processors to avoid conditional branches. For hard real-time applications, it is also efficient when *one of the two* subgraphs is simple. Otherwise, however, the cost of evaluating both subgraphs may be excessive, so alternative techniques are required.

4. A NOTE ON QUASI-STATIC SCHEDULING

The domain of applications of the ordered-memory architecture is constrained by the need to statically order shared-memory accesses. An automatic parallelizing compiler has been written to work with SDF graphs where the actor execution times are reasonably predictable. Since the SDF model supports recurrences, manifest iteration, and conditional assignment, it is not as limited as it might at first appear. Nonetheless, it is worth attempting to weaken the constraints of the SDF model in order to encompass more applications. At Berkeley we have been developing *quasi-static* scheduling strategies that may solve some of these problems [23]. The basic principle is that dynamic control is used only where absolutely necessary. For instance, with an if-then-else, control is dynamically transferred to one of two statically scheduled subgraphs. Similarly, for a data-dependent iteration (such as a do-while), a static schedule for each cycle of the iteration is dynamically repeated. The challenge, of course, is to develop strategies for constructing the static schedules for the subgraphs. Furthermore, these techniques imply changes to the ordered-memory architecture. The MOMA controller can no longer simply passively step through a static list of processors to which it must grant the bus. Instead, it has to follow the control path of the distributed executing program. The challenge is to accomplish this without increasing the complexity of the controller so much that the advantages of ordered-memory accesses evaporate. This is an active and promising line of inquiry.

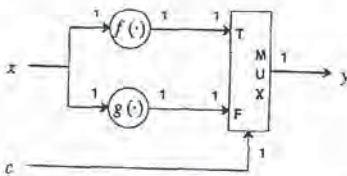


FIG. 10. A dataflow graph with conditional assignment. Both $f(\cdot)$ and $g(\cdot)$ are evaluated, and only one of the two outputs is selected.

5. CONCLUSIONS

It is well known that data-independent dataflow graphs can be scheduled statically, obviating the need for additional run-time hardware to control the execution. We have illustrated low-cost parallel architectures that take advantage of this and have shown that shared data structures can also be supported efficiently. A software simulation of an ordered-memory architecture has been built along with a compiler that fully automates the mapping. The compiler begins with a large-grain dataflow graph that conforms with the synchronous dataflow model of computation. This SDF model, although limited, can support recurrences, manifest iteration, and conditional assignment. The execution times of actors can vary slightly without seriously affecting the implementation, but wide variations can have considerable adverse impact on execution speed. For applications with little decision making, such as signal processing and some scientific computing, this approach appears attractive. To broaden the application base, quasi-static scheduling may provide a solution by introducing dynamic control only where absolutely necessary [23].

ACKNOWLEDGMENTS

Anonymous reviewers made many useful comments that greatly improved the organization and clarity of this paper.

REFERENCES

1. Arvind, and Gostelow, K. P. The U-interpretor. *Computer* 15, 2 (Feb. 1982).
2. Arvind, Nikhil, R. S., and Pingali, K. K. I-structures: Data structures for parallel computing. Computation Structures Group Memo 269, MIT, Feb. 1987 (revised Mar. 1989); *ACM Trans. Programming Languages Systems*, to appear.
3. Arvind. Private communication. 1989.
4. Bier, J. C., and Lee, E. A. Frigg: A simulation environment for multiple-processor DSP system development. *Proc. International Conference on Computer Design*, Boston, Oct. 2-4, 1989.
5. Bier, J., and Lee, E. A. A class of multiprocessor architectures for real-time DSP. *Proc. International Conference on Circuits and Systems*, New Orleans, May 1990.
6. Burton, F. W., and Sleep, M. R. Executing functional programs on a virtual tree of processors. *Proc. ACM Conference Functional Programming Language Computer Architecture*, 1981, pp. 187-194.
7. Chase, M. A pipelined data flow architecture for signal processing: The NEC uPD7281, in *VLSI Signal Processing*, IEEE, New York, 1984.
8. Chu, W. W., Holloway, L. J., Lan, L. M.-T., and Fife, K. Task allocation in distributed data processing. *IEEE Computer*, (Nov. 1980), 57-69.
9. Chu, W. W., and Lan L. M.-T. Task allocation and precedence relations for distributed real-time systems. *IEEE Trans. Comput.* C-36, 6 (June 1987), 677-679.
10. Cohen, G., Dubois, D., Quadrat, J. P., and Viot, M. A linear-system-theoretic view of discrete-event processes and its use for performance

- evaluation in manufacturing. *IEEE Trans. Automat. Control* AC-30 (1985), 210-220.
11. Cornish, M., Hogan, D. W., and Jensen, J. C. The Texas Instruments distributed data processor. *Proc. Louisiana Computer Exposition*, Lafayette, LA, Mar. 1979, pp. 189-193.
 12. Davis, A. L. The architecture and system method of DDM1: A recursively structured data driven machine. *Proc. Fifth Annual Symposium Computer Architecture*, Apr. 1978, pp. 210-215.
 13. Dennis, J. B. First version data flow procedure language. Tech. Memo MAC TM61, MIT Laboratory for Computer Science, May 1975.
 14. Dennis, J. B. Data flow supercomputers. *Computer* 13, 11 (Nov. 1980).
 15. Dennis, J. B., and Gao, G. R. An efficient pipelined dataflow processor architecture. *Proc. IEEE*, to appear. *Proc. ACM SIGARCH Conference on Supercomputing*, FL, Nov. 1988.
 16. Driscoll, J., Sarnak, N., Sleator, D., and Tarjan, R. Making data structures persistent. *Proc. 18th Annual ACM Symposium on Theory of Computing*, Berkeley, CA, May 1986, pp. 109-121.
 17. Efe, K. Heuristic models of task assignment scheduling in distributed systems. *IEEE Comput.* (June 1982), 50-56.
 18. Fisher, J. A. The VLIW machine: A multiprocessor for compiling scientific code. *Computer* 17, 4 (July 1984).
 19. Gao, G. R., Tio, R., and Hum, H. H. J. Design of an efficient dataflow architecture without dataflow. *Proc. International Conference on Fifth Generation Computer Systems*, 1988.
 20. Gaudiot, J. L. Structure handling in data-flow systems. *IEEE Trans. Comput.* C-35, 6 (June 1986).
 21. Gaudiot, J. L. Data-driven multicomputers in digital signal processing. *IEEE Proc.* 75, 9 (Sept. 1987), 1220-1234.
 22. Granski, M., Koren, I., and Silberman, G. M. The effect of operation scheduling on the performance of a data flow computer. *IEEE Trans. Comput.* C-36, 9 (Sept. 1987).
 23. Ha, S., and Lee, E. A. Compile-time scheduling and assignment of dataflow program graphs with data-dependent iteration. To appear, *IEEE Trans. Parallel Distrib. Comput.* Also Memorandum UCB/ERL M89/57, Electronics Research Lab. UC Berkeley, 1989.
 24. Hoare, C. A. R. Communicating sequential processes. *Comm. ACM* 21, 8 (Aug. 1978).
 25. Hu, T. C. Parallel sequencing and assembly line problems. *Oper. Res.* 9, 6 (1961), 842-848.
 26. Hudak, P. A semantic model of reference counting and its abstraction. *Proc. 1986 ACM Conference on Lisp and Functional Programming*, MIT, Cambridge, MA, Aug. 1986, pp. 351-363.
 27. Iannucci, R. A. A dataflow/von Neumann hybrid architecture. Tech. Rep. TR-418, MIT Laboratory for Computer Science, May 1988.
 28. Iqbal, M. A., Saltz, J. H., and Bokhari, S. H. A comparative analysis of static and dynamic load balancing strategies. *Proc. International Conference on Parallel Processing*, 1986, pp. 1040-1045.
 29. Jagadish, H. V., Mathews, R. G., Kailath, T., and Newkirk, J. A. A study of pipelining in computer arrays. *IEEE Trans. Comput.* C-35, 5 (May 1986).
 30. Keller, R. M., Lin, F. C. H., and Tanaka, J. Rediflow multiprocessing. *Proc. IEEE COMPCON*, Feb. 1984, pp. 410-417.
 31. Kim, S. J., and Browne, J. C. A general approach to mapping of parallel computations upon multiprocessor architectures. *Proc. 1988 International Conference on Parallel Processing*, Aug. 1988, pp. 1-8.
 32. Kunkel, J. Parallelism in COSSAP. *Internal Memorandum*, Aachen University of Technology, Federal Republic of Germany, 1987.
 33. Kung, S. Y. *VLSI Array Processors*. Prentice-Hall, Englewood Cliffs, NJ, 1988.
 34. Lee, E. A. A coupled hardware and software architecture for programmable digital signal processors. *Memorandum UCB/ERL M86/54*, EECS Department, UC Berkeley (Ph.D. Dissertation), 1986.
 35. Lee, E. A., and Messerschmitt, D. G. Static scheduling of synchronous data flow graph for digital signal processing. *IEEE Trans. Comput.* C-36, 1 (Jan. 1987), pp. 24-35.
 36. Lee, E. A., and Messerschmitt, D. G. Synchronous data flow. *IEEE Proc.* 75, 9 (Sept. 1987), pp. 1235-1245.
 37. Lee, E. A. Programmable DSP architectures. Part I. *ASSP Mag.* 5, 4 (Oct. 1988), pp. 4-19.
 38. Lee, E. A. Programmable DSP architectures. Part II. *ASSP Mag.* 6, 1 (Jan. 1989), pp. 4-14.
 39. Lee, E. A., Ho, W.-H., Goei, E., Bier, J., and Bhattacharyya, S. Gabriel: A design environment for DSP. *IEEE Trans. Acoust. Speech Signal Processing* 37, 11 (Nov. 1989), pp. 1751-1762.
 40. Lu, H., and Carey, M. J. Load-balanced task allocation in locally distributed computer systems. *Proc. International Conference on Parallel Processing*, 1986, pp. 1037-1039.
 41. Ma, P. R., Lee, E. Y. S., and Tsuchiya, M. A task allocation model for distributed computing systems. *IEEE Trans. Comput.* C-31, 1 (Jan. 1982), 41-47.
 42. Muhlenbein, H., Gorges-Schleuter, M., and Kramer, O. New solutions to the mapping problem of parallel systems: The evolution approach. *Parallel Comput.* 4, (1987), 269-279.
 43. Nikhil, R. S., and Arvind. Can dataflow subsume von Neumann computing? *Proc. 16th Annual International Symposium on Computer Architecture*, Jerusalem, Israel, May 28-June 1, 1989.
 44. Papadopoulos, G. M. *Implementation of a general purpose dataflow multiprocessor*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT, Aug. 1988.
 45. Peterson, J. L. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
 46. Plas, A., et al. LAU system architecture: A parallel data-driven processor based on single assignment. *Proc. 1976 International Conference Parallel Processing*, 1976, pp. 293-302.
 47. Renfors, M., and Neuvo, Y. The maximum sampling rate of digital filters under hardware speed constraints. *IEEE Trans. Circuits Systems* CAS-28, 3 (Mar. 1981).
 48. Sih, G., and Lee, E. A. Scheduling to account for interprocessor communication within interconnection-constrained processor networks. *Proc. International Conference on Parallel Processing*, Aug. 1990.
 49. Watson, L., and Gurd, J. A practical data flow computer. *Computer* 15, 2 (Feb. 1982).
 50. Zissman, M. A., and O'Leary, G. C. A block diagram compiler for a digital signal processing MIMD computer. *IEEE International Conference on ASSP*, 1987, pp. 1867-1870.

EDWARD A. LEE has been an assistant professor in the Electrical Engineering and Computer Science Department at U.C. Berkeley since July 1986. His research activities include parallel computation, architecture and software techniques for programmable DSPs, design environments for real-time software development, and digital communication. He was a recipient of the 1987 NSF Presidential Young Investigator award, an IBM faculty development award, and the 1986 Sakrison prize at U.C. Berkeley for the best thesis in electrical engineering. He is coauthor of "Digital Communication" with D. G. Messerschmitt, Kluwer Academic Press, 1988, and "Digital Signal Processing Experiments" with Alan Kamas, Prentice-Hall, 1989, as well as numerous technical papers. His B.S. degree is from Yale University

(1979), his masters (S.M.) from MIT (1981), and his Ph.D. from U.C. Berkeley (1986). From 1979 to 1982 he was a member of the technical staff at Bell Telephone Laboratories in Holmdel, New Jersey, in the Advanced Data Communications Laboratory, where he did extensive work with early programmable DSPs, and exploratory work in voiceband data modem techniques and simultaneous voice and data transmission.

JEFFREY C. BIER received the B.S.E. degree in electrical engineering from Princeton University, Princeton, New Jersey in 1987. From 1987 to

1989 he was a graduate student and research assistant at the University of California, Berkeley, where he earned his M.S.E.E. in 1989. His professional interests are in the areas of special-purpose architectures, simulation and design tools, digital signal processing, and multiprocessor systems. Mr. Bier is now with Acuson, in Mountain View, California, where he is engaged in the development of technology for medical diagnostic ultrasound imaging systems. Previously, he has held positions with Quinn and Feiner, Inc., Glen Cove, New York, and Hewlett-Packard Laboratories, Palo Alto, California. Mr. Bier is a member of Tau Beta Pi and the IEEE ASSP and Computer Societies.

Received December 1, 1989; revised May 30, 1990; accepted July 31, 1990

Attachment 4A

May 1998

Journal of

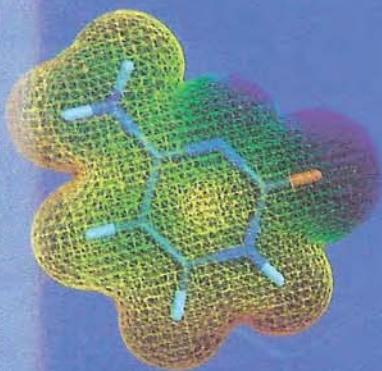
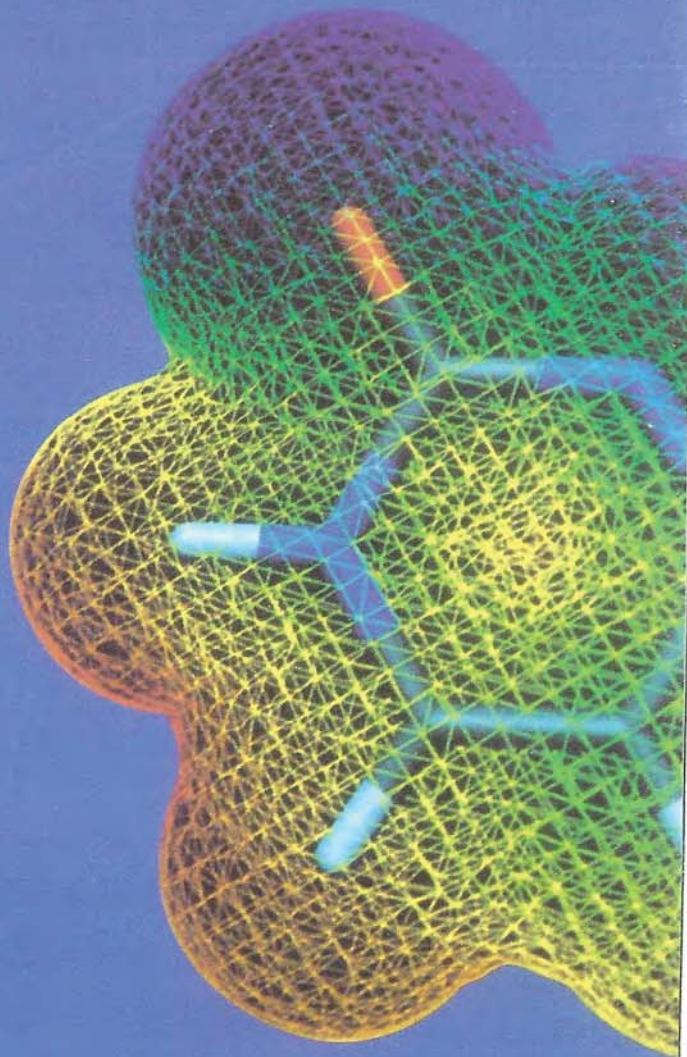
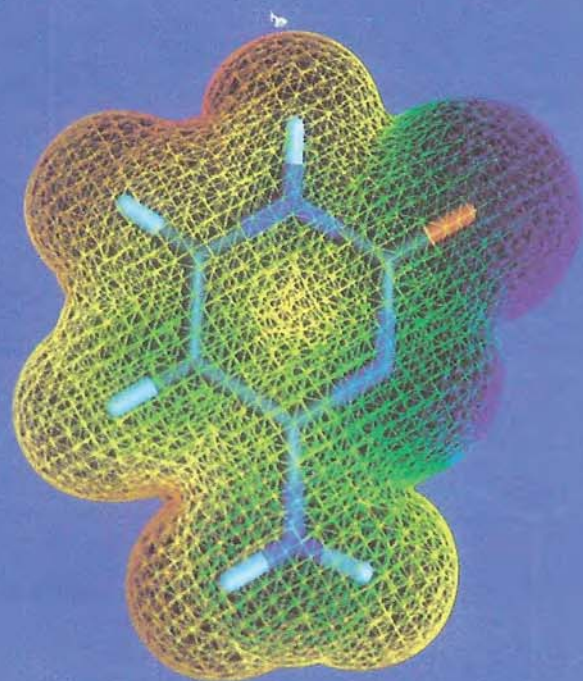
Computational Chemistry

ORGANIC / INORGANIC / PHYSICAL / BIOLOGICAL

Editors:

Norman L. Allinger

Paul von R. Schleyer



Visit Wiley Journals Online
www.interscience.wiley.com



WILEY-INTERSCIENCE

ISSN 0192-8651

Journal of Computational Chemistry

ORGANIC / INORGANIC / PHYSICAL / BIOLOGICAL

Editors:

Norman L. Allinger
Department of Chemistry
Computational Center for
Molecular Structure and
Design

University of Georgia
Athens, GA 30602-2526

Paul von R. Schleyer
Computer Chemistry Center
Institut für Organische Chemie
Universität Erlangen-Nürnberg
Henkestrasse 42, D-91054
Erlangen, Germany

Associate Editor:

Gernot Frenking
Fachbereich Chemie
Philipps-Universität Marburg
Hans-Meerwein-Strasse
D-35032 Marburg, Germany

Assistant Editors:

Martin Feigel
Institut für Organische Chemie
Universität Bochum
Universitätsstr. 150
D-44780 Bochum, Germany

Roger S. Grev
Department of Chemistry
University of Kentucky
Lexington, KY 40506

Editorial Advisory Board:

Enrico Clementi
Department of Chemistry
University L. Pasteur, 3
rue de l'Université
67084 Strasbourg, France

Warren J. Hehre
Wavefunction, 18401 Von
Karman, Suite 370
Irvine, CA 92715

William L. Jorgensen
Department of Chemistry
Yale University
PO Box 208107
New Haven, CT 06520-8107

Martin Karplus
Department of Chemistry
12 Oxford Street
Harvard University
Cambridge, MA 02138

Peter A. Kollman
Department of Pharmaceutical
Chemistry
School of Pharmacy
University of California
San Francisco, CA 94143

Paul G. Mezey
Department of Chemistry
University of Saskatchewan
Saskatoon, Canada S7N 0W0

Keiji Morokuma
Department of Chemistry
Emory University
Atlanta, GA 30322

Eiji Osawa
Department of Knowledge-
Based Information
Engineering
Toyohashi University of
Technology
Tempaku-cho, Toyohashi
441, Japan

John A. Pople
Department of Chemistry
Northwestern University
2145 Sheridan Road
Wilmette, IL 60208

Peter Pulay
Department of Chemistry
University of Arkansas
Fayetteville, AR 72701

Leo Radom
Research School of Chemistry
The Australian National
University
Canberra, A.C.T. 0200
Australia

Harold A. Scheraga
Baker Laboratory of Chemistry
Cornell University
Ithaca, NY 14853-1301

Andrew Streitwieser, Jr.
Department of Chemistry
University of California
Berkeley
Berkeley, CA 94720

Kenneth B. Wiberg
Department of Chemistry
Yale University
225 Prospect Street
New Haven, CT 06520

Michael C. Zerner
Department of Chemistry
University of Florida
Gainesville, FL 32611

Editorial Production, John Wiley: Merilee Croft Olson

The *Journal of Computational Chemistry* (ISSN: 0192-8651) is published 16 times a year, monthly and semi-monthly in January, April, July, and November, one volume per year by John Wiley & Sons, Inc.: 605 Third Avenue, New York, NY, 10158.

Copyright © 1998 John Wiley & Sons, Inc. All rights reserved. No part of this publication may be reproduced in any form or by any means, except as permitted under section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the publisher, or authorization through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923; tel: 508-750-8400, fax: 508-750-4470. Periodicals postage paid at New York, NY, and at additional mailing offices.

The code and the copyright notice in the journal indicate the copyright holders consent that copies may be made for personal or internal use, or for the personal or internal use of specific clients, on the condition that the copier pay for copying beyond that permitted by Sections 107 and 108 of the United States Copyright Law. The per-copy fee for each item is to be paid through the Copyright Clearance Center, Inc.

This consent does not extend to other kinds of copying, such as copying for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. Such permission requests and other permission inquiries should be addressed to the Permissions Dept.

Subscription price (Volume 19, 1998): \$1,198.00 in US, \$1,358.00 in Canada and Mexico, \$1,475.00 outside North America. \$150.00 US ACS member, \$246.00 outside US ACS member. Personal rate: \$150.00 in US, \$246.00 outside North America. Subscriptions at the personal rate are available only to individuals. All subscriptions outside US will be sent by air. Payment must be made in US dollars drawn on a US bank. Claims for undelivered copies will be accepted only after the following issue has been received. Please enclose a copy of the mailing label. Missing copies will be supplied when losses have been sustained in transit and where reserve stock permits. Please allow four weeks for processing a change of address. For subscription inquiries, please call 212-850-6645; E-mail: SUBINFO@jwiley.com

Postmaster: Send address changes to *Journal of Computational Chemistry*, Caroline Rollhaug, Director, Subscription Fulfillment and Distribution, John Wiley & Sons, Inc., 605 Third Avenue, New York, NY 10158. For subscription inquiries, please call customer service at 212-850-6645 or write to the above address.

Advertising: Inquiries concerning advertising should be forwarded to Susan Levey, Advertising Sales, John Wiley & Sons, Inc., 605 Third Avenue, New York, NY 10158, 212-850-8832. Advertising Sales, European Contacts: Bob Kem or Nicky Douglas, John Wiley & Sons, Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England. Tel: 44 1243 770 350/367; Fax: 44 1243 770 432; e-mail: adsales@wiley.co.uk.

Reprints: Reprint sales and inquiries should be directed to the customer service department, John Wiley & Sons, Inc., 605 Third Ave., New York, NY 10158; tel: 212-850-8776.

Manuscripts should be submitted to one of the editors: Dr. Norman L. Allinger, *Journal of Computational Chemistry*, Department of Chemistry, The University of Georgia, Athens, GA 30602; Prof. Dr. Paul von R. Schleyer, *Journal of Computational Chemistry*, Computer Chemistry Center, Institut für Organische Chemie, Universität Erlangen-Nürnberg, Henkestr. 42, D-91054, Erlangen, Germany; or Prof. Gernot Frenking, *Journal of Computational Chemistry*, Fachbereich Chemie, Philipps-Universität Marburg, Hans-Meerwein-Strasse, D-35032 Marburg, Germany.

Information for contributors appears in the first and last issue of each volume. All other correspondence should be addressed to *Journal of Computational Chemistry*, Publisher, Interscience Division, Professional, Reference, and Trade Group, John Wiley & Sons Inc., 605 Third Avenue, New York, NY 10158.

The contents of this journal are indexed in the following: *Chemical Abstracts*, *Chemical Titles*, *Current Contents/Physical, Chemical, & Earth Sciences*, *Mathematical Reviews*, *Reference Update*, *Research Alert (ISI)*, *Science Citation Index (ISI)*, and *SCISEARCH Database (ISI)*.

∞ This paper meets the requirements of ANSI/NISO Z39.48-1992 (Permanence of paper).

Journal of Computational Chemistry

Volume 19/Number 7/May 1998

CONTENTS

- Development of a Parallel Molecular Dynamics Code on SIMD Computers:
Algorithm for Use of Pair List Criterion 685
D. Roccatano, R. Bizzarri, G. Chillemi, N. Sanna, and A. Di Nola
- Multivariate Analysis of a Data Matrix Containing A-DNA and B-DNA
Dinucleoside Monophosphate Steps: Multidimensional Ramachandran
Plots for Nucleic Acids 695
M. L. M. Beckers and L. M. C. Buydens
- Ab Initio* Prediction of ^{15}N -NMR Chemical Shift in α -Boron Nitride
Based on an Analysis of Connectivities 716
Marcus Gastreich and Christel M. Marian
- The Flying Ice Cube: Velocity Rescaling in Molecular Dynamics
Leads to Violation of Energy Equipartition 726
Stephen C. Harvey, Robert K.-Z. Tan, and Thomas E. Cheatham III
- Systematic Prediction of the Products and Intermediates of Isotopic
Labeling in Reaction Pathway Studies 741
Andrew V. Zeigarnik and Raúl E. Valdés-Pérez
- Electrostatic Model for Infrared Intensities in a Spectroscopically
Determined Molecular Mechanics Force Field 754
Kim Palmo and Samuel Krimm
- Solvation Free Energies Calculated Using the GB/SA Model: Sensitivity
of Results on Charge Sets, Protocols, and Force Fields 769
*M. Rami Reddy, Mark D. Erion, Atul Agarwal, Vellarkad N. Viswanadhan,
D. Quentin McDonald, and W. Clark Still*

(Continued on next page)

Visit Wiley Journals Online
WILEY

InterScience

www.interscience.wiley.com

(Continued)

Method of Calculating Band Shape for Molecular Electronic Spectra <i>Greg M. Pearl, M. C. Zerner, Anders Broo, and John McKelvey</i>	781
Neighbor-List Reduction: Optimization for Computation of Molecular van der Waals and Solvent-Accessible Surface Areas <i>Jörg Weiser, Armin A. Weiser, Peter S. Shenkin, and W. Clark Still</i>	797
Erratum	809

Development of a Parallel Molecular Dynamics Code on SIMD Computers: Algorithm for Use of Pair List Criterion

D. ROCCATANO,¹ R. BIZZARRI,^{2,3} G. CHILLEMI,² N. SANNA,²
A. DI NOLA¹

¹Dipartimento di Chimica, Università "La Sapienza," Ple Aldo Moro 5, 00185 Rome, Italy

²CASPUR Consorzio per le Applicazioni di Supercalcolo per Università e Ricerca, c/o Università "La Sapienza," Rome, Italy

³Dipartimento di Fisica, Università "La Sapienza," Rome, Italy

Received 24 April 1996; accepted 24 October 1997



ABSTRACT: In recent years several implementations of molecular dynamics (MD) codes have been reported on multiple instruction multiple data (MIMD) machines. However, very few implementations of MD codes on single instruction multiple data (SIMD) machines have been reported. The difficulty in using pair lists of nonbonded interactions is the major problem with MD codes for SIMD machines, such that, generally, the full connectivity computation has been used. We present an algorithm, the global cut-off algorithm (GCA), which permits the use of pair lists on SIMD machines. GCA is based on a probabilistic approach and requires the cut-off condition to be simultaneously verified on all nodes of the machine. The MD code used was taken from the GROMOS package; only the routines involved in the pair lists and in the computation of nonbonded interactions were rewritten for a parallel architecture. The remaining calculations were performed on the host computer. The algorithm has been tested on Quadrics computers for configurations of 32, 128, and 512 processors and for systems of 4000, 8000, 15,000, and 30,000 particles. Quadrics was developed by Istituto Nazionale di Fisica Nucleare (INFN) and marketed by Alenia Spazio. © 1998 John Wiley & Sons, Inc. *J Comput Chem* 19: 685–694, 1998

Keywords: molecular dynamics; domain decomposition algorithm; parallel computers; pair list for molecular dynamics code on SIMD machines; array processor elaborator (APE)

Correspondence to: A. Di Nola

Contract/grant sponsors: Ente per le Nuove Tecnologie, l'Energia e l'Ambiente; Alenia Spazio

Journal of Computational Chemistry, Vol. 19, No. 7, 685–694 (1998)
© 1998 John Wiley & Sons, Inc.

CCC 0192-8651 / 98 / 070685-10

Introduction

Classical molecular dynamics (MD) is used to study the properties of liquids, solids, and molecules.^{1,2} The Newton equation of motion for each particle of the system is solved by numerical integration and its trajectory is obtained. From this microscopic point of view, many microscopic and macroscopic properties can be obtained. The need for numerical integration limits the time step to the femtosecond scale and makes MD simulation a very time consuming task. Therefore, considerable efforts have been concentrated on optimizing MD codes on parallel computers of different architectures.

Parallel computers are frequently described as belonging to one of two types: single instruction stream multiple data stream (SIMD), or multiple instruction stream multiple data stream (MIMD). In general, SIMD machines have a simpler architecture, but they have hardware limitations because the same instruction is executed in parallel on every SIMD processor and, furthermore, some SIMD machines do not have local addressing; that is, the processors are not able to access their own memory using different local addresses. In recent years, several MD codes have been implemented on MIMD architectures with a few dozen of processors³⁻⁵ and, more recently, also on 100- to 1000-processor MIMD machines.⁶⁻⁸ Parallel implementations of biological MD programs such as CHARMM⁹ and GROMOS¹⁰ on MIMD machines have been discussed in the literature.^{8,11-13}

Less work has been done using SIMD systems.¹⁴⁻¹⁷ In general, they make use of the full connectivity computation; that is, all atom pair interactions are calculated, and are useful for long-range force systems. This is due to the difficulty of using pair lists of nonbonded atoms on SIMD machines with no local addressing.

In the present study we propose an algorithm that permits the use of pair lists in a MD code for a SIMD machine with no local addressing. The algorithm requires simultaneous use of multiple time step¹⁸ and geometric decomposition¹³ methods. In addition, the systolic loop¹⁶ method is used to further reduce computation time.

The method was tested on Quadrics computers,¹⁹⁻²¹ a class of SIMD machines developed by INFN and Alenia Spazio, for configurations of 32, 128, and 512 processors. Quadrics is the only

massive parallel computer developed with fully European technology. As the Europort2/PACC project¹¹ has shown, the scalability for a MD code on MIMD architecture, for complex systems such as a protein in solution, is generally satisfactory only up to 12-16 nodes.

Moreover, there are interesting projects being undertaken on mixed architecture MIMD/SIMD machines that could supply the computational power of a SIMD machine, together with the flexibility of a MIMD. It is therefore worthwhile to determine whether these machines are able to perform such calculations.

The following molecular systems have been used as tests:

- *System 1:* Box of 1536 water molecules (4608 atoms).
- *System 2:* Box with a BPTI (bovine pancreatic trypsin inhibitor) molecule and 2712 water molecules (8704 atoms).
- *System 3:* Box with a SOD (superoxide dismutase) dimer and 4226 water molecules (15,360 atoms).
- *System 4:* Box with a SOD (superoxide dismutase) dimer and 9346 water molecules (30,720 atoms).

It should be noted that system 4 is nearly the same as test case I3, used as the industrial benchmark in the framework of the Europort2/PACC project¹¹ (system 4 has 9346 water molecules whereas test case I3 has 9436 water molecules). The results show that the speed-up of the algorithm is comparable to those obtained with MIMD machines.

Hardware

We tested the method on a Quadrics machine, Alenia Spazio's supercomputer derived from the APE100 (Array Processor Elaborator) project, developed by INFN.¹⁹⁻²¹ These computers are a family of massively parallel SIMD machines with implementations from 8 to 2048 processors. The biggest implementation allows a peak computing power of 100 GFlops in single precision (32-bit processors).

The processors are arranged in a three-dimensional (3D) cubic mesh and can exchange data with the six neighboring nodes, with periodic boundaries. Each processor board contains eight processors (floating point units) with their own

memory (4 megabytes). Up to 16 boards can be put into a crate. Configurations with more than 128 processors are made up connecting crates of $8 \times 4 \times 4$ (128) nodes.

The Quadrics controller board contains one integer CPU (Z-CPU), which controls the flux of the program and the integer arithmetic. The language used is called Tao, a Fortran-like high-level parallel language, which can be modified and extended through a dynamic ZZ parser. The Quadrics machine is connected to a host computer (a Sun Sparc-10 or -20). A host interface based on a HIPPI standard, which allows an I/O speed between the host and Quadrics of 20 MB/s, has recently been developed. The tests on the sequential machine have been run on a DEC-alpha 3000/500 machine. Barone et al.²² compared the accuracy of Quadrics in the field of molecular dynamics with that of a conventional computer to assess the limits of the single precision.

Molecular Dynamics

In a molecular dynamics simulation, the classical equations of motion for the system of interest are integrated numerically by solving Newton's equations of motion:

$$m_i \frac{d^2 r_i}{dt^2} = -\nabla_i V(r_1, r_2, \dots, r_N)$$

The solution gives the atomic positions and velocities as a function of time. The knowledge of the trajectory of each atom permits study of the dynamic or statistical properties of the system. The form of the interaction potential is complex and it includes energy terms that represent bonded and nonbonded (van der Waals and Coulombic) interactions:

$$\begin{aligned} V(r_1, r_2, \dots, r_N) &= \sum_{\text{bonds}} \frac{1}{2} K_b [b - b_0]^2 + \sum_{\text{angles}} \frac{1}{2} K_\theta [\theta - \theta_0]^2 \\ &+ \sum_{\text{improper}} \frac{1}{2} K_\xi [\xi - \xi_0]^2 \\ &+ \sum_{\text{dihedrals}} K_\varphi [1 + \cos(n\varphi - \delta)] \\ &+ \sum_{\text{pairs}(i,j)} \left[4\epsilon_{ij} \left(\frac{\sigma_{ij}^{12}}{r_{ij}^{12}} - \frac{\sigma_{ij}^6}{r_{ij}^6} \right) + \frac{q_i q_j}{4\pi\epsilon r_{ij}} \right] \end{aligned}$$

The first four terms represent the bonded potential. b , b_0 , and K_b are the actual bond length, its reference value, and the bond stretching force constant, respectively. θ , θ_0 , and K_θ are the actual bond angle, its reference value, and the angle bending force constant, respectively. ξ , ξ_0 , and K_ξ are the actual improper dihedral angle, its reference value, and the improper dihedral angle bending force constant, respectively. φ , K_φ , n , and δ are the actual dihedral angle, its force constant, the multiplicity, and the phase, respectively. The last term in the equation includes the nonbonded, van der Waals, and Coulombic terms. ϵ_{ij} and σ_{ij} are the dispersion well depth and the Lennard-Jones distance, q_i and q_j are the electrostatic atomic charges, r_{ij} is the distance between them, and ϵ is the dielectric constant.

The time step used for the numerical integration is in the femtosecond scale. The highest frequency of bond vibrations would require a time step ≤ 0.5 fs; however, if the simulation is performed with constant bond lengths, the time step can be ≤ 2 fs. For this reason, many MD codes perform simulations with constant bond lengths.

The most frequently used algorithm to perform MD simulation at constant bond lengths is the SHAKE algorithm based on an iterative procedure.²³

Computational Algorithm for Nonbonded Interactions

In a MD program, the calculation of the nonbonded forces is the most time-consuming task—in fact, it takes about 90% of the computational time, depending on the protocol used.

One of the most frequently used techniques to reduce the number of nonbonded forces is the cut-off criterion. With this method the interactions between atoms beyond a cut-off distance are neglected. If the cut-off radius is appropriate the lost energy contribution to the global potential is small. During a small number of steps the pairs of interacting atoms are considered to remain the same so that it is possible to create a list of these interactions, the nonbonded pair list, which will be updated every n steps (n is generally equal to 10). The number of nonbonded interactions is $N(N-1)/2$ (N is the number of atoms), so that it is proportional to N^2 . The use of the cut-off criterion reduces this number to kN (k is a constant).

Unfortunately, the cut-off criterion is not directly applicable to SIMD architecture, as the same instruction is executed at each time on each processor and, consequently, it is not possible to have a local branch (the cut-off condition) in the program flow. Moreover, on Quadrics it is not possible to have a local pair list on each node because local addressing of arrays is not possible. This explains the lower level of efficiency of a MD code on a SIMD machine with respect to a MIMD one. We have recently developed an algorithm, the global cut-off algorithm (CGA), based on a probabilistic approach, which allows the use of the cut-off condition on a SIMD machine with no local addressing.

Because the calculation of bonded interactions and the integration of the trajectory take a small amount of the total calculation time, in the present version we have chosen to carry them out on the front-end computer and to perform only the calculations of the pair list and of the nonbonded forces using the SIMD machine. It is, of course, possible to perform all force calculations and integration in the parallel machine using, for instance, the replicated data method.⁸

GEOMETRIC DECOMPOSITION

The assignment of the atoms to the nodes is obtained by a dynamically geometric decomposition¹³ in such a way that the same number of atoms is assigned to each node. In what follows, we discuss a decomposition for a bidimensional case; the extension to a third dimension is straightforward: given the bidimensional box of Figure 1a and a 2D parallel topology of $n = n_x \times n_y$ processors, with $n_x = n_y = 2$, the box is first divided into n_x boxes along the x -axis, as shown in Figure 1b, each containing the same number of atoms. Each box is successively divided into n_y boxes along the y -axis in such a way that each one of the $n_x \times n_y$ boxes contains the same number of atoms (Fig. 1c). When, as in a real case, a third dimension exists, a successive division along the z -axis has to be performed.

It is obvious that, before performing any division along a given axis, it is necessary to sort the atoms of each box along that axis.

The density of a molecular system, such as a protein, is not uniform; thus, the boxes do not have the same axis lengths. However, these differences do not significantly reduce the efficiency of the GCA described in what follows.

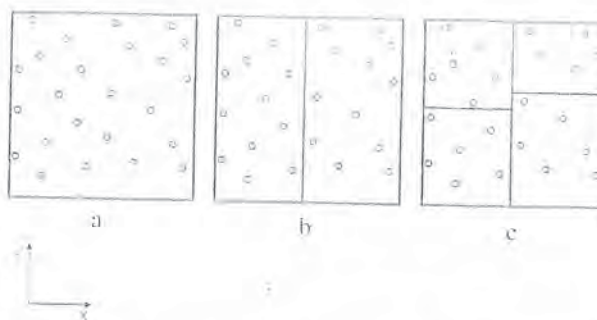


FIGURE 1. Domain decomposition of the molecular system in boxes with the same number of atoms, for a bidimensional case.

SYSTOLIC LOOP METHOD

Quadrics topology makes it possible to use a systolic loop to calculate the nonbonded interactions between the atoms assigned to the different nodes. The systolic loop method is one of the most efficient algorithms for calculation of two-body interactions on MIMD and SIMD machines.^{14, 16, 24, 25} The systolic loop algorithm passes the coordinates of all atoms around a ring of P processors in $P/2$ steps, such that half of the coordinates passes every processor exactly once (transient atoms). Each node also stores the coordinates of a group of atoms of the overall system (resident atoms). During the systolic cycle each processor evaluates and accumulates the interactions of the resident atoms with the transient ones. Only half of the atoms have to pass in each computational node as a consequence of the reciprocity of the interactions.

The systolic loop path for a 32-node Quadrics machine is shown in Figure 2. This machine has two nodes along the y and z directions and eight along the x direction.

The geometric decomposition of the system permits limitation of the search for nonbonded interactions only to the neighboring processors nearer than the cut-off radius, so that, depending on the number of nodes and on the system size, it is generally not necessary to perform the complete systolic loop. The computed forces are passed back to the owning processor to accumulate the full force.

GLOBAL CUT-OFF ALGORITHM

On a SIMD machine, all nodes simultaneously evaluate an interaction, but the atom pairs in each node are different. Figure 3 shows, as an example,

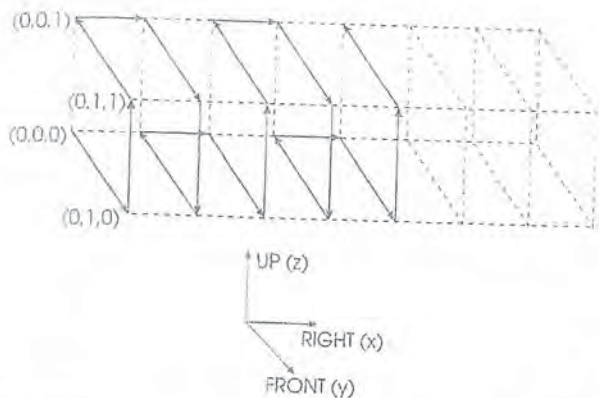


FIGURE 2. Systolic loop path for node (0, 0, 0) of a 32-node Quadrics machine. The transient groups of atoms visit only four neighboring $y-z$ planes, based on Newton's third law.

the case with four nodes: suppose that each node is evaluating the interaction a_i ; in this case, all a_i interactions fall within the cut-off radius. When the interactions are of the b_i type all the distances fall outside the cut-off radius and the interactions b_i are skipped. In the case of interactions of type c_i the interaction is outside the cut-off radius in nodes 1, 2, and 3, but it is inside the cut-off radius in

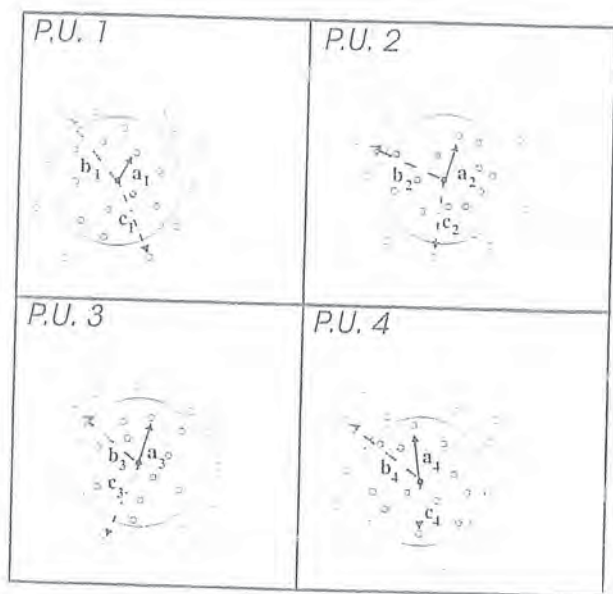


FIGURE 3. Different types of interactions in a case of four nodes. a_i , all the interactions fall within the cut-off distance; b_i , all the interactions fall outside the cut-off distance; c_i , one interaction falls within the cut-off, whereas three fall outside the cut-off. P.U. = processor unit.

node 4, so that all nodes have to calculate this interaction and only will be saved in the forces calculation. If the atoms in each node are ordered randomly, the interactions of type c_i result in being the most frequent.

The basis idea of the global cut-off algorithm (GCA) is to maximize the occurrence of interactions of type a_i and b_i and, conversely, reduce the occurrence of interactions of type c_i . To this end, it is necessary that the atoms in all nodes are sorted with the same criterion. Different types of sorting give comparable results. We have chosen the one shown in Figure 4. After this sorting procedure, a list of the interactions of type a_i and c_i is created in the integer CPU (Z-CPU) of the SIMD machine. This list is equivalent, but not identical, to the nonbonded pair list used in most MD programs and will be referred as the *nonbonded pair list*.

The ordering procedures for the domain decomposition and the sorting procedure previously described are time consuming and have to be performed on the host serial machine; however, as will be shown, they have to be performed every 100 to 200 steps so that they do not significantly affect the global computation time.

The global cut-off condition is based on a probabilistic approach, so that the number of pair interactions to be calculated is larger than the actual number of pairs included within the r_{cut} distance. Depending on the molecular system and on the number of nodes, the ratio between the number of the calculated interactions and the number of interactions actually included within the cut-off dis-

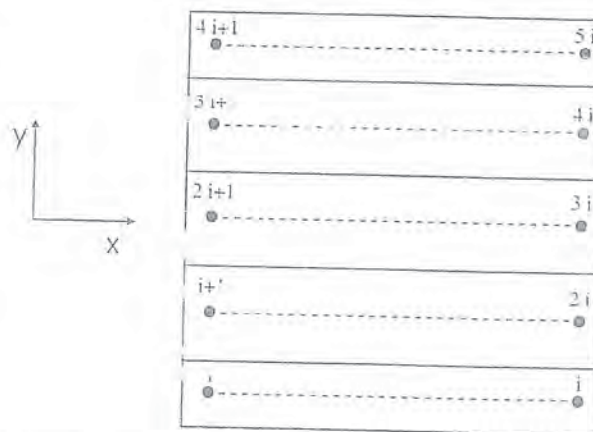


FIGURE 4. Sorting of atoms in each node for a bidimensional case. The atoms are represented as full circles.

tance varies from the three to five. In this sense, the GCA is not very efficient. However, it must be noted that almost all pair interactions within a distance $r_{\text{cut}} < r \leq r_{\text{cut}} + \Delta r$, with $\Delta r \cong 0.2$ nm, are calculated. As an example, given $r_{\text{cut}} = 0.6$ nm, 94% of the interactions in the range $0.6 < r \leq 0.8$ nm are calculated and only 6% of pairs in this range are lost. This suggests adoption of a cut-off value of r_{GCA} to be used in the global cut-off condition, somewhat shorter than the actual cut-off, r_{cut} , desired for the simulation. In the previous example, with $r_{\text{GCA}} = 0.6$ and $r_{\text{cut}} = 0.8$ nm, the number of pairs to be calculated is roughly two-to-threefold larger than the actual number of pairs within the r_{cut} distance. The remaining 6% of interactions in the range $0.6 < r \leq 0.8$ nm have to be calculated separately.

It is well known that nonbonded forces vary more slowly than the bonded ones. Moreover, nonbonded forces at large distances vary slower than nonbonded forces at short distances. This suggests updating of the forces at different steps, according to their nature (bonded and nonbonded) and to the distance of the interaction. The short-range interactions can be evaluated every step, and long-range interactions every m steps. Accordingly, the few interactions in the range $0.6 < r \leq 0.8$ nm that were lost using $r_{\text{GCA}} = 0.6$ nm can be updated every m steps. As these interactions are calculated while evaluating the *nonbonded pair list* (i.e., updated every n steps), we have chosen $m = n = 10$.

It must be noted that there are now two shells: an inner shell ($r \leq 0.6$ nm) and an outer shell ($0.6 < r \leq 0.8$ nm). All interactions are evaluated every m steps, whereas only those interactions corresponding to the inner shell are evaluated every step. It is therefore not necessary to have a skin distance and to store a list of atom pairs outside the outer shell.

In the present study it is seen that most interactions in the range 0.6 to 0.8 nm are evaluated every step and only a few of them are evaluated every m steps. According to all of the MTS algorithms, this choice does not affect the numerical accuracy; in fact, the same accuracy is obtained when an interaction, within the outer shell, is evaluated every short time step or every long time step. However, as every long time step all interactions within the outer shell are evaluated, it is possible to perform the MTS according to the classical procedure; that is, by collection all the interactions within the outer shell at every long time step and collecting only the interactions within the inner shell at every short time step. Among several algorithms proposed for the multiple time step (MTS)^{1, 18, 26} we have chosen the one developed by Scully and Hermans.¹⁸

It must be noted that all nonbonded interactions (van der Waals and Coulombic) between bonded and nonbonded atoms are calculated in this step, but the interactions between bonded atoms are not saved. This is obtained by attaching to each atom a list of the atoms bonded to itself. This procedure is certainly not efficient, but the time required to perform it is negligible. In the following tests, the values of r_{GCA} and r_{cut} are fixed at 0.6 and 0.8 nm, respectively.

Results

Table I shows the number of interactions within the cut-off radius r_{cut} compared with the number of interactions to be evaluated with the GCA. It can be noted that the number of interactions for calculation is two to three times the actual number of interactions within the cut-off radius. The time

TABLE I. Number of Actual Interactions, N , within a Cut-Off Radius ($r_{\text{cut}} = 0.8$ nm) Compared with the Number of Interactions Calculated Using GCA on 32-, 128- and 512-Node Quadrics Machines.

	N	32 nodes	128 nodes	512 nodes
System 1 (4608 atoms)	488,847	706,944	840,320	873,984
System 2 (8704 atoms)	891,252	1,612,864	1,810,944	2,070,016
System 3 (15,360 atoms)	1,487,054	3,153,344	3,768,576	4,452,864
System 4 (30,720 atoms)	3,129,972	6,754,240	8,249,216	9,938,944

required for performing the MD simulation with the present code is the sum of the following steps:

1. Ordering procedure (performed every k steps by the host computer).
2. Calculation of the *nonbonded pair list* (performed every n steps by Quadrics).
3. Calculation of the nonbonded forces (performed every step by Quadrics).
- 3a. Calculation of the bonded forces by the host computer while performing step 3.
4. I/O host \leftrightarrow Quadrics.
5. SHAKE and integration (performed by the host).

The ordering procedure is a time-consuming task and, due to the diffusion of the system, it has to be periodically repeated every k steps. If no reordering is performed the *nonbonded pair list* will include an increasing number of interactions, thus increasing the computational time. Figure 5 shows

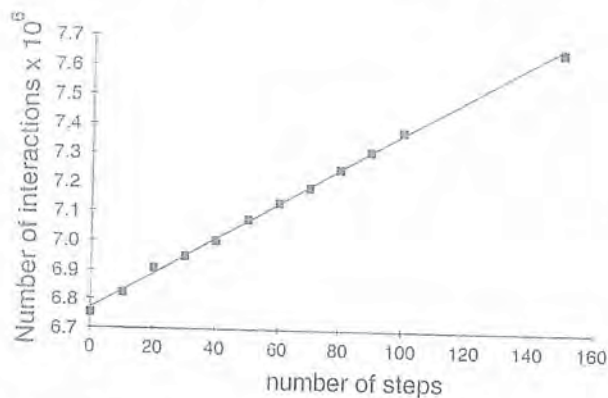


FIGURE 5. Number of interactions to be calculated versus the number of steps for system 4 on a 32-node machine. The atoms are not reordered during the simulation.

TABLE II. cpu Times for Nonbonded Interactions (Expressed in Milliseconds per Step).

	DEC-alpha 3000 / 500	32 nodes	128 nodes	512 nodes
System 1 (4608 atoms)	2943	632	174	68
System 2 (8704 atoms)	6097	1650	386	107
System 3 (15,360 atoms)	10,928	3274	894	214
System 4 (30,720 atoms)	27,610	6208	1624	466

the number of interactions to be evaluated versus the number of steps when the ordering procedure is performed at the beginning and not updated, for system 4 on a 32-node machine.

The loss of performance is nearly linear, being $\sim 0.08\%$ per step. The optimum k value depends on the time required for the ordering procedure and on the time required for items 2 and 3. It shows that, for all the systems and all the different numbers of nodes, the optimum k value is in the range of 100 to 200 steps. The ordering procedure for system 4 on a Sun Sparc-20 (the host computer) required 20 seconds, so that its cpu time per step is in the range of 100 to 200 ms.

The *nonbonded pair list* is evaluated every n steps ($n = 10$ in the present case) and the nonbonded interactions are evaluated every step. The average cpu time required for these tasks is reported in Table II for different systems and different numbers of nodes. It should be emphasized that the parallel machines perform the calculation on a number of pairs two to three times larger than the serial one. The almost linear scalability of these task is also worthy of note.

The data transferred from the host to the Quadrics and vice versa after each reordering step (i.e., every $k \sim 100$ steps) are reported in Table III (upper panel); that is, 23 words per atom.

The data to be transferred every step are reported in Table III (lower panel)—9 words per atom. The average time spent in transmission depends on the speed of information transfer. Taking into account the speed of the HIPPI interface (20 MB/s) the average I/O times per step required for systems 1 to 4 are 9, 16, 28, and 55 ms/step, respectively.

The sums of times for items 1 to 4 with different numbers of nodes are reported in Table IV. Figure 6 shows the number of steps per second versus the

TABLE III.
Data Transfer Results.

	Quantity transferred (words per atom)
Data transferred every <i>k</i> steps	
From host to Quadrics	
Coordinates of the atom	3
Coordinates of the geometric center of the charge group	3
Electric Charge	1
Sequential atomic number	1
van der Waals parameters	2
Exclusions 1-3	6
Exclusions 1-4	4
From Quadrics to host	
Forces	3
Total	23
Data transferred every step	
From host to Quadrics	
Coordinates of the atom	3
Coordinates of the geometric center of the charge group	3
From Quadrics to host	
Forces	3
Total	9

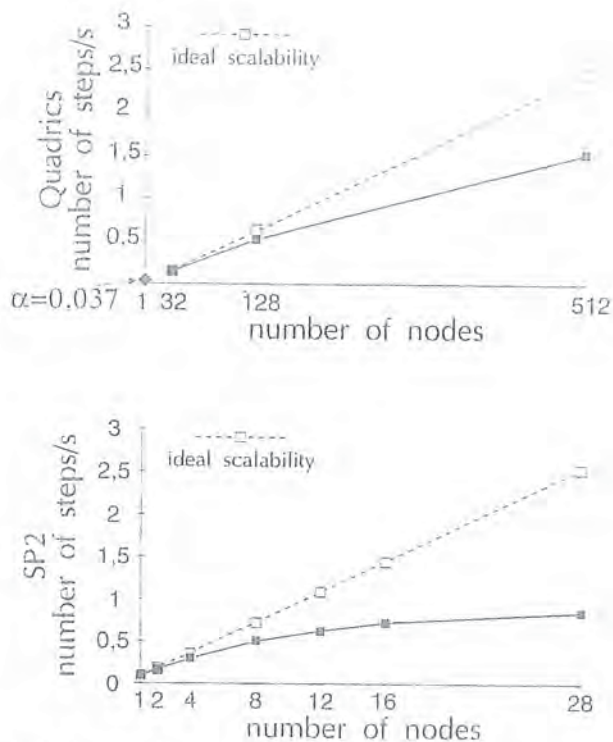


FIGURE 6. Number of steps per second versus the number of nodes for system 4. (a) Quadrics; (b) IBM SP2. The ideal scalability is expressed as a dashed line. Single processor DEC-alpha 3000 / 500 timing is shown for comparison.

number of nodes for Quadrics (Fig. 6a) and for an IBM-SP2 MIMD machine (Fig. 6b) for the same system (system 4). For clarity, the CPU times required for SHAKE and integration are not included. The code used for the MIMD machine was developed within the Europort2/PACC project.¹¹

Figure 6 also shows that no significant advantage is obtained with the MIMD machine when the number of nodes is ≥ 12 , whereas, a good scalability, up to 512 nodes, is obtained with the SIMD machine.

TABLE IV.
Total cpu Times Including Nonbonded Interactions, Ordering Procedure, and I/O Host Quadrics (Milliseconds per Step).

	32 nodes	128 nodes	512 nodes
System 1 (4608 atoms)	680	205	90
System 2 (8704 atoms)	1790	470	175
System 3 (15,360 atoms)	3540	1050	335
System 4 (30,720 atoms)	6660	1910	685

ability, up to 512 nodes, is obtained with the SIMD machine.

To complete the evaluation of the total time/steps of the present MD code, the times required on the host for the bonded interactions, SHAKE, and integration have to be calculated. The times required for these tasks, with the present MD code, are reported in Table V. It should be noted that the calculation of bonded interactions is performed by the host, whereas Quadrics computes the nonbonded interactions. As the former's calculation time is less than the latter, this task does not require any extra cpu time.

The integration task requires less cpu time than the nonbonded interaction calculation time (see Table IV) on a 32- or 128-node machine, and a comparable amount of time on a 512-node machine. Therefore, this task must be parallelized for machines with hundreds or thousands of processors. The integration can be implemented easily on a SIMD machine by, for example, the replicated data procedure.

The cpu time required to perform the SHAKE algorithm on the host is the actual bottleneck. It is

TABLE V.
cpu time on the Host (Sun Sparc-20) for Calculation of Bonded Interactions, SHAKE, and Integration (expressed in Milliseconds per Step).

	Bonded Interactions	SHAKE	Integration
System 1 (4608 atoms)	41	236	57
System 2 (8704 atoms)	96	488	117
System 3 (15,360 atoms)	73	1102	161
System 4 (30,720 atoms)	194	3013	437

very difficult to implement the SHAKE algorithm on a SIMD machine; therefore, an alternative procedure must be chosen. The MTS procedure can be used to evaluate the bond vibrations without reducing the time steps required for the nonbonded interactions.

Conclusions

The results reported in the present work show that the GCA permits use of pair lists even on a SIMD machine with no local addressing, thus overcoming one of the most severe disadvantages of SIMD vs. MIMD machines. The penalty to be paid is the number of interactions per step to be calculated; that is, two to three times the actual number of interactions.

The tests performed on Quadrics computers for configurations of 32, 128, and 512 nodes, for systems of different sizes, up to 30,000 particles, show that the scalabilities and performances are satisfactory and comparable to those obtained with MIMD machines. At the present time, the only routines for the calculation of the pair lists and nonbonded interactions have been parallelized. We have shown that the bonded forces can be calculated by the host while the parallel machine calculates the nonbonded ones. Also, the integration task can be calculated by the host if the parallel machine has up to tens of processors. With hundreds or thousands of processors this task must also be parallelized.

The SHAKE algorithm, which allows one to perform MD calculations at constant bond lengths, is the actual bottleneck of the calculation and its implementation is difficult with the parallel machine. We suggest the use of the MTS to evaluate the bond vibration contributions.

Acknowledgments

The authors thank ENEA for the use of the 128 and 512 Quadrics machines. One of us (R.B.) thanks ENEA for his hospitality during the realization of this work.

References

1. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Clarendon Press, Oxford, 1987.
2. W. F. van Gunsteren and H. J. C. Berendsen, *Angew. Chem. Int. Ed. Engl.*, **29**, 992 (1990).
3. M. R. S. Pinches, D. J. Tildesley, and W. Smith, *Mol. Simul.*, **6**, 51 (1991).
4. H. Heller, H. Grubmüller, and K. Shulten, *Mol. Simul.*, **5**, 133 (1990).
5. D. C. Rapaport, *Comput. Phys. Commun.*, **62**, 217 (1991).
6. P. A. J. Hilbers and K. Esselink, *Computer Simulation in Chemical Physics*, M. P. Allen and D. J. Tildesley, Eds., Kluwer Academic Publisher, London, 1993, p. 473.
7. S. P. Plimpton, *J. Comput. Phys.*, **117**, 1 (1995).
8. S. P. Plimpton and B. Hendrickson, *J. Comput. Chem.*, **17**, 326 (1996).
9. B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, *J. Comput. Chem.*, **4**, 187 (1983).
10. W. F. van Gunsteren and H. J. C. Berendsen, *GROMOS: Groningen Molecular Simulation (GROMOS) Library Manual*, Biomos, Groningen, 1987.
11. D. G. Green, K. E. Meacham, M. Surridge, F. van Hoesel, and H. J. C. Berendsen, *Methods and Techniques in Computational Chemistry: Metecc-95*, E. Clementi and G. Corongiti, Eds., STEF, Cagliari, 1995, p. 435.
12. S. L. Lin, J. Mellor-Crummey, B. M. Pettit, and G. N. Phillips, *J. Comput. Chem.*, **13**, 1022 (1992).
13. W. T. Clark, R. Hanxleden, J. A. McCammon, and L. R. Scott, *Technical Report CRPC-TR93356-S*, Center for Research on Parallel Computation, Houston, TX, November 1993.
14. A. Windemuth and K. Shulten, *Mol. Simul.*, **5**, 353 (1991).
15. D. Fincham, *Mol. Simul.*, **1**, 1 (1987).
16. K. M. Nelson, C. F. Cornwell, and L. T. Wille, *Comput. Mater. Sci.*, **2**, 525 (1994).
17. P. Ballestrero, P. Baglietto and C. Ruggiero, *J. Comput. Chem.*, **17**, 469 (1996).

18. J. L. Scully and J. Hermans, *Mol. Simul.*, **11**, 67 (1993).
19. A. Bartoloni, G. Bastianello, C. Battista, S. Cabasino, F. Marzano, P. S. Paolucci, J. Pech, F. Rapuano, E. Panizzi, R. Sarno, G. M. Todesco, M. Torelli, W. Tross, P. Vicini, N. Cabibbo, A. Fucci, and R. Tripicciono, *Int. J. Modern Phys.*, **4**, 969 (1993).
20. A. Bartoloni, G. Bastianello, C. Battista, S. Cabasino, F. Marzano, P. S. Paolucci, J. Pech, F. Rapuano, E. Panizzi, R. Sarno, G. M. Todesco, M. Torelli, W. Tross, P. Vicini, N. Cabibbo, and R. Tripicciono, *Int. J. Modern Phys.*, **4**, 955 (1993).
21. The APE Collaboration, *Int. J. High Speed Comput.*, **5**, 637 (1993).
22. L. M. Barone, R. Simonazzi, and A. Tenenbaum, *Comput. Phys. Commun.*, **90**, 44 (1995).
23. J. P. Ryckaert, G. Ciccotti, H. J. C. Berendsen, *J. Comput. Phys.*, **23**, 327 (1977).
24. W. Smith, *Comput. Phys. Commun.*, **62**, 229 (1991).
25. J. F. Janak and P. C. Pattnaik, *J. Comput. Chem.*, **13**, 533 (1992).
26. M. Tuckerman, B. J. Berne, and G. J. Martyna, *J. Chem. Phys.*, **97**, 1990 (1992).

Attachment 4B

Development of a Parallel Molecular Dynamics Code on SIMD Computers: Algorithm for Use of Pair List Criterion

D. ROCCATANO,¹ R. BIZZARRI,^{2,3} G. CHILLEMI,² N. SANNA,²
A. DI NOLA¹

¹Dipartimento di Chimica, Università "La Sapienza," Ple Aldo Moro 5, 00185 Rome, Italy

²CASPUR Consorzio per le Applicazioni di Supercalcolo per Università e Ricerca, c/o Università "La Sapienza," Rome, Italy

³Dipartimento di Fisica, Università "La Sapienza," Rome, Italy

Received 24 April 1996; accepted 24 October 1997

ABSTRACT: In recent years several implementations of molecular dynamics (MD) codes have been reported on multiple instruction multiple data (MIMD) machines. However, very few implementations of MD codes on single instruction multiple data (SIMD) machines have been reported. The difficulty in using pair lists of nonbonded interactions is the major problem with MD codes for SIMD machines, such that, generally, the full connectivity computation has been used. We present an algorithm, the global cut-off algorithm (GCA), which permits the use of pair lists on SIMD machines. GCA is based on a probabilistic approach and requires the cut-off condition to be simultaneously verified on all nodes of the machine. The MD code used was taken from the GROMOS package; only the routines involved in the pair lists and in the computation of nonbonded interactions were rewritten for a parallel architecture. The remaining calculations were performed on the host computer. The algorithm has been tested on Quadrics computers for configurations of 32, 128, and 512 processors and for systems of 4000, 8000, 15,000, and 30,000 particles. Quadrics was developed by Istituto Nazionale di Fisica Nucleare (INFN) and marketed by Alenia Spazio. © 1998 John Wiley & Sons, Inc. *J Comput Chem* 19: 685–694, 1998

Keywords: molecular dynamics; domain decomposition algorithm; parallel computers; pair list for molecular dynamics code on SIMD machines; array processor elaborator (APE)

Correspondence to: A. Di Nola

Contract grant sponsors: Ente per le Nuove Tecnologie, l'Energia e l'Ambiente; Alenia Spazio

Journal of Computational Chemistry, Vol. 19, No. 7, 685–694 (1998)
© 1998 John Wiley & Sons, Inc.

CCC 0192-8651/98/070685-10

Introduction

Classical molecular dynamics (MD) is used to study the properties of liquids, solids, and molecules.^{1,2} The Newton equation of motion for each particle of the system is solved by numerical integration and its trajectory is obtained. From this microscopic point of view, many microscopic and macroscopic properties can be obtained. The need for numerical integration limits the time step to the femtosecond scale and makes MD simulation a very time consuming task. Therefore, considerable efforts have been concentrated on optimizing MD codes on parallel computers of different architectures.

Parallel computers are frequently described as belonging to one of two types: single instruction stream multiple data stream (SIMD), or multiple instruction stream multiple data stream (MIMD). In general, SIMD machines have a simpler architecture, but they have hardware limitations because the same instruction is executed in parallel on every SIMD processor and, furthermore, some SIMD machines do not have local addressing; that is, the processors are not able to access their own memory using different local addresses. In recent years, several MD codes have been implemented on MIMD architectures with a few dozen of processors³⁻⁵ and, more recently, also on 100- to 1000-processor MIMD machines.⁶⁻⁸ Parallel implementations of biological MD programs such as CHARMM⁹ and GROMOS¹⁰ on MIMD machines have been discussed in the literature.^{8,11-13}

Less work has been done using SIMD systems.¹⁴⁻¹⁷ In general, they make use of the full connectivity computation; that is, all atom pair interactions are calculated, and are useful for long-range force systems. This is due to the difficulty of using pair lists of nonbonded atoms on SIMD machines with no local addressing.

In the present study we propose an algorithm that permits the use of pair lists in a MD code for a SIMD machine with no local addressing. The algorithm requires simultaneous use of multiple time step¹⁸ and geometric decomposition¹³ methods. In addition, the systolic loop¹⁶ method is used to further reduce computation time.

The method was tested on Quadrics computers,¹⁹⁻²¹ a class of SIMD machines developed by INFN and Alenia Spazio, for configurations of 32, 128, and 512 processors. Quadrics is the only

massive parallel computer developed with fully European technology. As the Europort2 PACC project¹¹ has shown, the scalability for a MD code on MIMD architecture, for complex systems such as a protein in solution, is generally satisfactory only up to 12-16 nodes.

Moreover, there are interesting projects being undertaken on mixed architecture MIMD SIMD machines that could supply the computational power of a SIMD machine, together with the flexibility of a MIMD. It is therefore worthwhile to determine whether these machines are able to perform such calculations.

The following molecular systems have been used as tests:

System 1: Box of 1536 water molecules (4608 atoms).

System 2: Box with a BPTI (bovine pancreatic trypsin inhibitor) molecule and 2712 water molecules (8704 atoms).

System 3: Box with a SOD (superoxide dismutase) dimer and 4226 water molecules (15,360 atoms).

System 4: Box with a SOD (superoxide dismutase) dimer and 9346 water molecules (30,720 atoms).

It should be noted that system 4 is nearly the same as test case I3, used as the industrial benchmark in the framework of the Europort2 PACC project¹¹ (system 4 has 9346 water molecules whereas test case I3 has 9436 water molecules). The results show that the speed-up of the algorithm is comparable to those obtained with MIMD machines.

Hardware

We tested the method on a Quadrics machine, Alenia Spazio's supercomputer derived from the APE100 (Array Processor Elaborator) project, developed by INFN.¹⁹⁻²¹ These computers are a family of massively parallel SIMD machines with implementations from 8 to 2048 processors. The biggest implementation allows a peak computing power of 100 GFlops in single precision (32-bit processors).

The processors are arranged in a three-dimensional (3D) cubic mesh and can exchange data with the six neighboring nodes, with periodic boundaries. Each processor board contains eight processors (floating point units) with their own

memory (4 megabytes). Up to 16 boards can be put into a crate. Configurations with more than 128 processors are made up connecting crates of $8 \times 4 \times 4$ (128) nodes.

The Quadrics controller board contains one integer CPU (Z-CPU), which controls the flux of the program and the integer arithmetic. The language used is called Tao, a Fortran-like high-level parallel language, which can be modified and extended through a dynamic ZZ parser. The Quadrics machine is connected to a host computer (a Sun Sparc-10 or -20). A host interface based on a HIPPI standard, which allows an I/O speed between the host and Quadrics of 20 MB/s, has recently been developed. The tests on the sequential machine have been run on a DEC-alpha 3000 500 machine. Barone et al.²² compared the accuracy of Quadrics in the field of molecular dynamics with that of a conventional computer to assess the limits of the single precision.

Molecular Dynamics

In a molecular dynamics simulation, the classical equations of motion for the system of interest are integrated numerically by solving Newton's equations of motion:

$$m_i \frac{d^2 r_i}{dt^2} = -\nabla_i V(r_1, r_2, \dots, r_N)$$

The solution gives the atomic positions and velocities as a function of time. The knowledge of the trajectory of each atom permits study of the dynamic or statistical properties of the system. The form of the interaction potential is complex and it includes energy terms that represent bonded and nonbonded (van der Waals and Coulombic) interactions:

$$V(r_1, r_2, \dots, r_N) = \sum_{\text{bonds}} \frac{1}{2} K_b (b - b_0)^2 + \sum_{\text{angles}} \frac{1}{2} K_\theta (\theta - \theta_0)^2 + \sum_{\text{improper dihedrals}} \frac{1}{2} K_\xi (\xi - \xi_0)^2 + \sum_{\text{dihedrals}} K_\varphi [1 - \cos(n\varphi - \delta)] + \sum_{\text{pairs } (i, j)} \left[4\epsilon_{ij} \left(\frac{\sigma_{ij}^{12}}{r_{ij}^{12}} - \frac{\sigma_{ij}^6}{r_{ij}^6} \right) - \frac{q_i q_j}{4\pi\epsilon r_{ij}} \right]$$

The first four terms represent the bonded potential. b , b_0 , and K_b are the actual bond length, its reference value, and the bond stretching force constant, respectively. θ , θ_0 , and K_θ are the actual bond angle, its reference value, and the angle bending force constant, respectively. ξ , ξ_0 , and K_ξ are the actual improper dihedral angle, its reference value, and the improper dihedral angle bending force constant, respectively. φ , K_φ , n , and δ are the actual dihedral angle, its force constant, the multiplicity, and the phase, respectively. The last term in the equation includes the nonbonded, van der Waals, and Coulombic terms. ϵ_{ij} and σ_{ij} are the dispersion well depth and the Lennard-Jones distance, q_i and q_j are the electrostatic atomic charges, r_{ij} is the distance between them, and ϵ is the dielectric constant.

The time step used for the numerical integration is in the femtosecond scale. The highest frequency of bond vibrations would require a time step 0.5 fs; however, if the simulation is performed with constant bond lengths, the time step can be 2 fs. For this reason, many MD codes perform simulations with constant bond lengths.

The most frequently used algorithm to perform MD simulation at constant bond lengths is the SHAKE algorithm based on an iterative procedure.²³

Computational Algorithm for Nonbonded Interactions

In a MD program, the calculation of the nonbonded forces is the most time-consuming task—in fact, it takes about 90% of the computational time, depending on the protocol used.

One of the most frequently used techniques to reduce the number of nonbonded forces is the cut-off criterion. With this method the interactions between atoms beyond a cut-off distance are neglected. If the cut-off radius is appropriate the lost energy contribution to the global potential is small. During a small number of steps the pairs of interacting atoms are considered to remain the same so that it is possible to create a list of these interactions, the nonbonded pair list, which will be updated every n steps (n is generally equal to 10). The number of nonbonded interactions is $N(N-1)/2$ (N is the number of atoms), so that it is proportional to N^2 . The use of the cut-off criterion reduces this number to kN (k is a constant).

Unfortunately, the cut-off criterion is not directly applicable to SIMD architecture, as the same instruction is executed at each time on each processor and, consequently, it is not possible to have a local branch (the cut-off condition) in the program flow. Moreover, on Quadrics it is not possible to have a local pair list on each node because local addressing of arrays is not possible. This explains the lower level of efficiency of a MD code on a SIMD machine with respect to a MIMD one. We have recently developed an algorithm, the global cut-off algorithm (CGA), based on a probabilistic approach, which allows the use of the cut-off condition on a SIMD machine with no local addressing.

Because the calculation of bonded interactions and the integration of the trajectory take a small amount of the total calculation time, in the present version we have chosen to carry them out on the front-end computer and to perform only the calculations of the pair list and of the nonbonded forces using the SIMD machine. It is, of course, possible to perform all force calculations and integration in the parallel machine using, for instance, the replicated data method.⁸

GEOMETRIC DECOMPOSITION

The assignment of the atoms to the nodes is obtained by a dynamically geometric decomposition¹³ in such a way that the same number of atoms is assigned to each node. In what follows, we discuss a decomposition for a bidimensional case; the extension to a third dimension is straightforward: given the bidimensional box of Figure 1a and a 2D parallel topology of $n_x \times n_y$ processors, with $n_x, n_y \geq 2$, the box is first divided into n_x boxes along the x -axis, as shown in Figure 1b, each containing the same number of atoms. Each box is successively divided into n_y boxes along the y -axis in such a way that each one of the $n_x \times n_y$ boxes contains the same number of atoms (Fig. 1c). When, as in a real case, a third dimension exists, a successive division along the z -axis has to be performed.

It is obvious that, before performing any division along a given axis, it is necessary to sort the atoms of each box along that axis.

The density of a molecular system, such as a protein, is not uniform; thus, the boxes do not have the same axis lengths. However, these differences do not significantly reduce the efficiency of the GCA described in what follows.

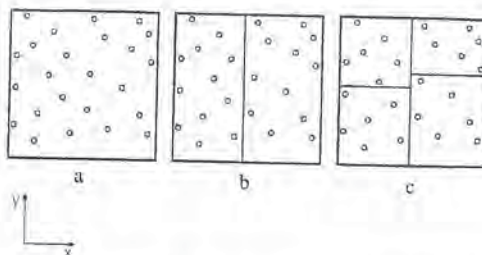


FIGURE 1. Domain decomposition of the molecular system in boxes with the same number of atoms, for a bidimensional case.

SYSTOLIC LOOP METHOD

Quadrics topology makes it possible to use a systolic loop to calculate the nonbonded interactions between the atoms assigned to the different nodes. The systolic loop method is one of the most efficient algorithms for calculation of two-body interactions on MIMD and SIMD machines.^{14, 16, 24, 25} The systolic loop algorithm passes the coordinates of all atoms around a ring of P processors in $P/2$ steps, such that half of the coordinates passes every processor exactly once (transient atoms). Each node also stores the coordinates of a group of atoms of the overall system (resident atoms). During the systolic cycle each processor evaluates and accumulates the interactions of the resident atoms with the transient ones. Only half of the atoms have to pass in each computational node as a consequence of the reciprocity of the interactions.

The systolic loop path for a 32-node Quadrics machine is shown in Figure 2. This machine has two nodes along the y and z directions and eight along the x direction.

The geometric decomposition of the system permits limitation of the search for nonbonded interactions only to the neighboring processors nearer than the cut-off radius, so that, depending on the number of nodes and on the system size, it is generally not necessary to perform the complete systolic loop. The computed forces are passed back to the owning processor to accumulate the full force.

GLOBAL CUT-OFF ALGORITHM

On a SIMD machine, all nodes simultaneously evaluate an interaction, but the atom pairs in each node are different. Figure 3 shows, as an example,

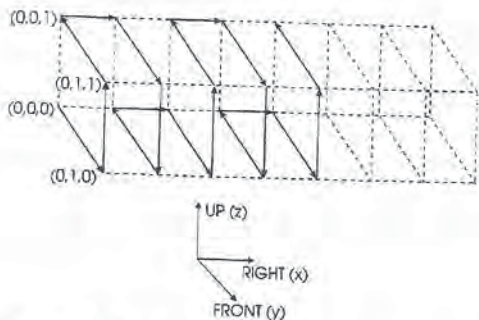


FIGURE 2. Systolic loop path for node (0,0,0) of a 32-node Quadrics machine. The transient groups of atoms visit only four neighboring $y-z$ planes, based on Newton's third law.

the case with four nodes: suppose that each node is evaluating the interaction a_i ; in this case, all a_i interactions fall within the cut-off radius. When the interactions are of the b_i type all the distances fall outside the cut-off radius and the interactions b_i are skipped. In the case of interactions of type c_i , the interaction is outside the cut-off radius in nodes 1, 2, and 3, but it is inside the cut-off radius in

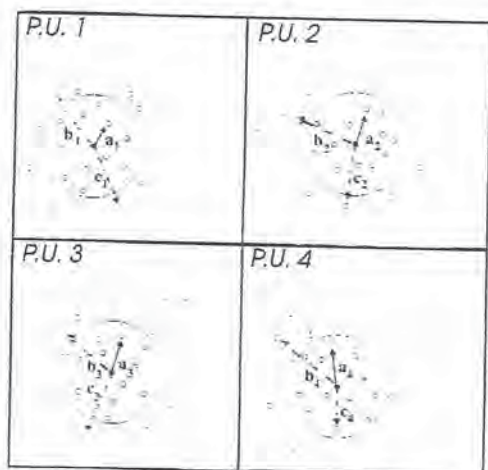


FIGURE 3. Different types of interactions in a case of four nodes. a_i , all the interactions fall within the cut-off distance; b_i , all the interactions fall outside the cut-off distance; c_i , one interaction falls within the cut-off, whereas three fall outside the cut-off. P.U. = processor unit.

node 4, so that all nodes have to calculate this interaction and only will be saved in the forces calculation. If the atoms in each node are ordered randomly, the interactions of type c_i result in being the most frequent.

The basis idea of the global cut-off algorithm (GCA) is to maximize the occurrence of interactions of type a_i and b_i and, conversely, reduce the occurrence of interactions of type c_i . To this end, it is necessary that the atoms in all nodes are sorted with the same criterion. Different types of sorting give comparable results. We have chosen the one shown in Figure 4. After this sorting procedure, a list of the interactions of type a_i and c_i is created in the integer CPU (Z-CPU) of the SIMD machine. This list is equivalent, but not identical, to the nonbonded pair list used in most MD programs and will be referred as the *nonbonded pair list*.

The ordering procedures for the domain decomposition and the sorting procedure previously described are time consuming and have to be performed on the host serial machine; however, as will be shown, they have to be performed every 100 to 200 steps so that they do not significantly affect the global computation time.

The global cut-off condition is based on a probabilistic approach, so that the number of pair interactions to be calculated is larger than the actual number of pairs included within the r_{cut} distance. Depending on the molecular system and on the number of nodes, the ratio between the number of the calculated interactions and the number of interactions actually included within the cut-off dis-

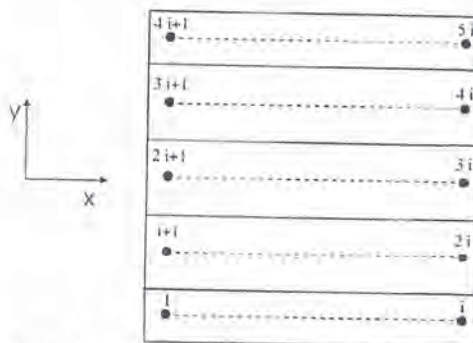


FIGURE 4. Sorting of atoms in each node for a bidimensional case. The atoms are represented as full circles.

tance varies from the three to five. In this sense, the GCA is not very efficient. However, it must be noted that almost all pair interactions within a distance $r_{cut} - r_{cut} - \Delta r$, with $\Delta r = 0.2$ nm, are calculated. As an example, given $r_{cut} = 0.6$ nm, 94% of the interactions in the range $0.6 - 0.8$ nm are calculated and only 6% of pairs in this range are lost. This suggests adoption of a cut-off value of r_{GCA} to be used in the global cut-off condition, somewhat shorter than the actual cut-off, r_{cut} , desired for the simulation. In the previous example, with $r_{GCA} = 0.6$ and $r_{cut} = 0.8$ nm, the number of pairs to be calculated is roughly two-to-threefold larger than the actual number of pairs within the r_{cut} distance. The remaining 6% of interactions in the range $0.6 - 0.8$ nm have to be calculated separately.

It is well known that nonbonded forces vary more slowly than the bonded ones. Moreover, nonbonded forces at large distances vary slower than nonbonded forces at short distances. This suggests updating of the forces at different steps, according to their nature (bonded and nonbonded) and to the distance of the interaction. The short-range interactions can be evaluated every step, and long-range interactions every m steps. Accordingly, the few interactions in the range $0.6 - 0.8$ nm that were lost using $r_{GCA} = 0.6$ nm can be updated every m steps. As these interactions are calculated while evaluating the *nonbonded pair list* (i.e., updated every n steps), we have chosen $m = n = 10$.

It must be noted that there are now two shells: an inner shell ($r = 0.6$ nm) and an outer shell ($0.6 - 0.8$ nm). All interactions are evaluated every m steps, whereas only those interactions corresponding to the inner shell are evaluated every step. It is therefore not necessary to have a skin distance and to store a list of atom pairs outside the outer shell.

In the present study it is seen that most interactions in the range 0.6 to 0.8 nm are evaluated every step and only a few of them are evaluated every m steps. According to all of the MTS algorithms, this choice does not affect the numerical accuracy; in fact, the same accuracy is obtained when an interaction, within the outer shell, is evaluated every short time step or every long time step. However, as every long time step all interactions within the outer shell are evaluated, it is possible to perform the MTS according to the classical procedure; that is, by collection all the interactions within the outer shell at every long time step and collecting only the interactions within the inner shell at every short time step. Among several algorithms proposed for the multiple time step (MTS)^{1,18,26} we have chosen the one developed by Scully and Hermans.¹⁸

It must be noted that all nonbonded interactions (van der Waals and Coulombic) between bonded and nonbonded atoms are calculated in this step, but the interactions between bonded atoms are not saved. This is obtained by attaching to each atom a list of the atoms bonded to itself. This procedure is certainly not efficient, but the time required to perform it is negligible. In the following tests, the values of r_{GCA} and r_{cut} are fixed at 0.6 and 0.8 nm, respectively.

Results

Table I shows the number of interactions within the cut-off radius r_{cut} compared with the number of interactions to be evaluated with the GCA. It can be noted that the number of interactions for calculation is two to three times the actual number of interactions within the cut-off radius. The time

TABLE I.
Number of Actual Interactions, N , within a Cut-Off Radius (r_{cut}) = 0.8 nm Compared with the Number of Interactions Calculated Using GCA on 32-, 128- and 512-Node Quadrics Machines.

	N	32 nodes	128 nodes	512 nodes
System 1 (4608 atoms)	488,847	706,944	840,320	873,984
System 2 (8704 atoms)	891,252	1,612,864	1,810,944	2,070,016
System 3 (15,360 atoms)	1,487,054	3,153,344	3,768,576	4,452,864
System 4 (30,720 atoms)	3,129,972	6,754,240	8,249,216	9,938,944

required for performing the MD simulation with the present code is the sum of the following steps:

1. Ordering procedure (performed every k steps by the host computer).
2. Calculation of the *nonbonded pair list* (performed every n steps by Quadrics).
3. Calculation of the nonbonded forces (performed every step by Quadrics).
- 3a. Calculation of the bonded forces by the host computer while performing step 3.
4. I/O host \leftrightarrow Quadrics.
5. SHAKE and integration (performed by the host).

The ordering procedure is a time-consuming task and, due to the diffusion of the system, it has to be periodically repeated every k steps. If no reordering is performed the *nonbonded pair list* will include an increasing number of interactions, thus increasing the computational time. Figure 5 shows

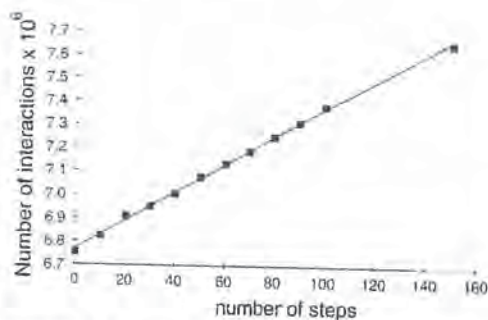


FIGURE 5. Number of interactions to be calculated versus the number of steps for system 4 on a 32-node machine. The atoms are not reordered during the simulation.

TABLE II.
cpu Times for Nonbonded Interactions (Expressed in Milliseconds per Step).

	DEC-alpha 3000 / 500	32 nodes	128 nodes	512 nodes
System 1 (4608 atoms)	2943	632	174	68
System 2 (8704 atoms)	6097	1650	386	107
System 3 (15,360 atoms)	10,928	3274	894	214
System 4 (30,720 atoms)	27,610	6208	1624	466

the number of interactions to be evaluated versus the number of steps when the ordering procedure is performed at the beginning and not updated, for system 4 on a 32-node machine.

The loss of performance is nearly linear, being $\sim 0.08\%$ per step. The optimum k value depends on the time required for the ordering procedure and on the time required for items 2 and 3. It shows that, for all the systems and all the different numbers of nodes, the optimum k value is in the range of 100 to 200 steps. The ordering procedure for system 4 on a Sun Sparc-20 (the host computer) required 20 seconds, so that its cpu time per step is in the range of 100 to 200 ms.

The *nonbonded pair list* is evaluated every n steps ($n = 10$ in the present case) and the nonbonded interactions are evaluated every step. The average cpu time required for these tasks is reported in Table II for different systems and different numbers of nodes. It should be emphasized that the parallel machines perform the calculation on a number of pairs two to three times larger than the serial one. The almost linear scalability of these task is also worthy of note.

The data transferred from the host to the Quadrics and vice versa after each reordering step (i.e., every $k \sim 100$ steps) are reported in Table III (upper panel); that is, 23 words per atom.

The data to be transferred every step are reported in Table III (lower panel)—9 words per atom. The average time spent in transmission depends on the speed of information transfer. Taking into account the speed of the HIPPI interface (20 MB/s) the average I/O times per step required for systems 1 to 4 are 9, 16, 28, and 55 ms/step, respectively.

The sums of times for items 1 to 4 with different numbers of nodes are reported in Table IV. Figure 6 shows the number of steps per second versus the

TABLE III.
Data Transfer Results.

	Quantity transferred (words per atom)
Data transferred every <i>k</i> steps	
From host to Quadrics	
Coordinates of the atom	3
Coordinates of the geometric center of the charge group	3
Electric Charge	1
Sequential atomic number	1
van der Waals parameters	2
Exclusions 1-3	6
Exclusions 1-4	4
From Quadrics to host	
Forces	3
Total	23
Data transferred every step	
From host to Quadrics	
Coordinates of the atom	3
Coordinates of the geometric center of the charge group	3
From Quadrics to host	
Forces	3
Total	9

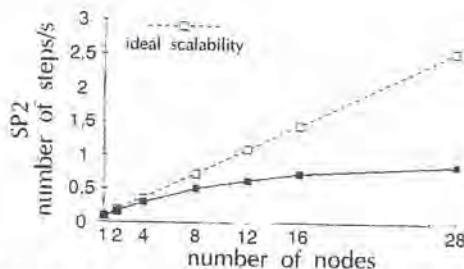
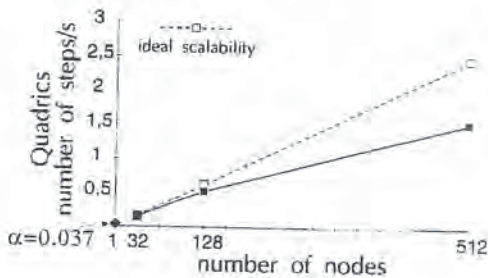


FIGURE 6. Number of steps per second versus the number of nodes for system 4. (a) Quadrics; (b) IBM SP2. The ideal scalability is expressed as a dashed line. Single processor DEC-alpha 3000 / 500 timing is shown for comparison.

number of nodes for Quadrics (Fig. 6a) and for an IBM-SP2 MIMD machine (Fig. 6b) for the same system (system 4). For clarity, the CPU times required for SHAKE and integration are not included. The code used for the MIMD machine was developed within the Europort2 PACC project.¹¹

Figure 6 also shows that no significant advantage is obtained with the MIMD machine when the number of nodes is 12, whereas, a good scalability, up to 512 nodes, is obtained with the SIMD machine.

TABLE IV.
Total cpu Times Including Nonbonded Interactions, Ordering Procedure, and I/O Host Quadrics (Milliseconds per Step).

	32 nodes	128 nodes	512 nodes
System 1 (4608 atoms)	680	205	90
System 2 (8704 atoms)	1790	470	175
System 3 (15,360 atoms)	3540	1050	335
System 4 (30,720 atoms)	6660	1910	685

bility, up to 512 nodes, is obtained with the SIMD machine.

To complete the evaluation of the total time steps of the present MD code, the times required on the host for the bonded interactions, SHAKE, and integration have to be calculated. The times required for these tasks, with the present MD code, are reported in Table V. It should be noted that the calculation of bonded interactions is performed by the host, whereas Quadrics computes the nonbonded interactions. As the former's calculation time is less than the latter, this task does not require any extra cpu time.

The integration task requires less cpu time than the nonbonded interaction calculation time (see Table IV) on a 32- or 128-node machine, and a comparable amount of time on a 512-node machine. Therefore, this task must be parallelized for machines with hundreds or thousands of processors. The integration can be implemented easily on a SIMD machine by, for example, the replicated data procedure.

The cpu time required to perform the SHAKE algorithm on the host is the actual bottleneck. It is

TABLE V.
cpu time on the Host (Sun Sparc-20) for Calculation
of Bonded Interactions, SHAKE, and Integration
(expressed in Milliseconds per Step).

	Bonded Interactions	SHAKE	Integration
System 1 (4608 atoms)	41	236	57
System 2 (8704 atoms)	96	488	117
System 3 (15,360 atoms)	73	1102	161
System 4 (30,720 atoms)	194	3013	437

very difficult to implement the SHAKE algorithm on a SIMD machine; therefore, an alternative procedure must be chosen. The MTS procedure can be used to evaluate the bond vibrations without reducing the time steps required for the nonbonded interactions.

Conclusions

The results reported in the present work show that the GCA permits use of pair lists even on a SIMD machine with no local addressing, thus overcoming one of the most severe disadvantages of SIMD vs. MIMD machines. The penalty to be paid is the number of interactions per step to be calculated; that is, two to three times the actual number of interactions.

The tests performed on Quadrics computers for configurations of 32, 128, and 512 nodes, for systems of different sizes, up to 30,000 particles, show that the scalabilities and performances are satisfactory and comparable to those obtained with MIMD machines. At the present time, the only routines for the calculation of the pair lists and nonbonded interactions have been parallelized. We have shown that the bonded forces can be calculated by the host while the parallel machine calculates the nonbonded ones. Also, the integration task can be calculated by the host if the parallel machine has up to tens of processors. With hundreds or thousands of processors this task must also be parallelized.

The SHAKE algorithm, which allows one to perform MD calculations at constant bond lengths, is the actual bottleneck of the calculation and its implementation is difficult with the parallel machine. We suggest the use of the MTS to evaluate the bond vibration contributions.

Acknowledgments

The authors thank ENEA for the use of the 128 and 512 Quadrics machines. One of us (R.B.) thanks ENEA for his hospitality during the realization of this work.

References

1. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Clarendon Press, Oxford, 1987.
2. W. F. van Gunsteren and H. J. C. Berendsen, *Angew. Chem. Int. Ed. Engl.*, **29**, 992 (1990).
3. M. R. S. Finches, D. J. Tildesley, and W. Smith, *Mol. Simul.*, **6**, 51 (1991).
4. H. Heller, H. Grubmüller, and K. Shulten, *Mol. Simul.*, **5**, 133 (1990).
5. D. C. Rapaport, *Comput. Phys. Commun.*, **62**, 217 (1991).
6. P. A. J. Hilbers and K. Esselink, *Computer Simulation in Chemical Physics*, M. P. Allen and D. J. Tildesley, Eds., Kluwer Academic Publisher, London, 1993, p. 473.
7. S. P. Plimpton, *J. Comput. Phys.*, **117**, 1 (1995).
8. S. P. Plimpton and B. Hendrickson, *J. Comput. Chem.*, **17**, 326 (1996).
9. B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, *J. Comput. Chem.*, **4**, 187 (1983).
10. W. F. van Gunsteren and H. J. C. Berendsen, *GROMOS: Groningen Molecular Simulation (GROMOS) Library Manual*, Biomos, Groningen, 1987.
11. D. G. Green, K. E. Meacham, M. Surrage, F. van Hoesel, and H. J. C. Berendsen, *Methods and Techniques in Computational Chemistry: Metacc-95*, E. Clementi and G. Corongiu, Eds., STEF, Cagliari, 1995, p. 435.
12. S. L. Lin, J. Mellor-Crummey, B. M. Pettit, and G. N. Phillips, *J. Comput. Chem.*, **13**, 1022 (1992).
13. W. T. Clark, R. Hanxleden, J. A. McCammon, and L. R. Scott, *Technical Report CRPC-TR93356-S*, Center for Research on Parallel Computation, Houston, TX, November 1993.
14. A. Windemuth and K. Shulten, *Mol. Simul.*, **5**, 353 (1991).
15. D. Fincham, *Mol. Simul.*, **1**, 1 (1987).
16. K. M. Nelson, C. F. Cornwell, and L. T. Wille, *Comput. Mater. Sci.*, **2**, 525 (1994).
17. P. Ballestrero, P. Baglietto and C. Ruggiero, *J. Comput. Chem.*, **17**, 469 (1996).

18. J. L. Scully and J. Hermans, *Mol. Simul.*, **11**, 67 (1993).
19. A. Bartoloni, G. Bastianello, C. Battista, S. Cabasino, F. Marzano, P. S. Paolucci, J. Pech, F. Rapuano, E. Panizzi, R. Sarno, G. M. Todesco, M. Torelli, W. Tross, P. Vicini, N. Cabibbo, A. Fucci, and R. Tripiccone, *Int. J. Modern Phys.*, **4**, 969 (1993).
20. A. Bartoloni, G. Bastianello, C. Battista, S. Cabasino, F. Marzano, P. S. Paolucci, J. Pech, F. Rapuano, E. Panizzi, R. Sarno, G. M. Todesco, M. Torelli, W. Tross, P. Vicini, N. Cabibbo, and R. Tripiccone, *Int. J. Modern Phys.*, **4**, 955 (1993).
21. The APE Collaboration, *Int. J. High Speed Comput.*, **5**, 637 (1993).
22. L. M. Barone, R. Simonazzi, and A. Tenenbaum, *Comput. Phys. Commun.*, **90**, 44 (1995).
23. J. P. Ryckaert, G. Ciccotti, H. J. C. Berendsen, *J. Comput. Phys.*, **23**, 327 (1977).
24. W. Smith, *Comput. Phys. Commun.*, **62**, 229 (1991).
25. J. F. Janak and P. C. Pattnaik, *J. Comput. Chem.*, **13**, 533 (1992).
26. M. Tuckerman, B. J. Berne, and G. J. Martyna, *J. Chem. Phys.*, **97**, 1990 (1992).

Attachment 4C



Search WorldCat

Search

[Advanced Search](#) [Find a Library](#)
[Cite/Export](#)
[Print](#)
[E-mail](#)
[Share](#)
[Permalink](#)
[Add to list](#)
[Add tags](#)
[Write a review](#)

Rate this item: 1 2 3 4 5

Journal of computational chemistry.

Publisher: [New York] Wiley.

Edition/Format: Journal, magazine : Periodical : English [View all editions and formats](#)Rating: (not yet rated) [0 with reviews - Be the first.](#)
 Subjects: [Chemistry -- Data processing -- Periodicals.](#)
[Chemistry -- methods.](#)
[Computer Simulation.](#)
[View all subjects](#)
More like this [Similar Items](#)

Search this publication for other articles with the following words:

 [Search](#)

Get a Copy

[Find a copy in the library](#)

Find a copy online

Links to this item

www3.interscience.wiley.com
 1980-1995

interscience.wiley.com
 1996-Current

Find a copy in the library

Enter your location: [Find libraries](#)

Submit a complete postal address for best results.

Displaying libraries 1-6 out of 812 for all 35 editions (5109 Cherry St, Kansas City, MO 64110, USA)

Show libraries holding [just this edition](#) or narrow results by [format](#)

Library	Held formats	Distance	
1. Linda Hall Library Kansas City, MO 64110 United States	Journal / Magazine / Newspaper	< 1 mile MAP IT	Library info Ask a librarian Add to favorites
2. University of Missouri-Kansas City Miller Nichols Library; Health Sciences Library; Dental Library Kansas City, MO 64110 United States	Journal / Magazine / Newspaper	< 1 mile MAP IT	Library info Add to favorites
3. University of Missouri-Kansas City UMKC School of Law, Leon E. Bloch Law Library Kansas City, MO 64110 United States	Journal / Magazine / Newspaper	< 1 mile MAP IT	Library info Add to favorites
4. University of Kansas Medical Center Archie R. Dykes Library Kansas City, KS 66160 United States	Journal / Magazine / Newspaper	2 miles MAP IT	Library info Ask a librarian Add to favorites
5. University of Kansas KU Library Lawrence, KS 66045 United States	Journal / Magazine / Newspaper	36 miles MAP IT	Library info Add to favorites

6. [Washburn University](#)
Mabee Library
Topeka, KS 66621 United States

 [Journal / Magazine / Newspaper](#)

60 miles
MAP IT

[Library info](#)
[Search at this library](#)
[Ask a librarian](#)
[Add to favorites](#)

<< First < Prev 1 2 3 Next > Last >>

Details

Genre/Form: Periodicals
Computer network resources

Additional Physical Format: Online version:
Journal of computational chemistry (Online)
(DLC)sn 97001336
(OCoLC)38142675
Online version:
Journal of computational chemistry
(OCoLC)753432630

Material Type: Periodical, Internet resource

Document Type: Journal / Magazine / Newspaper, Internet Resource

ISSN: 0192-8651

OCLC Number: 5081734

Notes: "Organic, inorganic, physical, biological."
"A Wiley-Interscience publication."
Published: Hoboken, N.J., Wiley Subscription Services, Inc., <2005->

Description: volumes illustrations 28 cm

Other Titles: Journal of computational chemistry

Reviews

User-contributed reviews

[Add a review](#) and share your thoughts with other readers. Be the first.

Tags

[Add tags](#) for "Journal of computational chemistry." Be the first.

Similar Items

Related Subjects: (7)

[Chemistry -- Data processing -- Periodicals.](#)

[Chemistry -- methods.](#)

[Computer Simulation.](#)

[Models, Chemical.](#)

[Chemistry -- Data processing.](#)

[Chemie.](#)

[Computermethoden.](#)

Linked Data

Attachment 4D



New Search:

[Search History](#)

[About](#)
[Permanent link to this record](#)

Journal of computational chemistry.

Title: Journal of computational chemistry.
Subjects: [Chemistry--Data processing--Periodicals.](#)
Additional title: [J. comput. chem.](#)
[Journal of computational chemistry](#)
Imprint: [New York] Wiley.
Physical format: v. ill. 28 cm.
Notes: "Organic, inorganic, physical, biological."
 "A Wiley-Interscience publication."
Frequency: 10 no. a year, 1990-
Former freq: Quarterly, -1983
 Bimonthly, 1984-<1989>
ISSN: 0192-8651
LCCN: 80643914 sn 79006892
CODEN: JCCHDD

This Item
Record view
 • [Staff view](#)


Actions
 • [Print](#)
 • [Export](#)
 • [E-mail](#)
 • [Add to My List](#)

Google Books
["About This Book"](#)

Holdings Information

Number of items: 1
Status: @LHL Serials - Closed Stacks
 Not Charged
Notes: Index in last issue of volume.
Issues: v.1:no.1(1980:Spring),v.4(1983)-v.34:iss.31/32(2013:Dec.5/15)-
 v.39 iss.15/16 2018 June 5/15
 Currently Received; waiting for v.39 iss.17/18 2018 June

Attachment 4E

[< Back to results](#) 1 of 1
[Export](#)
[Download](#)
[Print](#)
[E-mail](#)
[Save to PDF](#)
[Add to List](#)
[More... >](#)

[View at Publisher](#)

Computer Physics Communications
 Volume 139, Issue 1, 1 September 2001, Pages 1-19

The role of computer technology in applied computational chemical-physics (Article)

Chillemi, G., Rosati, M., Sanna, N. [✉](#)

CASPUR, Università La Sapienza, P. le A. Moro 5, 00185 Rome, Italy

Abstract

[View references \(48\)](#)

In this paper we would discuss the increasing role played by the past and upcoming silicon technology in solving real computational applications' cases in correlated scientific fields ranging from quantum chemistry, materials science, atomic and molecular physics and bio-chemistry. Although the wide range of computational applications of computer technology in this areas does not permit to have a full rationale of its present and future role, some basic features appear to be so clearly defined that an attempt to find common numerical behaviours become now feasible to be exploited. Several theoretical approaches have been developed in order to study the state of bound and unbound interactions among physical particles with the scope of having a feasible numerical path to the solution of the equations proposed. Apart from the evident scientific diversities among the cited computational fields, it is now becoming clear how they share common numerical devices, in terms of computer architectures, algorithms and low-level functions. This last fact, when coupled with the role of the numerical intensive technology provider who is committed to offer a computational solution to the needs of the scientific users on a common general-purpose computing platform, offers a unique way of analysis of the basic numeric requirements in this area. Some specific computational examples in classical and quantum mechanics of specific biochemistry and physics applications, will be reported in this paper and by the exposition of the basic elements of the theories involved, a discussion on the alternative to-and optimization-of-the use of current parallel technologies will be opened. Whenever possible, a comparison between some numerical results obtained on general purpose mid-range parallel machines and forecasts from on silicon routines will be carried out in order to understand the viability of this solution to the (bio)chemical-physics computational community. © 2001 Elsevier Science B.V. rights reserved.

Reaxys Database Information

[View Reactions](#) | [View Compounds](#)

Indexed keywords

Engineering
controlled terms:


Algorithms Computation theory Computer architecture Optimization Quantum theory

Engineering
uncontrolled terms:

Computational chemical-physics Computer technology

Engineering main
heading:

Computer aided analysis

Metrics  [View all metrics >](#)

4 Citations in Scopus
30th Percentile

0.23 Field-Weighted
Citation Impact






PlumX Metrics 

Usage, Captures, Mentions,
Social Media and Citations
beyond Scopus.

References (48)

[View in search results format >](#)

- All [Export](#)  [Print](#)  [E-mail](#) [Save to PDF](#) [Create bibliography](#)
- 1 Clementi, E., Corongiu, G. (1995) METECC-95, STEF, Italy
 - 2 Frish, M.J., Trucks, G.W., Schlegel, H.B., Scuseria, G.E., Robb, M.A., Cheesman, J.R., Zakrzewsky, V.G., (...), Pople, J.A. (1998) *Gaussian98 (Revision A.7)*. Cited 1679 times. Gaussian Inc., Pittsburg, PA
 - 3 Van Gunsteren, W.F., Daura, X., Mark, A.E. GROMOS force field (1998) *Encyclopedia of Computational Chemistry*, 2, pp. 1211-1216. Cited 169 times. N.L. Allinger, T. Clark, J. Gasteiger, P.A. Kollman, H.F. Schaefer III, P. Schreiner (Eds.), Wiley and Sons, London
 - 4 Achterop, S., Drunen, R.V., Spoel, D.V.D., Sijbers, A., Keegstra, H., Reitsma, B., Renardus, M.K.R. Gormacs: A parallel computer for molecular dynamics simulations (1993) *Proceedings of Physics Computing'92*, 1. World Scientific, Singapore
 - 5 Car, R., Parrinello, M. Unified approach for molecular dynamics and density-functional theory (1985) *Physical Review Letters*, 55 (22), pp. 2471-2474. Cited 7962 times. doi: 10.1103/PhysRevLett.55.2471
 [View at Publisher](#)
 - 6 Wang, C.Z., Ho, K.M. *Advances in Chemical Physics*, 89, p. 651. I. Prigogine, A.A. Rice (Eds.)
 - 7 Berendsen, H.J.C. Selected parallel applications and practical elements (1995) *Aspects of Computational Science* NCF, Chapter 6
 - 8 Green, D.G., Meacham, K.E., Surridge, M., Van Hoesel, F., Berendsen, H.J.C. Parallelization of molecular dynamics code. GROMOS87 parallelization for distributed memory architecture (1995) *Methods and Techniques in Computational Chemistry: METECC-95* STEF

Comparing the implementation of two-dimensional numerical quadrature on GPU, FPGA and ClearSpeed systems to study electron scattering by atoms

Gillan, C.J. , Steinke, T. , Bock, J. (2012) *Concurrency Computation Practice and Experience*

Programming challenges for the implementation of numerical quadrature in atomic physics on FPGA and GPU accelerators

Gillan, C.J. , Steinke, T. , Bock, J. (2010) *CCGrid 2010 - 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing*

SCELib3.0: The new revision of SCELib, the parallel computational library of molecular properties in the Single Center Approach

Sanna, N. , Baccarelli, I. , Morelli, G. (2009) *Computer Physics Communications*

[View all 4 citing documents](#)

Inform me when this document is cited in Scopus:

[Set citation alert >](#)

[Set citation feed >](#)

Related documents

Vector spherical harmonics: Concepts and applications to the single centre expansion method

Sanna, N. (2000) *Computer Physics Communications*




SCELib: a parallel computational library of molecular properties in the single center approach




Sanna, N. , Gianturco, F.A. (2000) *Computer Physics Communications*



SCELib2: The new revision of SCELib, the parallel computational library of molecular properties in the single center approach







Sanna, N. , Morelli, G. (2004) *Computer Physics Communications*

[View all related documents based on references](#)

- 9 Roccatano, D., Bizzarri, R., Chillemi, G., Sanna, N., Di Nola, A.
Development of a parallel molecular dynamics code on SIMD computers: Algorithm for use of pair list criterion
(1998) *Journal of Computational Chemistry*, 19 (7), pp. 685-694. Cited 4 times.
[http://onlinelibrary.wiley.com.ezproxy.lib.purdue.edu/journal/10.1002/\(ISSN\)1096-987X](http://onlinelibrary.wiley.com.ezproxy.lib.purdue.edu/journal/10.1002/(ISSN)1096-987X)
doi: 10.1002/(SICI)1096-987X(199805)19:7<685::AID-JCC1>3.0.CO;2-M
 View at Publisher
- 10 <http://www.caspar.it>
- 11 Sanna, N., Gianturco, F.A.
SCELib: a parallel computational library of molecular properties in the single center approach
(2000) *Computer Physics Communications*, 128 (1), pp. 139-169. Cited 14 times.
doi: 10.1016/S0010-4655(00)00078-3
 View at Publisher
- 12 <http://www.digital.com/hpc/software/dxml.html>
- 13 http://www.research.ibm.com/act/Opt_Lib/Topic_OptLibraries.html
- 14 note
- 15 Parr, R.G., Yang, W.
(1989) *Density Functional Theory of Atoms and Molecules*. Cited 14938 times.
Oxford University Press, New York
- 16 Colombo, L., Rosati, M.
Parallel tight-binding molecular dynamics simulations on symmetric multi-processing platforms
(2000) *Computer Physics Communications*, 128 (1), pp. 108-117. Cited 12 times.
doi: 10.1016/S0010-4655(00)00079-5
 View at Publisher
- 17 Chillemi, G., Pieretti, A., Rosati, M., Sanna, N.
The Performance of chemical-physics codes on CASPUR parallel architectures
CASPUR Tech. Rep. #10
in preparation

- 18 Van der Steen, A.J.
(1995) *Aspects of Computational Science*. Cited 6 times.
NCF Publ., Den Haag, The Netherlands
- 19 Roothaan, C.C.J.
New developments in molecular orbital theory
(1951) *Reviews of Modern Physics*, 23 (2), pp. 69-89. Cited 3709 times.
doi: 10.1103/RevModPhys.23.69
 View at Publisher
- 20 Hall, G.G.
(1951) *Proc. Roy. Soc. (London) A*, 205, p. 541. Cited 385 times.
- 21 Clementi, E.
(1990) *Appendices 7C-7E in MOTECC-90, Modern Techniques in Computational Chemistry*
ESCOM, Leiden, The Netherlands
- 22 Raffanetti, R.C.
Pre-processing two-electron integrals for efficient utilization in many-electron self-consistent field calculations
(1973) *Chemical Physics Letters*, 20 (4), pp. 335-338. Cited 36 times.
doi: 10.1016/0009-2614(73)80060-0
 View at Publisher
- 23 Schlegel, H.B., McDouall, J.
(1991) *Computational Advances in Organic Chemistry*. Cited 59 times.
C. Ogretir, I.G. Csizmadia (Eds.), Kluwer, The Netherlands
- 24 Pople, J.A., Schleyer, P.V., Hehre, W.J., Radom, L.
AB INITIO molecular orbital theory
AB INITIO molecular orbital theory. Cited 11457 times.
ISBN: 978-047181241-8

- 25 Allen, M.P., Tildesley, D.J.
(1991) *Computer Simulations of Liquids*. Cited 25797 times.
Oxford University Press, Oxford
- 26 Colombo, L.
(1996) *Annual Review of Computational Physics IV*, 4, p. 147. Cited 51 times.
D. Stauffer, (Ed.), World Scientific, Singapore


- 27 Kwon, I., Biswas, R., Wang, C.Z., Ho, K.M., Soukoulis, C.M.
Transferable tight-binding models for silicon
(1994) *Physical Review B*, 49 (11), pp. 7242-7250. Cited 208 times.
doi: 10.1103/PhysRevB.49.7242
 View at Publisher
- 28 <http://www.netlib.org/lapack>
- 29 Golub, G.H., Van Loan, C.F.
(1996) *Matrix Computations, 3rd edn.* Cited 33885 times.
Johns Hopkins University Press, Baltimore
- 30 Dongarra, J.J., Sorensen, D.C.
A fully parallel algorithm for the symmetric eigenvalue problem
(1987) *SIAM J. Sci. Stat. Comp.*, 8, pp. 175-186.
- 31 <http://www.nag.com/numeric/FS/FS.html>
- 32 <http://www.netlib.org/blas>
- 33 Ryckaert, J.-P., Ciccotti, G., Berendsen, H.J.C.
Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes
(1977) *Journal of Computational Physics*, 23 (3), pp. 327-341. Cited 12231 times.
doi: 10.1016/0021-9991(77)90098-5
 View at Publisher
- 34 Beguelin, A., Dongarra, J., Geist, A., Manchek, R., Sunderam, V.
(1991) *Oak Ridge Natl. Laboratory Tech Report TM-11826*, 1. Cited 3 times.
- 35 Gropp, W., Lusk, E., Skjellum, A.
(1994) *Using MPI Portable Parallel Programming with the Message Passing*. Cited 2235 times.
MIT Press
- 36 Bransden, B.H., Joachain, C.J.
(1987) *Physics of Atoms and Molecules*, p. 290. Cited 830 times.
J. Wiley and Sons, New York

- 37 Moccia, R.
One-center basis set SCF MO's. I. HF, CH₄, and SiH₄
(1964) *The Journal of Chemical Physics*, 40 (8), pp. 2164-2176. Cited 114 times.
doi: 10.1063/1.1725489
 View at Publisher
- 38 Gianturco, F.A., Thompson, D.G., Jain, A.K.
(1994) *Computational Methods for Electron Molecule Collisions*. Cited 238 times.
W.M. Huo, F.A. Gianturco (Eds.), Plenum, New York
- 39 Gianturco, F.A., Lucchese, R.R., Sanna, N.
Low-energy electron scattering by halomethanes: Elastic and differential cross sections for CF₄
(1996) *Journal of Chemical Physics*, 104 (17), pp. 6482-6490. Cited 26 times.
<http://scitation.aip.org.ezproxy.lib.purdue.edu/content/aip/journal/jcp>
doi: 10.1063/1.471361
 View at Publisher
- 40 Gianturco, F.A., Lucchese, R.R., Sanna, N.
Computed elastic cross sections and angular distributions of low-energy electron scattering from gas phase C₆₀ fullerene
(1999) *Journal of Physics B: Atomic, Molecular and Optical Physics*, 32 (9), pp. 2181-2193. Cited 32 times.
doi: 10.1088/0953-4075/32/9/309
 View at Publisher
- 41 Altmann, S.L.
On the symmetries of spherical harmonics
(1957) *Mathematical Proceedings of the Cambridge Philosophical Society*, 53 (2), pp. 343-367. Cited 81 times.
doi: 10.1017/S0305004100032370
 View at Publisher
- 42 Gianturco, F.A., Lucchese, R.R., Sanna, N.
Calculation of low-energy elastic cross sections for electron-CF₄ scattering
(1994) *The Journal of Chemical Physics*, 100 (9), pp. 6464-6471. Cited 103 times.
doi: 10.1063/1.467237
 View at Publisher
- 43 Homeier, H.H.H., Steinborn, E.O.
Some properties of the coupling coefficients of real spherical harmonics and their relation to Gaunt coefficients
(1996) *Journal of Molecular Structure: THEOCHEM*, 368 (1-3 SPEC. ISS.), pp. 31-37. Cited 49 times.
doi: 10.1016/S0166-1280(96)90531-X
 View at Publisher

44 Sanna, N., Gianturco, F.A.
SCELib API: An interface to the parallel computation of molecular properties
in preparation

45 Sanna, N.
Vector spherical harmonics: Concepts and applications to the single centre expansion method


(2000) *Computer Physics Communications*, 132 (1-2), pp. 66-83. Cited 3 times.
doi: 10.1016/S0010-4655(00)00137-5

 View at Publisher

46 Press, W.H.
(1992) *Numerical Recipes, 2nd edn.* Cited 31489 times.
Cambridge University Press

47 Smith, J.M., Olver, F.W.J., Lozier, D.W.
Extended-Range Arithmetic and Normalized Legendre Polynomials

(1981) *ACM Transactions on Mathematical Software (TOMS)*, 7 (1), pp. 93-105. Cited 23 times.
doi: 10.1145/355934.355940

 View at Publisher

48 Curik, R., Gianturco, F.A., Sanna, N.
(2000) *J. Phys. B*
to be published

© Copyright 2017 Elsevier B.V., All rights reserved.

[< Back to results](#) 1 of 1

[^ Top of page](#)

About Scopus

[What is Scopus](#)
[Content coverage](#)
[Scopus blog](#)
[Scopus API](#)
[Privacy matters](#)

Language

[日本語に切り替える](#)
[切换到简体中文](#)
[切换到繁體中文](#)
[Русский язык](#)

Customer Service

[Help](#)
[Contact us](#)



[Terms and conditions](#) [Privacy policy](#)

Copyright © 2018 Elsevier B.V. All rights reserved. Scopus® is a registered trademark of Elsevier B.V.

Cookies are set by this site. To decline them or learn more, visit our [Cookies page](#).





The role of computer technology in applied computational chemical-physics

G. Chillemi, M. Rosati, N. Sanna*

CASPUR, Università "La Sapienza", P. le A. Moro 5, 00185 Rome, Italy

Abstract

In this paper we would discuss the increasing role played by the past and upcoming silicon technology in solving real computational applications' cases in correlated scientific fields ranging from quantum chemistry, materials science, atomic and molecular physics and bio-chemistry. Although the wide range of computational applications of computer technology in this areas does not permit to have a full rationale of its present and future role, some basic features appear to be so clearly defined that an attempt to find common numerical behaviours become now feasible to be exploited.

Several theoretical approaches have been developed in order to study the state of bound and unbound interactions among physical particles with the scope of having a feasible numerical path to the solution of the equations proposed. Apart from the evident scientific diversities among the cited computational fields, it is now becoming clear how they share common numerical devices, in terms of computer architectures, algorithms and low-level functions. This last fact, when coupled with the role of the *numerical intensive technology provider* who is committed to offer a computational solution to the needs of the scientific users on a common general-purpose computing platform, offers a unique way of analysis of the basic numeric requirements in this area.

Some specific computational examples in classical and quantum mechanics of specific biochemistry and physics applications, will be reported in this paper and by the exposition of the basic elements of the theories involved, a discussion on the alternative to — and optimization of — the use of current parallel technologies will be opened. Whenever possible, a comparison between some numerical results obtained on general purpose mid-range parallel machines and forecasts from *on silicon* routines will be carried out in order to understand the viability of this solution to the (bio)chemical-physics computational community. © 2001 Elsevier Science B.V. All rights reserved.

1. Introduction

In the last two–three decades or so, a huge scientific production of numerical results have been carried out in the fields of applied theoretical chemical-physics, an area which embrace many different computational fields which include quantum chemistry, atomic and molecular physics, classical (and *ab-initio*) molecular dynamics, just to cite a few.

* Corresponding author.

E-mail address: n.sanna@caspur.it (N. Sanna).

Many attempts have been made to summarize the theories and methods involved in those scientific fields [1] while some numerical specific codes play now a leading role in this arena [2–5]. Thanks to this huge scientific work it is now customary to adopt the computational tool as a basic investigation device in almost any area of the experimental (bio)chemical-physics.

The area of Quantum Chemistry is traditionally a field which requires huge computing resources in order to solve quantum Hamiltonians describing static properties of nuclear and electronic particles by means of variational energy-based methods. With the last decade of increasing computational applications of the Density Functional Theory (DFT) method to the solution of physical problems, an open debate is on going on the use of this, against the traditional *wavefunction* quantum methods. Both methods remain extremely costly in terms of computing resources and several studies are on going in order to setup computational techniques able to scale with the increasing complexity of the atomic and molecular systems studied nowadays which are still domain of the classical particles simulation.

In the last years the remarkable growth of the computer power has made the application of refined and expensive models possible to materials science. The well-known Car–Parinello Molecular Dynamics (CPMD) method [5] is a parameter-free simulation scheme in principle superior than any MD simulation based on empirical interatomic potential. Unfortunately, its computational cost is so heavy than only short simulations (~ 10 ps) and systems with limited number of atoms (~ 100), are actually feasible. Quite a number of physical systems (like amorphous materials, extended defects nano-crystalline materials) and phenomena (like defect migration, microstructural evolution under irradiation, crack propagation) relevant to materials science, are out of reach of CPMD. On the other hand, the classical potentials in many cases do not present a good transferability to different physical conditions. Recently it has been proposed the Tight-Binding Molecular Dynamics [6] simulation scheme that has the accuracy needed to describe complex systems and a reduced computational workload with respect to CPMD.

Biomolecules are one of the most complex subjects of study as they involve many thousands degrees of freedom for a given molecule and, at the same time, their biological mechanisms span many different timescales from nanoseconds to milliseconds and beyond. It is not surprising, then, if computer simulations of biological systems has quickly taken advantage of parallel computer architectures. In recent years many efforts have been made in order to realize efficient implementations of parallel MD codes specifically designed to simulate biological systems [7–9].

Leading calculations using those methods require a huge computing power either in term of CPU and/or I/O resources and top level computer architectures are often used. Those computing machines are built on top of commodity components and this trend seems that will be consolidated even in the upcoming future parallel architectures. From the application side, it is now becoming evident from the analysis of the computational production in those fields, that those areas share some common unique features: (i) they require sophisticated numerical algorithms; (ii) they make use of the most numerical intensive hardware and software devices; (iii) they iteratively use basic — low level — routines to reach the numerical solution.

Among others, the last point open a way to understand the role that can be played by *on silicon*, specific-purpose device, which can represent a viable, valid alternative to the use of mid-range to massively parallel machines.

2. The numerical application requirements

Independently of the theories and methods (classical or quantum) involved, we would try to summarize some key factors which affect the performances of the codes in which those theories are implemented on. We would approach this by the point of view of the *technology provider* — often a computer center, like CASPUR [10] — where many of those codes run on the same *general purpose* parallel machines. Even if this approach cannot be thought to be exhaustive, perhaps it can help to understand some common requirements from the point of view of an *external observer* with respect of the scientific areas those theories were generated from.

During the last ten years or so, we at CASPUR have identified some key algorithms and low-level functions that have had the broader usage among the numerical codes which implement the most representative theories and methods in (bio)chemical-physics. By selecting the proper input data for those codes, we were able to use them as a reliable benchmarks/test-beds of our machines before to open them to the whole community of users.

Let us now summarize below some key factors present in the theories and methods described so far, from which we have identified leading sections of codes representing some of the most time consuming applications at CASPUR:

- Many methods use basis functions (orbitals) to project quantum operators over them [2,5,11]. Cartesian or spherical Gaussian Type Orbital (GTO) are often used and manipulated.
- Several methods make use of pure/mixed numerical basis sets which are composed by an analytic (typically the angular part) and a pure numeric part [11].
- Among the operations between basis sets elements, their numerical integration over a given range of the independent variables is often performed [2,5,11]. This, can be accomplished either analytically or numerically, in dependence of the fact that the integrand function is known or not.
- First and second (and less often, higher) derivatives of basis sets elements, their combined products and integrals, are often performed at different level in all of the referenced codes. This can be carried out as above, either analytically or numerically.
- Some basic library or intrinsic functions are intensively used in many ways in all of the referenced codes, to perform higher level numerical operation. These include linear algebra subprograms, Fast Fourier Transforms and Splines fitting, just to cite a few.

The actual implementation of the algorithms for the manipulation of the above cited theory key-factors, make intensive use of external library functions, often adapted by the hardware vendor to perform at the best on a specific processor and/or computer architecture [12,13]. Moreover, some intrinsic functions and low-level routines, like transcendental and exponent/power functions or simple vector/matrix operation routines, are now implemented in assembly language codes external with respect to the implicit on chip functions [14].

3. Application specific computational kernels

In this section we would sketch the basic elements of the theories which give rise to the most intensive computational kernels in the area of (bio)chemical-physics as we have experienced at CASPUR. Of course, we understand that a broad coverage of this topics is out of reach for anybody involved in this area, but we would in any case try to follow a *connection approach* among related scientific disciplines, a method that we found extremely useful in order to understand the computational needs of our users in this scientific areas.

By *connection approach* we mean a path of describing the core elements of the various computational procedures used, by trying to connect them with some elements of similarity which are uniquely defined.

In this context, let us begin our discussion with wavefunction based Quantum Chemistry codes, and connect them with the basic linear algebra (BLAS) subsections they intensively use, to the *BLAS-dependent* Tight-Binding codes [6]. From here, by expanding the needs of a more effective potential form for many particles systems, we will discuss some computational elements of classical Molecular Dynamics which can be correlated with the area of electron/positron scattering by atoms and molecules from bound state potentials.

We should note however, that the conclusions derived here are not specific of the codes analyzed below and they can easily be applied to closely related computational areas (i.e. Ab-Initio Carr-Parinello molecular dynamics [5] and Density Functional Theory methods [15]) which we do not discuss here for the sake of readability, but that we think will surely play an important role in the next future as key computational applications in chemical-physics.

A last note before to going into the details, is about the parallel architectures used for the codes under test. In order to maintain a broad architectural similarity, we decided to use Symmetric Multi-Processing machines (SMPs) belonging to the class of the mid-range general-purpose computing servers. Among those available on the

market we chose only those currently available at CASPUR that were integrated in our computing environment at production level. Although this can be seen as a reductive choice to apply, it seemed to us that apart from the similarity in the architecture, one key point was to test the hardware and software for numerical production within the same computing environment (Operating and Queuing system, hardware/software configuration), a fact which is often overlooked when benchmarks results are published from machine vendors. We would just recall here that we used Compaq (ES40, 4x ev6@500 MHz), IBM (SP3, 8x PWIII@222 MHz) and SUN (ES4500, 14x SparcII@336 MHz) SMP machines with similar hardware configuration, while we leave the interested reader to recent papers [11,16,17] for any additional details on the computing environment used for the tests subject of discussion in this paper.

3.1. Quantum chemistry: the Hartree-Fock method

Let us begin our discussion by reporting some specific results of one of the most used package at CASPUR: the Gaussian98 [2]. We show in Table 1 the parallel timings and speedups resulting from the runs of this code over the three SMP architectures we had given to our users. In order to better show the parallel efficiencies among the tested SMP architectures, we also compare in Fig. 1 the resulting collective speedups. The results refer to frequency calculations of the FluoroBenzene molecule at the Hartree-Fock SCF level (HF/6-311++G(3df,pd)), a computational test we have had running for several years at CASPUR over different serial and parallel machines [17].

Although the relative performances among the architectures we tested are very different [17], at a first sight the results obtained behave very similar and a common trend in parallel efficiencies can be easily derived. In fact, the speedups closely follow the Amdahl law [18] expected for this type of code up to 14 SMP processors and this behaviour does not seem to depend on the the parallel SMP machines under test, a fact that confirms the similarities in their architectural design. Furthermore, this common trend in the parallel efficiency, shows that

Table 1
Hartree-Fock SCF — HF/6-311++G(3df,pd) — FluoroBenzene Gaussian98 parallel runs on Compaq, IBM and SUN SMP architectures.
Elapsed times (E.T.) in seconds and speedup vs. the number of nodes

Nodes	Compaq 4x ES40		IBM 8x SP3		SUN 14x ES4500	
	E.T. (s)	Speedup	E.T. (s)	Speedup	E.T. (s)	Speedup
1	34.536	1.0	47.201	1.0	157.670	1.0
2	19.407	1.8	23.676	2.0	79.848	2.0
3	13.043	2.6	16.911	2.8	54.393	2.9
4	10.093	3.4	13.108	3.6	42.628	3.7
5	N/A	N/A	10.522	4.5	35.053	4.5
6	N/A	N/A	8.769	5.4	30.335	5.2
7	N/A	N/A	7.892	6.0	26.290	6.0
8	N/A	N/A	7.516	6.3	23.625	6.7
9	N/A	N/A	N/A	N/A	21.032	7.5
10	N/A	N/A	N/A	N/A	19.201	8.2
11	N/A	N/A	N/A	N/A	17.925	8.8
12	N/A	N/A	N/A	N/A	16.638	9.5
13	N/A	N/A	N/A	N/A	15.465	10.2
14	N/A	N/A	N/A	N/A	14.340	11.0

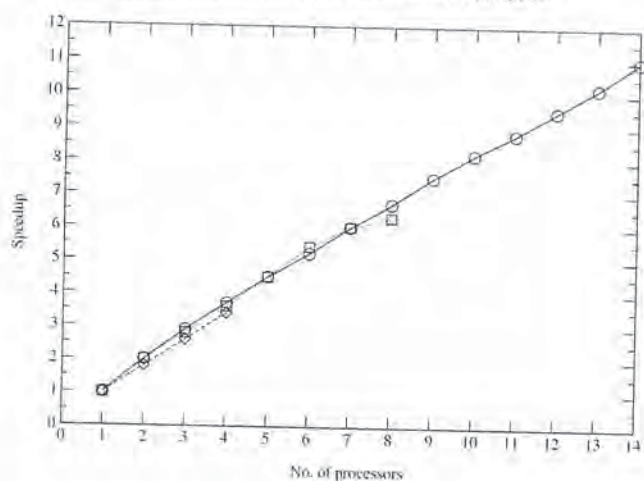


Fig. 1. Speedup vs. no. of processors for the SMP architectures under test: Compaq ES40 (diamonds), IBM SP3 (squares) and SUN ES4500 (circles).

similar low-level algorithms and programming models were used (i.e. the same section of code was executed) and that the parallel implementation (via Inter Process Communication — IPC) remains the same across the binary executables used over the parallel machines under test.

In order to understand this behaviour, let us show some key elements of the theory behind those results, with the aim to focus the attention over its most time-consuming computational parts.

The basic algorithmic elements of any wavefunction based Quantum-Chemistry code, refers to the Hartree-Fock — HF — (Roothaan-Hall) [19] eigenvalue matrix equation

$$FC = SC\epsilon, \quad (1)$$

where the unknowns are the C eigenvectors with the ϵ eigenvalues, provided you got the overlap matrix S and the Fock matrix F . The computational process is iteratively solved within the Self-Consistent Field (SCF) approach with the calculation of the Fock matrix terms

$$F_{\mu\nu} = H_{\mu\nu}^{\text{core}} + \sum_{\lambda\sigma} P_{\lambda\sigma} \left[(\mu\nu|\sigma\lambda) - \frac{1}{2} (\mu\lambda|\sigma\nu) \right] \quad (2)$$

and then by the F diagonalization, to obtain the C eigenvectors and the ϵ eigenvalues (in Eq. (2) we labeled with Greek letters the single elements of the basis set used to build the Molecular Orbitals — MOs).

So, the entire computing flow depends on two steps, regarding the Fock (F) matrix: (i) the F calculation and, (ii) the F diagonalization.

The F calculation depends on the so called *two-electron integrals* evaluation, being these some objects built over Cartesian Gaussian basis functions (the basis set) on top of primitive Gaussians

$$g(\alpha, \vec{r}) = N x^n y^m z^l e^{-\alpha r^2}, \quad (3)$$

this computational step can be thought to be roughly dominated by the *exp* function evaluation. But, by focusing our attention on the hidden details, we see that the calculation of these integrals (often referred to as *four centres integrals*) depends on the evaluation of two parts, here shown as the product of two functions and called *f* and *F*, respectively

$$(ij|kl) = f(r) \times F(e^{-ur^2}) \quad (4)$$

which have the appealing feature to be *recursive functions* of the π constant [20] and of the *exp* function.

The second time-consuming step of the HF-SCF procedure, the *F* diagonalization, is a typical linear algebra problem which is mediated by the use of the molecular symmetry eventually present in the chemical system under study. In fact, the diagonalization of the whole Fock matrix *F* occurs very seldom, while typically this step is carried out on sub-matrices belonging to specific Irreducible Representations (IRs) of the molecular symmetry the chemical system under study belongs to.

We should note however, that the HF-SCF procedure can be carried out in two (among others) distinct methods: (i) the Conventional SCF and (ii) the Direct SCF. The former procedure refers to the traditional method of once calculate-and-store on disk the two-electron integrals at the beginning of the iterative procedure [21], while the latter carries out an integral evaluation at each step of the SCF procedure with some thresholds to be applied on the exponents of the Gaussian integrand functions [22].

By taking into account this, the *F* calculation/diagonalization timing needs of a SCF procedure, can be expressed as a function of the basis set expansion *N*. We should note however, that *N* represent the actual basis set number, often referred to as *contracted basis* where several linear combination of primitive Gaussians are present. We report in Table 2 the expected computational performance of the SCF Conventional and Direct procedures with respect to the actual number (*N*) of contracted basis set functions used [23].

It remains now to identify the relative weights of the *F* calculation and diagonalization phases in order to completely evaluate the performance of a generic HF-SCF based code. In Table 3 we report the percentage of the mean CPU time spent in the computation/diagonalization phases of the Fock *F* matrix. We should note that, the resulting percentages are derived from a mean evaluation of the CPU time spent of serial runs over several tests we collected at CASPUR during the last ten years or so.

It becomes clear at this stage that the computational performance of a generic SCF-HF code depends on the balance of two well identified phases (the *F* calculation and then its diagonalization) which are dependent on two

Table 2
The scaling of the *F* evaluation in the HF-SCF procedure as function of the number of contracted basis sets functions (*N*)

Calculation stage	HF method	Scaling factor
<i>F</i> calculation	Conventional	$\approx N^4$
<i>F</i> calculation	Direct	$\approx N^{2.7}$
<i>F</i> diagonalization	Both	$\approx N^3$

Table 3
Percentage of the CPU time spent using the conventional/direct HF-SCF method

Calculation stage	SCF direct	SCF conventional
	CPU time	CPU time
<i>F</i> calculation	80–95%	30–50%
<i>F</i> diagonalization	5–20%	50–70%

low-level computing cores: (i) simple and recursive functions evaluation and (ii) iterative use of basic linear algebra sub-programs. The relative importance of these two computational elements cannot be easily evaluated (as will be later pointed out in Section 4) but, they are uniquely identified as the *computational kernels* of those kind of codes.

Now, by assuming the case of a specific SCF-HF run bounded to the latter computing core (it is not so rare that the diagonalization phase dominates a HF-SCF calculation), let us discuss the computational behaviour of a scientific area, the Tight-Binding Molecular Dynamics (TBMD), which make an extensive use of linear algebra sub-programs and in particular of diagonalization routines. This can offer a unique way to exploit the specific features on the application of these computing cores but keeping in mind their relations with closer scientific areas.

3.2. Materials science: the Tight-Binding Molecular Dynamics

In a Molecular Dynamics (MD) run we must compute the interatomic forces F_α ($\alpha = 1, 2, \dots, N_{at}$)¹ to numerically integrate the Newton's equation of motion and to generate trajectories in the phase space [24]. The force F_α is given by:

$$\mathbf{F}_\alpha = -\frac{\partial H}{\partial \mathbf{R}_\alpha}, \quad (5)$$

where H is the Hamiltonian of the system under study and \mathbf{R}_α is the position vector of the α th particle.

In the Tight-Binding Molecular Dynamics (TBMD) the Hamiltonian is described by the equation:

$$H = \sum_\alpha \frac{p_\alpha^2}{2m_\alpha} + 2 \sum_n^{(\text{occup})} \epsilon_n + U_{\text{rep}}, \quad (6)$$

where the superscript (occup) indicates that we use just electron energies ϵ_n belonging to the lower half spectrum of the TB matrix h [25]. Besides eigenvalues ϵ_n , we need the eigenvectors \mathbf{b}^n of the h matrix to compute the interatomic forces F_α :

$$\mathbf{F}_\alpha = -\frac{\partial H}{\partial \mathbf{R}_\alpha} = -\frac{\partial}{\partial \mathbf{R}_\alpha} 2 \sum_n^{(\text{occup})} \epsilon_n - \frac{\partial}{\partial \mathbf{R}_\alpha} U_{\text{rep}} \quad (7)$$

and

$$\begin{aligned} -\frac{\partial}{\partial \mathbf{R}_\alpha} 2 \sum_n^{(\text{occup})} \epsilon_n &= -2 \frac{\partial}{\partial \mathbf{R}_\alpha} \sum_n^{(\text{occup})} \langle \Psi_n | h | \Psi_n \rangle \\ &= -2 \frac{\partial}{\partial \mathbf{R}_\alpha} \sum_n^{(\text{occup})} \sum_{l\gamma} \sum_{l'\beta} b_{l\gamma}^n b_{l'\beta}^n \langle \varphi_{l\gamma} | h | \varphi_{l'\beta} \rangle \\ &= -2 \frac{\partial}{\partial \mathbf{R}_\alpha} \sum_n^{(\text{occup})} \sum_{l\gamma} \sum_{l'\beta} b_{l\gamma}^n b_{l'\beta}^n h_{ll'}(\mathbf{R}_{\gamma\beta}), \end{aligned} \quad (8)$$

where $b_{l\gamma}^n$ is the component of the eigenvector \mathbf{b}^n related to the l th orbital of the γ th atom and $h_{ll'}(\mathbf{R}_{\gamma\beta})$ is the hopping integral (element of the TB matrix) of the l and l' orbitals of the γ th and β th atom, respectively, when the h matrix is expressed using a basis set of orthogonal one electron states (Löwdin orbitals).

Therefore, at each time-step of TBMD, we need the first half set of eigenvalues and eigenvectors of a real and symmetric matrix of order $N = \sum_\alpha n_\alpha^{\text{orb}}$ where n_α^{orb} is the number of active orbitals of the α th atom. In a typical TBMD simulation more than 95% of CPU time is spent in diagonalization of the TB matrix:

$$h \mathbf{b}^n = \epsilon_n \mathbf{b}^n, \quad (9)$$

¹ N_{at} is the number of atoms of the system under study.

Table 4
CPU time per TBMD time-step (in seconds) on a Compaq cv6 platform as function of the number N_{at} of atoms of the system under study for three different eigensolvers of LAPACK version 3

N_{at}	DSYEV	DSYEVX	DSYEVD
64	0.32	0.25	0.25
144	3.03	1.74	2.17
216	10.22	5.23	7.05
288	33.60	12.63	16.23
384	88.43	31.63	38.11
448	145.92	49.56	58.30
512	240.67	110.71	112.44
576	299.97	106.08	122.61
640	414.98	146.80	169.47

Furthermore, this operation impose the overall workload scaling law with respect to the number of atoms in the system under study: it will result of $O(N^3)$ type.

In order to optimize the numerical throughput of TBMD simulations we have tested various eigensolvers of real and symmetric large matrices: we report in Table 4 a comparison of the CPU time needed for a time step of TBMD² per various system sizes using three different routines of the last version of LAPACK [27]:

DSYEV: first of all this routine reduces the matrix to symmetric tridiagonal form by an orthogonal similarity transformation, then it computes all eigenvalues and relatively eigenvectors using an implicit QR method [28].

DSYEVX: this routine compute a user defined subset of eigenvalues spectrum and related eigenvectors; we have used them to determine the part of spectrum corresponding to the the first half of the occupied states.

DSYEVD: this routine reduces the matrix to symmetric tridiagonal form using the same algorithm of DSYEV, then all eigenvalues and eigenvectors are evaluated by Cuppen's *divide and conquer* scheme, which presents a workload scaling with respect to the size of the matrix better than QR procedure [28]; moreover *divide and conquer* algorithms are naturally suitable for parallel computing [29].

The first set of benchmark reported in Table 4 sketches an interesting scenario: the routine based on *divide and conquer* scheme is much more performing then its competitors: it is 2–3 time faster than the corresponding full eigensolver DSYEV for any system size. Moreover DSYEVD is only 13–15% slower with respect to DSYEVX for bigger systems within of reach of single processor high-end workstations. This is a very promising result at least for two reasons:

- (1) the much higher amount of physical information provided by DSYEVD than by DSYEVX;
- (2) on the contrary of DSYEVX, the *divide and conquer* routine DSYEVD presents an high rate of intrinsic parallelism [29].

Therefore we are confident than the next generation of parallel TBMD code at CASPUR (*TBpack*), based on *divide and conquer* eigensolver provided by the new version of NAG SMP Fortran Library [30], will be run on our SMP architectures with speedup as fast as those of the actual version [16] and with an algorithmic efficiency in diagonalization task 2–3 times better than the present *TBpack* version, which is based on a parallel DSYEV like routine provided by NAG SMP Fortran Library 1.0.

The correct selection of an eigensolver is only the first step to increase the computational throughput of a TBMD run; the real key issue to obtain best performance is in the choice of the BLAS (Basic Linear Algebra Subprograms)

² All TBMD simulations have been carried out on a crystalline silicon supercell at 600 K, using the TB representation by Kwon et al. [26].

Table 5
CPU time (in seconds) needed by a *divide and conquer* eigensolver as function of the order of the matrix, using BLAS library either vendor provided or generated by source code Fortran77 freely available. Results has been produced using a Compaq Alpha ev67@667 MHz and an IBM Power3@345 MHz both with 4 MB of L2 Cache memory

Size	Compaq ev67@667 MHz		IBM Power3@345 MHz	
	F77 BLAS	CXML	F77 BLAS	ESSL
256	0.28	0.14	0.26	0.14
384	0.88	0.40	0.76	0.40
576	2.76	1.17	2.44	1.18
864	10.31	3.81	8.54	3.60
1152	26.85	9.09	22.21	8.47
1536	69.35	21.48	57.37	20.96
1792	120.23	33.51	92.09	31.64
2048	191.44	66.10	140.56	48.33
2304	N/A	N/A	198.41	66.98
2560	N/A	N/A	277.00	93.13

library [31], whose Level 3 routines are the basic building blocks of the *divide and conquer* eigensolvers by LAPACK. To estimate the effects of the BLAS performance on the DSYEVD routine, first of all we have generated real TB matrices of order ranging $N = 256$ to $N = 2560$. Then, we have used the DSYEVD routine linked to a BLAS library especially tuned by the machine vendor on its CPU architecture and memory hierarchy³ and the equivalent Fortran77 BLAS routine available in NETLIB [31], compiled with the typical options⁴ used in our computing environment. This test has been carried out on Compaq and IBM high-end serial workstations, based on a Alpha ev67@667 MHz and a Power3@345 MHz processors, respectively.

Results reported in Table 5 show that, if the size of the matrix is greater than 600,⁵ the DSYEVD routine based on the CXML/ESSL BLAS library is 3–4 times faster than the same routine linked with the Fortran77 coded BLAS. Therefore a TBMD simulation build up on top of a *divide and conquer* eigensolver and a BLAS library especially tuned on the hardware available, could be even an order of magnitude faster than the same simulation based on a standard QR algorithm and the NETLIB BLAS library.

In addition to the full diagonalization of the TB matrix, a time step of TBMD requires the evaluation of the repulsive part of the potential energy U_{rep} . This contribution is expressed in term of a many-body analytical function, whose evaluation requires less than 1% of the total CPU time so, unlike the Classical Molecular Dynamics, an optimization of this stage is unnecessary in TBMD codes.

3.3. Classical Molecular Dynamics: Simulation of a biological system

The basic elements of a Classical Molecular Dynamics (MD) simulation are the same of the Tight-Binding TBMD ones; what makes the difference between the two methods is the form of the inter-particle interaction potential and the relative computational weight of its components.

³ CXML library for Compaq Alpha architecture and ESSL for IBM RS6000 architecture.

⁴ Compaq Alpha: -O5 -fpe1 -tune host and IBM RS6000: -O3 -qarch=auto -qtune=auto -qmaxmem=-1.

⁵ The size of the TB matrix is surely in this range, in a typical large scale TBMD simulation.