A Specialized Data Management System For Parallel Execution of Particle Physics Codes

Jean L. Bell
Department of Mathematical and Computer Sciences
Colorado School of Mines
Golden, Colorado 80401

ABSTRACT

The specialized data management system described in this paper was motivated by the need for much more efficient data management than a standard database management system could provide for particle physics codes in shared memory multiprocessor environments. The special characteristics of data and access patterns in particle physics codes need to be fully exploited in order to effect efficient data management. The data management system allows parameteric user control over system features not usually available to them, especially details of physical design and retrieval such as horizontal clustering, asynchronous I/O, and automatic distribution across processors. In the past, each physics code has constructed the equivalent of a primitive data management system from scratch. The system described in this paper is a generic system that can now be interfaced with a variety of physics codes.

1. Introduction

A specialized data management system (DMS) has been designed and implemented to permit efficient access to particle data during the execution of particle physics programs on shared memory multiprocessor systems. The system allows the user to control parameters of the data management system not usually available to them, especially details of physical design and retrieval. The key features of the specialized system include the specification in one command of an entire sequence of queries, the specification of block size and composition for horizontal clustering of retrieved tuples, asynchronous I/O and prefetching into processor buffers according to pre-defined access patterns, automatic assignment and reassignment of particle tuples to parallel processors, efficient access to particles according to spatial parameters not stored in the tuples themselves, and organization for multidimensional retrieval of continuous (as opposed to discrete) domains.

Section 2 in this paper gives an overview of the target applications. Section 3 discusses the additional requirements for data management placed by the parallel processing environment. In section 4, the motivation for a specialized system is explained. The key features of the system are described in section 5, and the user interface is given in Section 6. Section 7 gives future directions for this research and the conclusions about such systems.

2. Overview of particle physics applications

Particle physics codes are computer simulation models of the movement of materials in three dimensional space. Their many uses include modeling of model bomb explosions, laser fusion, mixing of fluids, and other fluid motion. The material motion is simulated by tracking representative particles as they move through a three dimensional spatial domain (Figure 1). The typical particle physics code today tracks several million particles across a grid of size 200 by 200 by 20, and future simulations will track billions of particles across grids of 1000 by 1000 by 1000. This leads to very large data volumes, on the order of 10^7 to 10^{10} bytes of data in a single snapshot of the physical simulation. One physics simulation may take up to 200 hours of execution time on a CRAY-1 computer. The codes are thus highly data intensive as well as calculation intensive. However, the access patterns in the codes are very regular, so that specialized data management tailored to these regular patterns can lead to much greater efficiency.

The WAVE code is a typical example of particle physics applications [Forslund]. This code uses a numerical approximation method known as particle-in-cell to simulate the motion of particles. The code consists of a iterative

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and / or specific permission.

© 1988 ACM 0-89791-268-3 / 88 / 0006 / 0277 $1.50

metadata is usually hard-coded in the application.

2.2 Access patterns

The database for a particle physics code stores the state vector of the numerical computations. That is, values of each particle and grid cell are stored for the current timestep. Actually, two entire database versions are stored: for the current and for the previous timestep. All values in the current database are updated during each timestep to form the next database version. Over the duration of thousands of timesteps, billions of updates are performed per scientific simulation. Hence there is a critical need for efficiency.

Data dependence analysis shows that the update of one particle tuple is dependent on data in the particle tuple itself and the set of tuples for the grid cells "near" the particle's physical location. "Near" is usually defined as the cell in which the particle is centered and the set of the cells physically adjacent to that cell. A particle update is not dependent on any other particles during its movement. However, it may spawn new particles, e.g. through fission, for which new particle tuples are created. A particle may also disappear, in which case its tuple is removed from the database.

Given the access pattern for a particle update, the basic query in the particle update phase of WAVE is the retrieval of a particle and the "near" grid cells to which it is joined (Figure 5). The timestep is then viewed as a predetermined sequence of queries, so that the all particles are retrieved and modified in each timestep (Figure 2).

Since all particles are updated in each timestep, an alternative view of the basic query in WAVE is a single retrieval of the entire particle relation (actually, of the entire temporary relation consisting of each particle tuple joined to the tuples of nearby grid cells). The definition of an atomic transaction is, in fact, an update sweep through all particles; i.e. perform the entire timestep or none of it. However, the entire relation will usually not fit into central memory; so, the relation is horizontally clustered for transfer from disk to main memory. The user needs to have control over horizontal clustering, a feature not usually available in database management systems. This feature is discussed in Section 5.

The current mode of operation for most particle physics codes is batch processing. The application code requests data in a predetermined order. Scientists are beginning to introduce interactive processing for particle physics codes. For instance, they would like to be able to stop the simulation, change a few particle values and then restart the simulation. Hence, in the future there will be an increasing need for random access as well as batch processing.

3. Parallel processing

The particle update phase can be parallelized very easily on a shared memory multiprocessor by horizontally fragmenting the particle relation on each processor. Each processor updates the particle tuples in its horizontal fragment. For example, on a four-processor CRAY-XMP each processor will update approximately one-fourth of the particles. Particle updates are independent of each other, so no message passing or synchronization of particle data is necessary. Grid data resides in shared central memory, where it is accessible by all the processors.

The particles can be partitioned into subsets in many different ways. The simplest strategy is to divide the particles arbitrarily into NPROC groups, where NPROC is the number of parallel processes. Each process manages its own subset of particles. This "arbitrary subset" strategy works quite well in actual trials on a CRAY-XMP. Very little overhead is introduced for partitioning the particle tuples, and there is no need to transfer particle data among processors.

In a parallel processor with a large shared central memory, the arbitrary-subset strategy will require concurrency control for the update of charge and current attributes of grid cells, since each processor may be simultaneously updating the same grid cell. In parallel codes, concurrency control is usually effected via the use of critical sections to guard the concurrent updates. Overhead is introduced for the critical sections. Also, even accesses to the grid attributes which are not updated during particle movement phase (e.g. electromagnetic fields) will still introduce some overhead for memory access conflicts.

On a parallel architecture with a local memory for each processor, or if the central memory is not big enough to allow the entire grid to be memory resident, then the arbitrary-subsets strategy will require swapping of grid data. Under these circumstances, the strategy

---

```
SELECT *
FROM  PARTICLE, GRID
WHERE particle_cell_location <= grid_cell_identifier +1
  AND particle_cell_location >= grid_cell_identifier -1
```

Figure 5. Query to retrieve a particle and the grid cells near it

is very inefficient. Instead, each processor is assigned the particles in a particular region of the grid. That is, the particles are clustered and fragmented horizontally by grid region. Each regional grouping of particles is assigned to a different processor, which now need access only to the grid cell data for its grid region.

Using this "subgrid" strategy, there can be no concurrent updates of the grid, so the updates need not be guarded. Memory conflicts are avoided, and overhead for memory references goes down. However, there are compensating extra sources of overhead. First, the overhead for managing the particles increases. A particle may move from one region to another, so its particle tuple must be transferred to the processor handling its new region. Also, overlapping grid regions will be required so that a particle in a cell on a region boundary will have access to its adjacent cell in a neighboring region. Finally, particles are not equally distributed across the grid, so some regions will have many more particles than others; this leads to load imbalance among processors. To restore load balance, dynamic adjustments in the region boundaries must be made.

Partitioning strategies for particle and grid data are beyond the scope of this article. The point is that the access patterns for particle physics codes become more complex when the code is ported to a different machine architecture. Movement across memory hierarchies and among parallel processors complicates the data management task.

Another issue in a parallel environment is whether the data manager should be centralized or decentralized. A centralized server, e.g. via a database machine [Dewitt 79, Hawthorn 82], consolidates the data management, but also creates a bottleneck. Since particle physics application codes are already I/O bound, efficiency demands that each processor have parallel access to data. Hence, our specialized DMS takes the decentralized approach found in distributed systems [Stonebraker 86]. Each processor manages its own database of particles, which are mostly accessed only by itself. Operations for data transfer among processors are also necessary, e.g. when a particle crosses a grid region boundary.

4. Motivation for a data management system

A data management system is desirable for particle physics applications for the same reasons that one is desirable for any other applications. Primary reasons include the desire for efficient software; the economies of using general purpose, standardized software packages, e.g. not having to write the data management procedures over again for every application code; and, portability across computer and storage architectures. The data management system must provide all of the standard database functions: create the database; insert, delete, modify, and retrieve tuples; and, perform physical organization tasks (including both access methods within a processor and across distributed processors). In this section, we discuss the motivation for using a data management system. In Section 5, we describe the features missing from a standard database management system which are needed in particle physics applications.

An obvious question is whether a simpler file management package would suffice. A file manager would not be sufficient for most particle physics applications, for the following reasons. The primary reason is that managers generally can access data only in sequential storage order, and only the simplest particle physics codes limit themselves to a single fixed order of access. Particle accesses are multidimensional, by species (e.g. all electrons are accessed together, for compression and vectorization reasons) and by grid location. Grid location is dynamic, so a fixed particle storage order is impossible. After a particle movement is completed, its tuple must be inserted into a new file location, based on the particle's new spatial location.

Second, file managers do not generally perform joins between files, but particle physics simulations require coordinated access to grid and particle data. Third, file managers do not coordinate access across processors in a distributed parallel environment. Fourth, file managers force the separate handling of metadata, and scientists are increasingly aware of the advantages of coordinated access to metadata. Finally, file managers usually handle only a few simple physical file organizations: but, efficient storage and access to particle data requires sophisticated techniques such as inverted files and horizontal clustering.

To be sure, some database functions can be simpler for these applications than is needed in a general purpose database management system (DBMS). Specifically, recovery management, concurrency control, integrity constraints and security are much simpler. Integrity constraints exist in the code, not in the DBMS. Security is not an issue, since the data management runs inside the address space of the application, for performance reasons. Recovery is much simpler since the database consists of the state vector for the physics calculations. Recovery does not require transaction logs and rollback/rollforward. Instead, a full backup is taken periodically at the end of a timestep, and these backups are archived for later analysis or graphic (movie) presentation. If needed for recovery, the application is simply restarted from the last state saved.

Concurrency control is also much simpler than in a general purpose DBMS, because the database is created for a single related set of processes with highly circumscribed access. The particles are clustered into independent fragments, so no concurrent access is possible. Only the grid relation is concurrently updated, and the only update operation on grid data during the particle phase is accumulation. Since accumulation is a commutative operation, the constraints on concurrent access can be relaxed:

order of updates is not important, just the integrity of each update. The notion of a transaction reduces to the timestep, which sweeps through and updates the entire particle relation.

5. Key features for a specialized system

Given the need for a database management system, the next question is whether a commercially available system will be adaptable for particle physics codes. The capabilities needed by particle physics applications are certainly all present in distributed database management systems, e.g. INGRES [Stonebraker 86]. The problem with these systems is that they were optimized for very different kinds of applications, specifically, for random access by tuple identifier. Key optimization features needed for particle physics codes are missing from standard databases management systems, including: transfer of data in large blocks under user control, asynchronous I/O, efficient handling of array data, efficient handling of complicated joins, and automatic physical organization for parallel processing.

5.1 Data transfer in large blocks

Transfer of one particle record at a time is obviously inefficient because of startup time for each transfer request. To avoid the delay for each tuple transfer, particle tuples are grouped into large blocks. In particle physics codes, a block size on the order of 10,000 particle tuples (480,000 bytes) is not unusual. A data management system must contain procedures to create the buffer space, to transfer the blocks and to divide them into logical records.

Also, the data management system must allow user commands for specification of horizontal clustering, including the appropriate block size and characteristics for grouping data into units of transfer. For example, user priorities may dictate the use of small buffers in one execution, and large buffers in another execution. Similarly, one application code may need blocks of particles grouped only by particle species, and another application may need blocks grouped only according to spatial region.

5.2 Efficient prefetching

One of the primary sources of efficient I/O in particle codes is that the order in which the data will be read can be established prior to the request for transfer. That is, the every particle tuple in the database will be read exactly once during each timestep, in a preestablished order. Even though the retrieval order may be different than sequential storage order, the fact that the retrieval order is predetermined allows for asynchronous prefetching of data into multiple buffers. Prefetching reduces waiting for I/O, and hence increases efficiency and decreases total response time for the application code. Furthermore, these codes are often run as the sole application on a supercomputer, i.e. not timeshared with other codes. In that environment, reducing the wait for I/O through prefetching also increases the throughput of the system. A data management system tailored for this application class must, therefore, include a prefetching feature.

To accomplish the prefetching, the data management system must have the capability of processing asynchronous requests. Most database management systems are set up for random access to individual tuples. There is no concept of asynchronous request because the applications for which these systems were designed cannot anticipate the order in which data will be requested. On the other extreme, in a sequential file management system, prefetching can be done, but the only allowed order is the physical sequential storage order. For particle physics codes prefetching is required according to some user-specified order that is not necessarily physically sequential.

5.3 Relational join based on calculated values

The basic access pattern during a particle update is its particle tuple together with the tuples for grid cells near its location. This is an ordinary join of particle to cell relations, which can be easily expressed, e.g. in Figure 5 in pseudocode similar to SQL. However, the implementation of join is complicated by the non-correspondence of identifier domains in the PARTICLE and GRID relations. The key of the particle relation is a particle's location coordinates in three dimensional space (X, Y, Z), which are in the continuous domain of real numbers. On the other side of the join, the grid cell identifier is its integer array subscripts (I, J, K).

Thus, the join must be preceded by a derivation that transforms the particle identifier into discrete array subscripts, based on information (usually hard-coded into the application) about the interval in real space represented by each grid cell. A new, intermediate relation must be constructed for the particle data that includes its cell identifier, which was calculated from its location identifier. It would be obviously inefficient to have to store both real space and cell identifiers for millions of particles. It would be equally inefficient to have a separate pass through the particle data to create an actual temporary relation. What is needed is an operation which allows complicated calculations in the specification of a join.

5.4 Handling of arrays

As mentioned above, the GRID relation is compressed using array linearization techniques, where the grid cell identifier (I, J, K) is not explicitly stored. In the Fortran code, this is no problem, since the grid cell subscript is used directly to locate its tuple in the linearized array. In relational systems, an array is stored either as single atomic object, or as an entire relation of array element tuples. The first choice does not permit access to individual cells or subarrays, and the second choice promotes inefficient storage and access by

neglecting the locality in the data.

Our experimental data management system recognizes this problem and creates an intermediate level of objects. The grid is broken into user-defined regions, and the relation is over grid regions. Each region is separately retrievable, identified by the cell identifier range associated with the region (or, by its ordinal region number). Within each region, the storage order for grid cells preserves the subscript mappings.

5.5 Database creation and dynamic reorganization

In most database applications, the database is created and then used without the need for restructuring or re-creation for a long period of time. In contrast, particle physics applications create a database of particles for each simulation, i.e. for each execution of the code. The data management system must be able to create the database quickly using the data generated by the program, and it must be organized efficiently for the sole application which will use it.

Efficient database generation is based on the regularities in the data itself. The particles' initial characteristics and locations are generated by the application code based on parameters set by the user. In particular, the particles are usually placed across a grid in some regular pattern. The specialized data management system contains commands to direct physical data organization of the database.

Furthermore, the physical placement of tuples will need to be reorganized frequently to maintain large transfer blocks according to the spatial and particle species groupings discussed above. The data management system will oversee automatic reorganization. That is, whenever a block of data are updated and returned to the database for storage, the data management system will remove the tuples of particles that have moved out of the spatial region of that the block, and it will insert the removed tuples into a block where they belong.

The databases in particle physics are so large that they are nor likely to fit into central memory in their entirety, even on the supercomputers on which they typically run. However, a significant fraction of the database may be in central memory at any given time. Also, solid state memory (e.g. the "SSD" on the CRAY-XMP), is attachable to extend the central memory in a relatively transparent way. Our experimental data management system incorporates new possibilities for database organization and access being developed for memory resident databases [e.g. Dewitt 84, Lehman 86].

6. The user interface

The interface to the specialized particle data management system is similar to the host language interface available in many standard DBMS. The user calls on the data management system to perform the desired tasks: creation, retrieval, update, and reorganization. Details of some commands are discussed in this section to illustrate the user interaction and the parameters available to the user.

6.1 Creation

The initial creation of the database begins with the generation of the schema, using the SETPCL and SETGRID commands. If running on a parallel processor architecture, the command DIVGRID is used to inform the DMS of the desired horizontal partitions for each processor. Finally, the retrieval pattern for particle tuples is described via the PREORDER command.

The PREORDER will specify a list of attributes in major sort order, e.g. to retrieve in ascending order by ranges of X locations of the particles (corresponding to grid planes), and within X location in order by particle species. The prespecified order will be later be used for the asynchronous scan through the entire set of particle tuples during retrieval. The retrieval order is described in the creation phase so that the database can be organized in accordance with the retrieval pattern. In the future, we may be able to automatically extract the relevant details from the code itself, but in the current implementation, the user must inform the DMS of the retrieval pattern.

After schema creation, the application program generates particle tuples and sends them in blocks to the DMS via the PUTPCL command. At this stage, the DMS simply writes them on a sequential file. After all the particle tuples have been generated, a call to the DMS via the ORGPCL command organizes the particle tuples on direct access files, using the organizational directives already given.

6.2 Storage, Retrieval, and Reorganization

Within a timestep in the particle update phase of the application program, there is a loop to process all blocks of particle data. The first command in the loop is GETPCL, which retrieves the next block of particle tuples from the current version of the database. The block is placed into a user array whose address is specified as a parameter in the GETPCL command. After the entire block has been updated, the STOREPCL command writes the block into the new version of the database.

Note, however, that the actual transfer of data occurs asynchronously prior to the issuance of a GETPCL command. That is, the DMS keeps its multiple buffers full of unprocessed data, waiting for a GETPCL request. Whenever data are written from a buffer, the DMS automatically requests a new bufferful to be input from the disk. Since the order is prespecified, this strategy will always provide the data in the required order, and with minimal I/O wait time.

An alternative to the GETPCL and STOREPCL is the "datapass" construct similar to that used for grid regions in the Flotran data management system [Anderson 77]. The datapass construct is

a loop over all tuples in a relation, in a prespecified order; it encompasses the loop over all blocks, the GETPCL at the beginning and the STOREPCL at the end. We considered implementing the DATAPASS, but decided to go with the individual GET and STORE commands because of the additional flexibility that they offer. That is, if necessary a block could be retrieved individually, and not as part of an entire pass through all blocks.

The GETGRID and STOREGRID commands get and store grid regions asynchronously in the same order prespecified for fetching particles. Alternatively the user can place the grid data in shared memory and manage it himself, in which case no special retrieval command is needed.

The system performs much of the data management automatically, without explicit commands. The level of automation is made possible because the system is tailored for the application class and environment. For example, no explicit command to delete particle tuples is needed, since the application program can itself delete the tuple from the block in which it was located. No tuple insertion command is needed, for the same reason. The new tuple is simply included in a block being passed to the data management system.

Similarly, no explicit command is provided for reorganization of the particle tuples. Instead, the DMS automatically analyzes the spatial locations of the tuples passed to it for storage by the STOREPCL command, and removes the tuples not belonging in a block (because the particle has left the spatial region represented by the block). The removed tuples are eventually consolidated with a block in which they belong. No reorganization command is needed because the data management system is designed to continuously reconfigure physical organization according to the attributes in the PREORDER command, e.g. particle locations.

7. Conclusions and Future Directions

The specialized data management system described in this paper was motivated by the need for much more efficient data management than a standard DBMS could provide for particle physics codes. The special characteristics of data and access patterns in particle physics codes need to be fully exploited in order to effect efficient data management. In the past, each physics code has constructed the equivalent of a primitive data management system from scratch. The system described in this paper is a generic system that can now be interfaced with a variety of physics codes.

An initial, limited version of the data management system described above has been implemented on a CRAY-XMP four processor system. It is now being interfaced with the WAVE particle physics code from Los Alamos. Initial experiments are promising: the interface between the DMS and the application code was easy to program, and the code runs relatively efficiently with the DMS. Our intent in this paper is to explain the system motivation and design. Results of using the system with a variety of physics codes will be reported in a subsequent paper.

Future research directions for the specialized DMS include further work on storage organization. In the initial implementation, the DMS does not store or retrieve the grid data, since the central memory of the CRAY-XMP is large enough to easily accommodate the entire grid. Research is needed to optimize grid storage and retrieval when the grid is stored out of memory. Also, in the initial implementation a relatively simple inverted file structure is used for the particle data on external storage. Experimentation with several different underlying storage strategies, especially grid files [Nievergelt 84], is in progress.

Future work also includes experimentation with parallel architectures with local memory to determine the suitability of the DMS design for radically different architectures. The handling of grid data, in particular, may need to be redesigned because of the much slower data transfer rate among local memories.

The advantages for the application programmer of using the new DMS are those for using any standardized software. First, costs of developing the data management system will be saved. Application development costs will also be lower because of the the modular, high-level interface to the data management routines. Second, the package was designed for portability. In the case of parallel environments, the savings by using a portable DMS are especially significant. Since I/O is among the least portable part of most applications, the isolation of the I/O functions is itself a major step towards program / device independence. Finally, the I/O at execution time may be more efficient than hand-coded data management, since the underlying data structures will be more optimal than non-specialists would use. In individual cases, of course, some speed may be sacrificed because a more general-purpose package is being used. However, initial results indicate that the penalty for generality is very small.

This research was supported in part by a grant from the IBM Corporation.

REFERENCES

Anderson 77
Anderson, D.L. "FLOW User Manual", National Center for Atmospheric Research, Boulder, Colorado, July 14, 1977.

Bell 82
Bell, J.L. "Data Modelling of Scientific Simulation Programs", Proc. SIGMOD International Conf. on Management of Data, (ACM) Orlando, Florida, 1982, pp. 79-86.

Bell 87
Bell, J.L. and Patterson, G.S. Jr. "Data Organization in Large Numerical Computations", The Journal of Supercomputing, 1, 1, 1987, pp.105-136.

Buneman 80
Buneman, O., Barnes, C.W., Green, J.C. and Nielsen, D.E. "Principles and Capabilities of 3-D, E-M Particle Simulations", Journal of Computational Physics 38, 1 (Nov. 1980), pp. 1-44.

Cray 86
"Cray Computer Systems Technical Note: Multitasking User Guide", Cray Research, Inc., Mendota Heights, Minn., 1986.

Dennis 80
Dennis, J.B. "Data Flow Supercomputers", IEEE Computer, November, 1980, pp.48-56.

Dewitt 79
Dewitt, D.J. "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems", IEEE Transactions on Computers C-28, 6 (June 1979), pp. 395-406.

Dewitt 84
Dewitt, D., Katz, R., Olken, F., Shapiro, L., Stonebraker, M., and Wood, D. "Implementation Techniques for Main Memory Database Systems" Proc. SIGMOD International Conf. on Management of Data (ACM) Boston, 1984, pp. 1-8.

Forslund
Forslund, D. W., The Wave Reference Manual, Los Alamos National Laboratory, Los Alamos, N.M. undated.

Harlow 71
Harlow, F. H. and Amsden, A. A. "Fluid Dynamics", LASL Monograph LA-4700 UC-34, Los Alamos Scientific Laboratory, University of California, Los Alamos, N.M., June 1971.

Hawthorn 82
Hawthorn, P. and Dewitt, D.J. "Performance Analysis of Alternative Database Machine Architectures", IEEE Transactions on Software Engineering SE-8, 1 (Jan. 1982), pp. 61-75.

Horowitz 77
Horowitz, E. and Sahni, S. Fundamentals of Data Structures, Computer Science Press, Inc., Potomac,Md., 1977.

Kent 79
Kent, W. "Limitations of Record-based Information Models", ACM Transactions on Database Systems, 4, 1, (1979) pp. 107-131.

Lehman 86
Lehman, T.J., and Carey, M.J. "Query Processing in Main Memory Database Management Systems", Proc. SIGMOD International Conf. on Management of Data (ACM) Washington, D.C. 1986, pp. 239-250.

Nievergelt 84
Nievergelt, J., Hinterberger, H. and Sevcik, K.C. "The Grid File: An Adaptive, Symmetric Multikey FIle Structure", ACM Transactions on Database Systems, 9, 1, (1984) pp. 38-71.

Ozsoyoyoglu 85
Ozsoyoyoglu, G., Ozsoyoyoglu, M., and Mata, F. "A Language and Physical Organization Technique for Summary Tables", Proc. SIGMOD International Conf. on Management of Data (ACM) Austin, Texas, 1985, pp. 3-16.

Shoshani 84
Shoshani, A., Olken, F., and Wong, H.K.T. "Characteristics of Scientific Databases", Proc. 10th International Conf. on Very Large Databases (VLDB), Singapore, 1984, pp. 147-160.

Stonebraker 86
Stonebraker, M. "The Design and Implementation of Distributed INGRES", in The INGRES Papers: Anatomy of a Relational Database System, M. Stonebraker (ed.), Addison-Wesley, 1986, pp. 187-196.

```
FOR t= 1 to Number_of_Timesteps

                              /* particle update phase */
    FOR p = 1 to Number_of Particles

       Retrieve Particle(p) and nearby gridcells(p_position)
       Particle(p).new_location = F(Gridcell(p_position) and
                                    Particle(p), at time=t-1)
       Update Particle(p)
       Gridcell(p_position).count = Gridcell(pposition).count +1

       Gridcell(p_position).current= Gridcell(p_position).current
                                    + Particle(p).current

       Gridcell(p_position).charge= Gridcell(p_position).charge
                                    + Particle(p).charge

    ENDFOR

                              /* grid update phase */
    FOR g= 1 to Number_of_Gridcells

       Gridcell(g).electromagnetic_fields =
              F(Gridcell.count, gridcell.current, gridcell.charge
                for all Gridcells)

   ENDFOR

ENDFOR
```

Figure 2. Pseudocode overview of the WAVE code.

```
PARTICLE(Species,Time, Xlocation,YL,ZL, Xvelocity,YV,ZV)

Notes: 1. No unique identifier.
       2. Species and time are factored out of inverted
          files,not actually stored.
```

Figure 3. Particle relation in particle physics codes.

```
GRID(I,J,K,Time, Xelectric,YE,ZE, XMagnetic,YM,ZM,
                 Particle_count, charge, Xcurrent,YC,ZC)

Notes: 1. I,J,K,Time are the primary key.
       2. I,J,K,Time are factored out of inverted files, not
          actually stored.
```

Figure 4. Grid cell relations in particle physics codes.

irregular, dense, and have high time variation. Hence the need to store particle coordinates at every time interval. There is very little meta-data about particles. There is often associated data, for example, the characteristics of a species (e.g. the speed at which an electron moves, or the probability of an ion splitting through fission). The associated data are usually hard-coded in the calculations. For some application codes, e.g. Monte Carlo, large tables of species characteristics are stored in a separate database.

In contrast, the grid is stationary (no time variation), contains equally spaced grid cells (very regular), and all cells in the matrix represent data points (very dense). Because of its regular, dense qualities, the grid maps directly onto a three dimensional array. The array subscripts (I, J, K) comprise the identifier for each grid cell tuple. Array linearization [Horowitz 77], directly available in Fortran, is then used to avoid storing the array subscripts explicitly. Similar data compression schemes have been used in statistical databases, which also have array-like characteristics [Ozsoyoyoglu 85].

Metadata about the grid includes the transformation between spatial and grid scales, e.g. how much space is represented by each grid cell. As with particle metadata, the grid

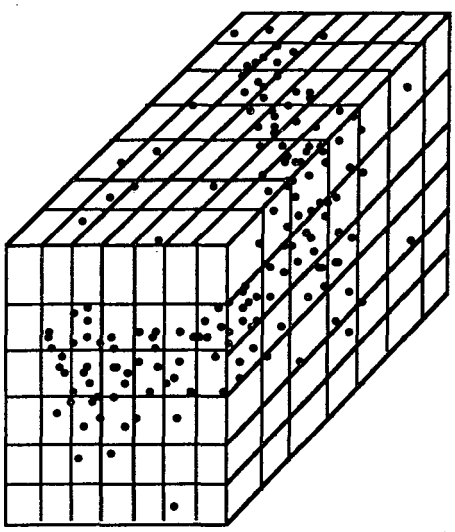

Figure 1. Particles moving through a three dimensional grid.

cycle, where each itration predicts new particle positions based on the positions in the previous cycle. Each cycle, or "timestep", advances the simulation one small step forward in time. There are thousands of timesteps in a single execution of the code.

Each timestep consists of two phases, shown in pseudocode in Figure 2. During the particle update phase, the particles are all moved to new grid locations, using laws of motion and electromagnetism to calculate each particle's direction and extent of motion. At the same time, the grid cell tuples through which the particle passes are updated to accumulate the contribution of the particle to the total count, charge and current for those grid cells. After the particle update phase is finished, the grid update phase calculates new electromagnetic fields based on the final charge and current accumulated for grid cells. The new electromagnetic fields represent the collective behavior of the particles for the timestep. The new electromagnetic fields are then input to the particle update phase of the next timestep, and the cycle repeats itself.

Our studies have concentrated on two types of physical models: particle-in-cell [e.g. Buneman 80, Harlow 71], and Monte Carlo. [Bell 87] details the workings of several of these models, and the relationship between numerical methods and data structures. Here, we give only enough background to motivate the specialized data management system. Furthermore, we focus only on the particle update phase of the applications, because it is the management of particle data which creates the most difficulty.

2.1 Data Content

The WAVE application generates two types of experiment data: data about individual particles, and data about a cells in a fixed grid superimposed on the spatial domain. The PARTICLE relation (Figure 3) includes spatial coordinates of the particle (X, Y, Z), and characteristics such as particle species type (e.g. electron) and amount of charge. Note that the PARTICLE relation is not fully relational in that the particle identifier is not necessarily unique. That is, two particles could have the same coordinates. The non-uniqueness is not a problem, since particles with duplicate identifiers are in a real sense duplicates.

The grid stores aggregate information about the state of the physical system. The grid exists to simplify particle movement, by eliminating the need to separately calculate a particle's interactions with each nearby particle. Instead, the movement of a particle is controlled by the electromagnetic field in its grid cell, which represents the aggregate influence of the nearby particles. All attributes of a grid cell are facts aggregated over the entire cell, such as average density of particles within the cell, average current, and average electromagnetic field (Figure 4).

Using the terminology developed in [Shoshani 84], the particle data are very