Chapter 1

The Charm++ Programming Model

Laxmikant V. Kale

Department of Computer Science, University of Illinois at Urbana-Champaign

Gengbin Zheng

National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign

CHARM++ [131] is a C++ based parallel programming system developed at the University of Illinois. It has been designed and refined in the context of collaborative development of multiple science and engineering applications, as the later chapters in this book illustrate. The signature strength of CHARM++ is its *adaptive runtime system*, which allows programmers to deal with increasingly complex supercomputers and sophisticated algorithms with dynamic and evolving behavior. Its basic innovation is the idea of *over-decomposition* (explained further in Section 1.2): the programmer decomposes the computation into objects rather than processors, and leaves the decision about which object lives on which processor to the runtime system. Specifically, some of the benefits of CHARM++ to the programmer include:

- *Processor-independent programming:* The programmer decomposes the computation into logical units that are natural to the application, uncluttered by the notion of what data is found on which processor, and which computations happen on which processor.

- *Asynchronous programming model with message-driven execution:* Communication is expressed in a highly asynchronous manner, without opportunities for the program to block the processor awaiting a remote event. This model is supported by message-driven execution, where the processor-level scheduler selects only those computations for which all data dependencies have been satisfied.

- *Automatic communication/computation overlap:* Without any explicit programming effort, the CHARM++ runtime ensures that processors are not held up for communication, and that the communication is spread more uniformly over time. This leads to better utilization of the communication network, and to programs that are highly tolerant of communication latencies.

- *Load Balancing:* The CHARM++ runtime automatically balances load, even for applications where the load evolves dynamically as application progresses. It can handle both machine-induced and application-induced load imbalances.

- *Resilience:* CHARM++ applications can be automatically checkpointed to disk, and restarted on a different number of processors, within memory limits. Further, CHARM++ applications can be made automatically tolerant of node failures, automatically restarting based on in-memory checkpoints when the system detects a failure, on machines where the schedulers will not kill a job if one node dies.

The purpose of this chapter is to introduce the basic concepts in CHARM++, and describe its capabilities and benefits for developing complex parallel applications using it. The next chapter illustrates the process of designing applications in CHARM++ with choices and design decisions one must make along the way. A more thorough tutorial on how to design CHARM++ applications can be found elsewhere. For example, for online resources, see `http://charm.cs.illinois.edu`.

1.1 Design Philosophy

To appreciate the features of CHARM++, it is necessary to understand the main design principles that were used as guidelines when developing it.

The first of these principles has to do with the question: *what aspects of the parallel programming task should the "system" automate.* The design of CHARM++ is guided by the idea of seeking an *optimal division of labor* between the programmer and the system, i.e, we should let the programmers do what they can do best, and automate those aspects that are tedious for the programmer but relatively easy (or at least feasible) for a system to automate. Some parallel programming systems are designed with the view that the system should simply provide minimal mechanisms, such as basic communication primitives, and get out of the way. This has the advantage that the application developers are least constrained. An alternative is presented by the ideal of a perfect parallelizing compiler: the programmers write (or better still, just brings their own dusty deck) sequential code, and the system auto-magically

parallelizes it effectively for the target machine. The former approach is inadequate because it does not raise the level of abstractions, while the latter has been seen to be unrealizable, despite valiant efforts. *Seeking an optimal division of labor* between the application programmer and the system has led to foundational design features in CHARM++.

The second design principle is to develop features only in an application-driven manner. This is to counter a common and natural tendency among computer scientists toward a platonic approach to design, which one could call *design in a vacuum*: features are developed because they appear beautiful to their developers, without serious consideration of their relevance to a broad set of applications. To avoid this, CHARM++ evolved in the context of development of parallel science and engineering applications, and abstractions or features were added to it when the application use cases suggested them [118].

1.2 Object-based Programming Model

It is important to note that although CHARM++ is certainly a novel and distinct parallel programming model, different than prevailing models such as MPI, it is **not** a different "language" – code is still written in C++[1]. Programmers are only required to provide declarations of the methods that are meant to be invoked remotely, so that the system can generate code to pack and unpack their parameters automatically. Beyond that, one uses the C++ API provided to make calls into the CHARM++ runtime system.

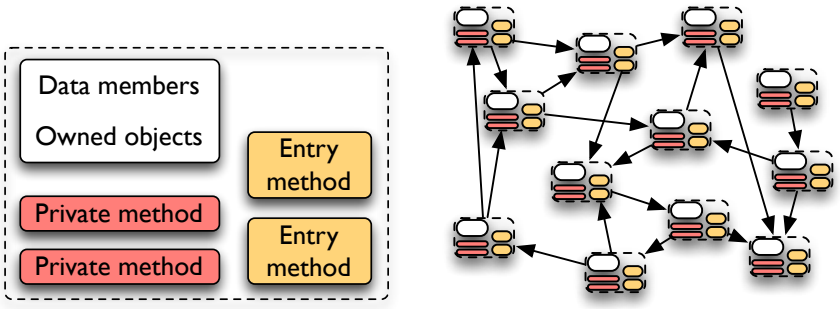
The basic innovation in CHARM++ is that the computation is broken down by the programmer into a large number of objects, independent of the number of processors. These objects interact with each other through asynchronous method invocations (defined below). In such interactions and communications, it is the objects that are named explicitly and not the processors; the program is mostly free of the concept of a *processor*. This empowers the adaptive runtime system at the heart of CHARM++ to place objects on processors as it pleases, and to change the placement during the execution of the program. This *separation of concerns between the application logic and resource management* is at the heart of many benefits that this programming model confers on application developers.

These *"migratable"* objects, which are the units of decomposition in the parallel program, are called *chares*[2] in CHARM++. Of course, a CHARM++

[1] Although there exist bindings for C and Fortran, we will focus on the C++ bindings in this chapter. Most of the applications in the book are written in C++. It is also, of course, possible to write most of the application in C or Fortran by using CHARM++ to express all the parallel aspects, and calling sequential C and Fortran functions for the application specific code.

[2] The 'a' in ch**a**re (\'**tʃɑr**\) is pronounced like the 'a' in f**a**ther and the 'e' is silent.

program may also include regular C++ objects — but the runtime system does not need to pay attention to them. Each such sequential regular C++ object is "owned" by a single chare (Figure 1.1(a)). So, they migrate with the chare if the runtime system decides to migrate the chare to another processor. The programmer's view of the overall computation is that of many such chares interacting with each other, as shown in Figure 1.1(b).



(a) A single chare (C++ object)

(b) Programmer's view: Collection of interacting chares

FIGURE 1.1: Chare: a message-driven object

Let us examine a chare in isolation first, as shown in Figure 1.1(a): it is a C++ object comprising data elements and private or public methods. Its public methods are remotely invocable, and so are called "entry" methods. It is the existence of these entry methods that distinguishes a chare class from a plain C++ class. Chares can directly access their own data members, and cannot usually access data members of other chares. In that sense, a chare can be thought of as a processor as well. Since, typically, multiple chares live on a real processor, we can call them "virtual" processors. Consequently, we have called our approach the *"processor virtualization"* approach [127]; however, it is important to note that it is significantly different than (but related to) the relatively recent idea of *OS virtualization* made popular for the "cloud" infrastructure by VMWare and Xen systems.

Asynchronous method invocation: A running object may execute code that tells the runtime system to invoke an entry method on a (potentially) remote chare object with given parameters. The programmer understands that such method invocation is asynchronous: all that happens at the point where the call is made is that the parameters are packaged into a message, and the message is sent towards the chare object in question. It will execute at some undetermined point in future. No return values are expected from an asynchronous method invocation. If needed, the called chare will send a method invocation to caller at some time in the future.

The **execution model**, from the point of view of the programmer, is as

follows: an ongoing computation consists of a collection of chare objects and a collection of entry method invocations that have been directed at these objects. The computation begins with construction of a designated *main chare*. The user code in the constructor of the main chare may initialize the *Read-only variables* . These should not be changed by user code afterwards. The runtime system makes a copy of such variables available on each processor. The constructor of the main chare typically contains user code that create chares and collections of chares (see below), and thus seeds the overall computation. On each processor, a message driven scheduler (Figure 1.2) in the runtime system selects one of the entry method invocations targeted at some object residing on its processor, unpacks the parameters for the invocation, and executes the entry method on the identified object with the given parameters. In the baseline model, it lets the method invocation continue to completion (see Section 1.4 for exceptions), at which point control returns back to the runtime scheduler. Since the asynchronous method invocations can be thought of as messages, this aspect of the execution model is called *message-driven execution*.

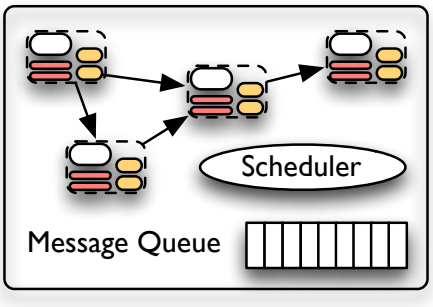


FIGURE 1.2: Message-driven scheduler

One of the major benefits of message-driven execution is an automatic and adaptive overlap between communication and computation. There is no call in CHARM++ that will block the *processor* waiting for some remote data. Instead, control passes to some chare that already has some data waiting for it, sent via the method invocation from a local or remote object. The time an object is waiting for some communication from its remote correspondent is thus naturally overlapped with computation for some other object that is ready to execute.

A chare normally just sits passively. Whenever a method invocation (typically initiated asynchronously by some other chare) arrives at a chare, it executes the method with the parameters sent in the invocation. This may result in creation of some new asynchronous method invocations for other chares (or even itself) that are handed over to the runtime system to deliver. It changes its own state (i.e. values of its data member variables) as a result

of the invocation. It then goes back to the passive state waiting for another method invocation[3].

The code inside each entry method can carry out any computation it wishes using data completely local to its object. In addition, it can use data that has been declared as read-only.

From the application's point of view, a chare could be a piece of the domain (in "domain decomposition" methods used commonly in parallel CSE applications). It may also be a data structure, or a chunk of a matrix, or a work unit devoid of any persistent state. The programmer is responsible for deciding how big the chare should be, viz. the *grainsize* of the chare. More on that in the next chapter.

One can create a singleton chare instance dynamically, and the system will decide on which processor to anchor it. All that happens at the call is that a *seed* for the new chare is created, which captures the constructor arguments for it; a seed-balancer dynamically moves the seeds among processors for load balancing, until it is scheduled for execution on some processor by executing its constructor, at which point we can assume that the chare has *taken root* there. Chares can obtain their own global IDs (called *proxies* in CHARM++), and methods can be invoked asynchronously using these proxies. Parallel programming based on such dynamic creation of individual chares is useful in a variety of situations, including combinatorial search.

For applications in science and engineering, we need a further abstraction: multiple chares may be organized into a collection, and each chare belonging to a collection can be named (and accessed) by an *index*. For example, one may have a one-dimensional array of chares. One can then broadcast method invocations to all the elements of a collection, or to a single named one. These collections are called *chare arrays*. However, they are not limited to be simple arrays. The index structures may define collections that are multi-dimensional sparse structures (e.g. a 6-dimensional array, with only a small subset of possible indices being instantiated as chares). They can also be indexed by other arbitrary indices, such as strings or bit vectors, but such usage is not common in current CSE (Computational Science and Engineering) applications.

A single program may contain multiple chare arrays. These may arise from multiple application modules, or a single module whose algorithm is more naturally expressed in terms of multiple chare arrays. To communicate with chares belonging to a chare array, one must get hold of a "proxy" — an object that stands for (or refers to) the whole collection. A proxy is returned when a chare array is first created. So, the code `A[i].foo(x,y);` specifies asynchronously sending a method invocation for the method `foo` with parameters `x,y` to the i'th object of a 1-dimensional array referenced via its proxy `A`. The

[3]The model up to this point is similar to the "actor" model developed by Hewitt, Agha, Yonezawa,and others, with the possible exception of the idea of an explicit "mailbox" that an *actor* has access to. More important points of departure come in the features described after this point, and in the reliance of CHARM++ on its extensive adaptive runtime system.

call immediately returns, while the method invocation is packaged and sent to the processor where the i^{th} element resides.

Chare arrays support reductions and broadcasts over all its elements, but, unlike in MPI-1 or MPI-2, these are both asynchronous non-blocking operations. (MPI-3 standard has now adopted non-blocking collectives). A broadcast can be initiated by any element or even from other chares not belonging to the target chare array. In our example above, `A.foo(z,t)` will result in asynchronous invocations of the `foo` method of all the member chares of `A` — a broadcast. The system ensures that all the chares belonging to a chare array receive the successive broadcasts in the same sequence. Reductions are carried out via non-blocking "contribute" calls that allow other computations to proceed while the result of the reduction (such as a global sum) is delivered to its intended target, via an entry method invocation or via a general-purpose `callback` abstraction. In particular, the members of the chare array over which the reduction is being carried out are free to execute other entry methods while the reduction is in progress.

The chares belonging to a chare array are assigned to processors by the runtime system (RTS) as shown in Figure 1.3; the RTS may change this assignment at runtime as needed. A scalable location manager [153] keeps track of which chares are on which processor, resulting in messages being delivered quickly and with low overhead to the right processor.

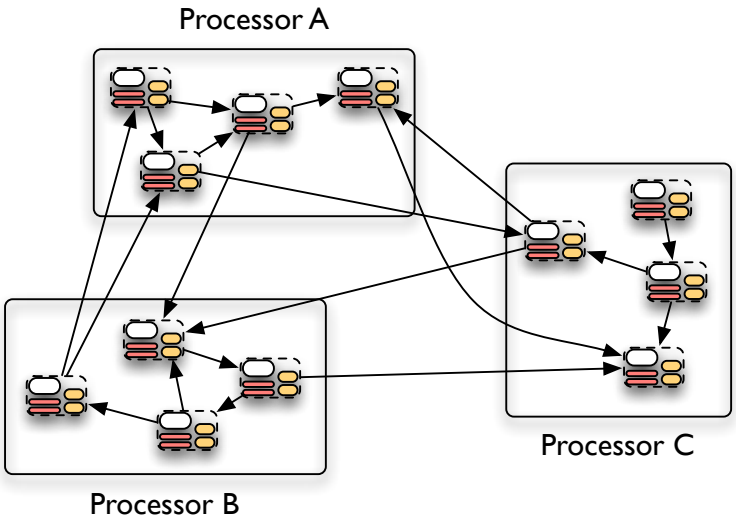


FIGURE 1.3: System view of chares

1.3 Capabilities of the Adaptive Runtime System

The heart of the CHARM++ system, and its signature strength, is its adaptive runtime system. The primary responsibilities of the runtime system are:

1. To decide which objects reside on which processor, when they are created
2. To schedule (i.e. sequence) the execution of all pending entry method invocations on the targeted chare objects on each processor,
3. To keep track of current location of each chare, in spite of chare migrations, in a scalable and low-overhead manner,
4. To mediate communication between chares by delivering entry method invocations to the correct target object on the processor where it resides. And, finally
5. To migrate chares across processors, if needed, in response to runtime conditions.

The CHARM++ programming model provides much flexibility to the runtime system, in terms of placement of chares on processors, sequencing of method invocations, and mediating and intercepting communication between chares. The CHARM++ adaptive runtime system (RTS), thus empowered, leverages this flexibility to optimize performance as the program executes. Here, we will briefly discuss its capabilities in balancing load dynamically, tolerating faults, optimizing communications, and managing power.

Dynamic Load Balancing: CHARM++ supports a large suite of load balancing strategies. Some of these strategies use measurements of computational loads and communication graph between chares, which the RTS can readily obtain because of its role in scheduling chares and mediating their communication. With CHARM++, load balancing can be thought of as a two-phase process: the programmer decomposes the work (and data) into chares. This division does not have to be perfect: i.e. significant variation in the work/size of chares is permissible, since there are typically tens of chares on each processor core. At runtime, the RTS assigns and reassigns chares to individual processors, to attain such goals as better load balancing, and/or minimization of communication volume. As the number of chares is much smaller than the number of underlying data-structures elements (e.g. grid points, or mesh elements), the load balancing decisions are much faster than, say, applying a graph partitioner such as ParMETIS to the entire underlying structure. Occasionally, chares may have to be split or merged to keep their size within a desired range; CHARM++ supports dynamic creation and deletion of chare array elements if needed. But this is not needed for most applications, and when

its needed, it is still simpler than a complete repartitioning of the application data structures.

The suite of strategies provided with CHARM++ includes some that ignore communication volume and some that consider it. It also includes some strategies that *refine* load balance, by moving a relatively small number of objects from overloaded processors, and other schemes that comprehensively repartition the object graph. For large machines, it includes strategies that optimize placement with respect to the interconnect topology, and hierarchical strategies that significantly reduce decision time. Further, one can write application-specific strategies (or new, general-purpose ones) using a relatively simple plug-in architecture. Also, a *meta-balancer* that examines application characteristics, to choose the appropriate strategy, and decide when to apply it, has been developed recently.

Automatic Checkpointing: Parallel application developers often need to write code for periodically checkpointing the state of their application to the disk. Simulation runs are often long, and need to be broken down into segments that will fit within system-allowed durations; also, hardware failures may cut short an ongoing simulation. Checkpoints allow one to handle such situations without losing much computation. Since CHARM++ already has the capability of migrating objects to other processors (with users providing information to optimize the amount of data saved, if needed), the RTS can leverage this capability to "migrate" copies of objects to the file system, along with the state of the runtime system itself. This reduces the burden on the programmer as they do not need to write additional checkpointing code. Further, when restarting, they can use a different number of processors than what was used for the original simulation, e.g., a job that was running on 10,000 cores can be restarted on 9,000 cores! This works automatically for baseline CHARM++ programs, and requires little extra programming for programs with user-defined groups and node-groups (Section 1.4).

Fault tolerance: One can also make a CHARM++ application continue to run in spite of individual nodes crashing in the middle of the execution! CHARM++ offers multiple alternative schemes for this purpose. The most mature, and probably most useful for applications today, is the double checkpointing scheme, which stores a checkpoint of each object locally and on a buddy node. An automatic failure-detection component checks the "heartbeat" of each node in a scalable and low-overhead manner. When a node fails, the system effects a recovery by automatically and quickly restoring the state of the last checkpoint. How quickly? We have measured restarts in hundreds of milliseconds for a molecular dynamics benchmark on over 64k cores [260]! Even on applications with very large checkpoints, it usually takes no more than a few seconds. One can use spare processors or make do with remaining processors on failure. For large runs, running with a few spares is inexpensive and simplifies the load balancing the system must do after restart.

A more advanced scheme based on *message-logging with parallel restart*

has also been developed [37, 36, 175]. With the double-checkpoint scheme (as with any other checkpoint-restart scheme), when a node fails, *all* the nodes must be rolled back to their checkpoints, wasting energy and wasting a lot of computation. With our message-logging schemes, when a node fails, only its objects restart form the checkpoint, while the others wait. The restarting objects can recover in parallel on other processors, thus speeding recovery. It does require storing of messages at the senders, which can add to memory overhead. Many strategies aimed at reducing this overhead have been developed [176]. This scheme is expected to be more important beyond Petascale, when node failures are likely to be frequent.

Power Management: Power, energy and temperature constraints are becoming increasingly important in parallel computing. CHARM++, with its introspective runtime system can help by monitoring core temperatures and power draw, and automatically changing frequencies and voltages. It can rely on its rate-aware load balancers (i.e. strategies that take into account the different speeds of different processors) to optimize either execution time or energy, while satisfying temperature and total-power constraints. As an illustration [219, 220], we were able to reduce cooling energy in a machine room by increasing the A/C thermostat setting; of course, that may lead to some chips overheating. However, the CHARM++ runtime system monitored chip temperatures periodically, and lowered the frequencies of chips that were getting too hot, while increasing them if they were cold enough. This creates a load imbalance which would slow the whole application down, as the rest of the processors wait for data from the processor whose frequency was lowered. However, the runtime is able to rebalance load by migrating objects after such frequency changes. These power-related features are available only in experimental versions of CHARM++ at the time of this writing, but are expected to be more broadly available in near future.

Communication optimizations: The CHARM++ runtime system is continually observing the communication patterns of the application, since it is delivering messages to chares. It can replace communication mechanisms based on the observed patterns. For example, algorithms for collective communication can be changed at runtime, between iterations of an application, based on the size of messages, number of nodes, and machine parameters [144].

1.4 Extensions to the Basic Model

In section 1.2 we described the basic CHARM++ programming model, consisting of chares, indexed collections (arrays) of chares, asynchronous method invocations, broadcasts and reductions. This baseline description is very useful

for developing an intuition about the programming model, and its underlying operational semantics (or execution model). A few important extensions to the base model, which enrich the programming model without changing its essential character, are noted below. These "extensions" are as mature and old as CHARM++ itself, and are in common use in applications today.

Supporting "blocking" calls: CHARM++ supports two additional kinds of methods, specifically tagged as such, that do allow "blocking" calls. They do not block the processor; only the affected entry method is paused, and control is returned to the scheduler. These are called `Structured Dagger` methods and `threaded` methods, as explained below.

A `Structured Dagger` (also abbreviated `sdag`) entry method allows users to define a DAG (directed acyclic graph) between computations within a chare, and asynchronous method invocations expected by the chare. This typically allows one to express the life cycle of a chare object more clearly than a baseline program would. An important statement in the structured-dagger notation is the so-called `when` statement, which specifies (1) that the object is ready to process a particular entry method invocation, and (2) what computation to execute when this method invocation arrives.

Just to give a flavor of how `sdag` code looks like, we present a snippet of code below. This comes from a molecular dynamics example, discussed briefly in the next chapter. But that is not important here; we are just illustrating the structure of `sdag` code. The "run" method of this chare includes a time step loop. In each time step `t`, the `run` method waits for two invocations of coordinates method, and when both are available executes some sequential object methods atomically. The sequential code calculates forces on each set of atoms C1 and C2 due to the other set of atoms, and sends the resultant forces back. Since this is not usual C++ code, sdag entries are specified in a separate file, which is translated into C++ code. One can thus think of `sdag` code as a *script* for describing data-dependent behavior of a chare. Typically, it describes the entire life cycle of a chare, as signified by the name, "run" method, in this particular case.

```
entry void run() {
  for (t=0; t<steps; t++) {
    when coordinates(vector <Atom> C1),
         coordinates(vector <Atom> C2)
      serial {
        calculateInteractions(C1, C2);
        sendForcesBack();
      }
  }
};
```

When a *threaded method* is invoked the runtime system creates a lightweight user-level thread and starts a method invocation inside this thread. A threaded entry method can block waiting for a *future* [99], or for a return value from a *synchronous* method invocation. Correspondingly, the system al-

lows users to define entry methods that return a value, as well as a simple future abstraction. One can create a `future`, set value to it, or access value from it (which is a blocking call). If a thread tries to access the value of a *future*, and the value is not set yet, the thread is blocked, and control is transferred to the CHARM++ scheduler. Later, when the value is set, the thread is added back to the scheduler's queue, so it can be resumed in its turn.

Array Sections: A subset of chares belonging to a chare array can be organized into a *section*, somewhat like an MPI sub-communicator. One can invoke broadcasts and reductions over sections as well. The system organizes efficient spanning trees over the subset of processors that house elements belonging to a section. It ensures that the broadcasts and reductions are carried out correctly even when element chares migrate to other processors, and reorganizes the spanning trees periodically, typically after a load balancing phase.

Processor-awareness: Another extension has to do with awareness of processors by the programmer. In the model described so far, there is no need for the programmer to know anything about the processors, including which processor is the current location of a particular object. However, there are some situations in which an "escape valve" into processor-aware programming is needed. This is especially true for libraries, or performance oriented optimizations. For example, many objects on the same processor may request the same remote data; it makes sense in this situation to use a shared processor-level software cache; Requests for remote data can go via this cache object, and if requested data was already obtained due to another chare's request, unnecessary remote communication is avoided. For such purposes, CHARM++ provides a construct called chare-group. Just like an array of chare objects, a chare-group is a collection of chares. However, (1) there is exactly one member mapped to each processor, and (2) unlike regular chares, chare group members are allowed to provide public methods that are invoked directly, without needing the packaging and scheduling of method invocations. Also, given the group ID, the system provides a function that returns a regular C++ pointer to the local (branch) chare of the group. With these two features, chares can communicate using low-overhead function calls with the member ("branch") of a group on their processor. Note that such group objects also allow additional data sharing mechanisms [228] beyond the read-only variables mentioned earlier.

So far, we intentionally left the notion of what we mean by a "processor" only loosely defined. In CHARM++, for processor-aware programming, there is a notion of PE (processing element). A PE corresponds to a single scheduler instance; a CHARM++ application may associate a PE with a hardware thread, a core, or a whole or a part of a multicore node, based on command-line options. If a PE includes multiple hardware resources (say cores), the parallelism within a PE is managed by the user orthogonally, by using pthreads, openMP, or CHARM++'s own task library (called `CkLoop`). Associating a PE with a hardware thread is a common practice in current CHARM++ applica-

tions, and it obviates the need to deal with an additional level of parallelism, so we will assume this in our description.

Of course, the group construct and other such low-level features should be used sparingly. As a design guideline, one should strive to avoid using processor-aware programming as much as possible, and push it into low-level libraries when needed. The example in the above paragraph, involving multiple requests for remote data, is a common enough feature that a new library, `CkCache`, has been developed as a common library for use by multiple applications. The system libraries for implementing asynchronous reductions are another example. Although one could implement a spanning tree over all the chares of a chare-array, it is much more efficient to do a processor (and node) based spanning tree, collecting inputs from all the local chares first.

Since objects may be migrated by the runtime system to other processors, Charm++ also supports a special callback method that gets called after the object has been re-incarnated on another processor; this can be used to update any processor-specific information, such as pointers to local branches of groups, stored by the objects.

1.5 Charm++ Ecosystem

Charm++ is a mature and stable parallel programming system. Thanks to the popularity of applications such as NAMD, it is used by tens of thousands of users worldwide (The biomolecular simulation code NAMD, described in Chapter 4, has 45,000 registered users, as of December 2012). Charm++ is available on most national supercomputer installations in the US. Charm++ runs on almost all the parallel computer types that are widely known, including Cray machines, IBM Blue Gene series machines, Linux Clusters, Windows clusters etc. It supports multiple network types including proprietary networks on supercomputers, as well as commodity networks including Ethernet and Infiniband. Charm++ is regression-tested via a nightly build system on dozens of combinations of compilers, operating systems, processor families and interconnection networks.

The maturity of Charm++ is also reflected in the ecosystem of program development and analysis tools available for it. *Projections* is a sophisticated performance visualization and analysis tool. *CharmDebug* is a more recent and highly sophisticated debugging tool. In addition, the *LiveViz* library can be used to collect application or performance data during application run and visualize it as the program is running. The CCS (Converse Client-Server) library that underlies LiveViz also allows one to develop interactive parallel applications, whereby queries or messages can be injected into a running computation, either to examine specific attributes of a running simulation, or to effect changes in the execution of the application.

There are several online resources for learning CHARM++ and working with it. The software, manuals, tutorials and presentations, are available at http://charm.cs.illinois.edu. An active mailing list (charm@cs.illinois.edu) is used for reporting bugs and discussing programming issues and upcoming features. There is an annual workshop on CHARM++ and its applications in Urbana Illinois; the presentations from the workshop (startin in the year 2002), most including the video of the presentations, are also available online.

1.6 Other Languages in the Charm++ Family

CHARM++ is just one instance of a broader programming paradigm based on message-driven execution, migratable work and data-units, and an introspective and adaptive runtime system. Although CHARM++ is the earliest member of this family of programming languages there are a few others that we have developed that deserve a mention here. All of these are built on top of CHARM++, as CHARM++ turns out to be an excellent backend for developing new abstractions within this broad paradigm.

XMAPP is the name we have chosen for the abstract programming model that underlies CHARM++ as well as all the other languages described below. XMAPP is characterized by a few defining attributes:

- Over-decomposition: the interacting entities, be they units of work or units of data (or a mix of the two, as in CHARM++), into which the computation is decomposed by the programmer in such models are independent of the number of processors, and typically their number is much larger than the number of processors.
- Processor-independence: the interaction/communication between entities is in terms of names of those entities and not in terms of processors.
- Migratability: these entities can be migrated across processors during program execution, either by the runtime system, or the application itself, or both.
- Asynchrony: collectives and other communication-related operations are designed so that their implementations do not block the processor.
- Adaptivity: the runtime system takes responsibility of balancing load by leveraging its ability to migrate objects.

Adaptive MPI (AMPI) is an implementation of the MPI standard on top of CHARM++. In MPI, the computation is expressed as a collection of processes that send and receive messages among themselves. With AMPI, each

MPI process is implemented as a user level thread. These threads are embedded inside CHARM++ objects, and are designed to be migratable across processors with their own stack. As with CHARM++, multiple "processes" (i.e. MPI ranks) are typically mapped to a single core. Standard MPI calls, such as those for receiving messages, provide natural points to allow context switching among threads within a core, thus avoiding complexities of preemptive context switching. AMPI programs have shown to have comparable performance (somewhat slower for fine-grained messaging, but comparable for most applications) as the corresponding MPI program, even when no AMPI-specific features are being used. Those features, such as over-decomposition (and adaptive overlap of communication with computation), asynchronous collectives, load balancing, and fault tolerance, provide the motivation for using AMPI instead of plain MPI implementations.

MSA (Multiphase Shared Arrays) [51, 179] is a mini-language on top of CHARM++ that supports the notion of disciplined shared memory programming. It is a partitioned global address space (PGAS) language. The program here consists of a set of migratable threads and a set of data arrays. The data arrays are partitioned into user-defined "pages", which again are migratable data units implemented as chares. The main point of departure for the language is the notion of *access modes*. Each array may be in one of the few possible modes, such as "read-only" or "accumulate". All the threads must collectively synchronize to switch the mode of an array. This model is shown to avoid all data races, and yet captures a very large fraction of use cases where one would find shared global data useful.

Charisma [108] is a language that allows elegant expression of applications or modules that exhibit a static data-flow pattern. The computation is divided into chares. If the chares exchange the same set of messages (with different lengths and contents, to be sure) in every iteration, one can express the life-cycle of entire collections of chares in a simple script-like notation, where entry-methods are seen to publish and subscribe to tagged data.

Charj [179] is a compiler supported language that provides the same abstractions as CHARM++ combined with MSA. With compiler supported static analysis, Charj provides a more convenient and elegant syntax, automatic generation of serialization code, and several other optimizations based on static analysis. Charj is an experimental or research language at the current time.

1.7 Historical Notes

The precursors to CHARM++ (the "Chare Kernel") developed by us were aimed at combinatorial search applications, and at supporting parallel func-

tional and logical programming. However, once we turned our attention to science and engineering applications, we decided to mold our abstractions based on the needs of full-fledged and diverse applications. The first two applications examined were fluid dynamics [95] and biomolecular simulations [116]. These two (along with many small examples, and parallel algorithms such as the Fast multipole algorithm, histogram-based sorting [120, 142]) adequately demonstrated to us that our approach was avoiding the trap of being too specialized. This was especially true because we considered full-fledged applications, in addition to kernels or isolated algorithms. We thought that only by immersing ourselves in the nitty-gritty of developing a full-fledged application, would we be able to weigh the importance and relevance of alternative abstractions, and capabilities.

This position and approach towards development of abstractions were explicitly written down in a position paper around 1994 [118]. The biomolecular simulation program NAMD funded by NIH (and NSF, in the early days, under the "Grand Challenge Application Groups" program), provided us a good opportunity to practice and test this approach. NAMD was developed in collaboration with Klaus Schulten, a biophysicist with a computational orientation, and Bob Skeel, a numerical analyst, both Professors at University of Illinois.

1.8 Conclusion

We believe that Charm++ and the underlying XMAPP abstract programming model constitute an approach that is ready to deal with the upcoming challenges in parallel computing, arising from increasingly complex hardware, and increasingly sophisticated applications. It appears to us that the basic concepts in XMAPP are going to have to be inexorably adopted by the community, whichever language they choose to use in future. So, why not Charm++? Charm++ itself is a production-quality system that has demonstrated its capabilities in improving programmer productivity and in attaining high scalability on a wide variety of the parallel applications in science and engineering, as demonstrated by this book. Some applications have demonstrated scaling beyond half a million processor cores by now.

To simplify and ease adoption, Charm++ supports interoperability with MPI: some modules can be written in regular MPI, while others can be based on Charm++ or AMPI (or any of the other mini-languages in the Charm++ family). We invite the readers to experiment with this approach by writing modules of their applications in it, or by using an existing Charm++ library in their MPI application, or testing it by developing an isolated algorithm using it, and then possibly moving on to developing entire applications using Charm++, and reap the productivity and performance benefits.

Bibliography

[1] A. Adcroft, C. Hill, and J. Marshall. Representation of topography by shaved cells in a height coordinate ocean model. *Monthly Weather Review*, 125(9):2293–2315, 1997.

[2] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. Hpctoolkit: tools for performance analysis of optimized parallel programs http://hpctoolkit.org. *Concurr. Comput. : Pract. Exper.*, 22:685–701, April 2010.

[3] M.P. Allen and D.J. Tildesley. *Computer Simulations of Liquids*. Claredon Press, Oxford, (1989).

[4] R. J. Anderson. Tree data structures for n-body simulation. *SIAM J. Comput.*, 28:1923–1940, 1999.

[5] R. E. Angulo, V. Springel, S. D. M. White, A. Jenkins, C. M. Baugh, and C. S. Frenk. Scaling relations for galaxy clusters in the Millennium-XXL simulation. *ArXiv e-prints*, March 2012.

[6] Gabriel Antoniu, Luc Bouge, and Raymond Namyst. An efficient and transparent thread migration scheme in the PM^2 runtime system. In *Proc. 3rd Workshop on Runtime Systems for Parallel Programming (RTSPP) San Juan, Puerto Rico. Lecture Notes in Computer Science 1586*, pages 496–510. Springer-Verlag, April 1999.

[7] Amnon Barak, Shai Guday, and Richard G. Wheeler. The mosix distributed operating system. In *LNCS 672*. Springer, 1993.

[8] Kevin Barker, Andrey Chernikov, Nikos Chrisochoides, and Keshav Pingali. A Load Balancing Framework for Adaptive and Asynchronous Applications. In *IEEE Transactions on Parallel and Distributed Systems*, volume 15, pages 183–192, 2003.

[9] Kevin J. Barker and Nikos P. Chrisochoides. An Evaluation of a Framework for the Dynamic Load Balancing of Highly Adaptive and Irregular Parallel Applications. In *Proceedings of SC 2003*, Phoenix, AZ, 2003.

[10] J. Barnes and P. Hut. A Hierarchical O(NlogN) Force-Calculation Algorithm. *Nature*, 324:446–449, December 1986.

[11] C. Barrett, R. Beckman, K. Berkbigler, K. Bisset, B. Bush, K. Campbell, S. Eubank, K. Henson, J. Hurford, D. Kubicek, M. Marathe, P. Romero, J. Smith, L. Smith, P. Speckman, P. Stretz, G. Thayer, E. Eeckhout, and M. Williams. TRANSIMS: Transportation Analysis Simulation System. Technical Report LA-UR-00-1725, LANL, 2001.

[12] C. L. Barrett, K. Bisset, S. Eubank, M. V. Marathe, V.S. Anil Kumar, and Henning Mortveit. *Modeling and Simulation of Biological Networks*, chapter Modeling and Simulation of Large Biological, Information and Socio-Technical Systems: An Interaction Based Approach, pages 101–147. AMS, 2007.

[13] C. L. Barrett, S. Eubank, and M. V. Marathe. An interaction based approach to computational epidemics. In *AAAI' 08: Proceedings of the Annual Conference of AAAI*, Chicago USA, 2008. AAAI Press.

[14] C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. Complexity of Reachability Problems for Finite Discrete Dynamical Systems. *J. Comput. Syst. Sci.*, 72(8):1317–1345, 2006.

[15] Christopher L. Barrett, Richard J. Beckman, Maleq Khan, V.S. Anil Kumar, Madhav V. Marathe, Paula E. Stretz, Tridib Dutta, and Bryan Lewis. Generation and analysis of large synthetic social contact networks. In M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, editors, *Proceedings of the 2009 Winter Simulation Conference*, Piscataway, New Jersey, December 2009. Institute of Electrical and Electronics Engineers, Inc.

[16] A. Basermann, J. Clinckemaillie, T. Coupez, J. Fingberg, H. Digonnet, R. Ducloux, J.-M. Gratien, U. Hartmann, G. Lonsdale, B. Maerten, D. Roose, and C. Walshaw. Dynamic load balancing of finite element applications with the DRAMA Library. In *Applied Math. Modeling*, volume 25, pages 83–98, 2000.

[17] Jerome Baudry, Emad Tajkhorshid, Ferenc Molnar, James Phillips, and Klaus Schulten. Molecular dynamics study of bacteriorhodopsin and the purple membrane. *Journal of Physical Chemistry B*, 105:905–918, 2001.

[18] A.D. Becke. Density-Functional exchange-energy approximation with correct assymptotic behavior. *Phys. Rev. A*, 38:3098, (1988).

[19] R. J. Beckman, K. A. Baggerly, and M. D. McKay. Creating synthetic baseline populations. *Transportation Research Part A: Policy and Practice*, 30(6):415–429, 1996.

[20] Milind Bhandarkar, L. V. Kale, Eric de Sturler, and Jay Hoeflinger. Object-Based Adaptive Load Balancing for MPI Programs. In *Proceed-*

ings of the International Conference on Computational Science, San Francisco, CA, LNCS 2074, pages 108–117, May 2001.

[21] Abhinav Bhatele, Eric Bohm, and Laxmikant V. Kale. Optimizing communication for charm++ applications by reducing network contention. *Concurrency and Computation: Practice and Experience*, 23(2):211–222, 2011.

[22] Abhinav Bhatelé, Laxmikant V. Kalé, and Sameer Kumar. Dynamic topology aware load balancing algorithms for molecular dynamics applications. In *23rd ACM International Conference on Supercomputing*, 2009.

[23] Scott Biersdorff, Chee Wai Lee, Allen D. Malony, and Laxmikant V. Kale. Integrated Performance Views in Charm ++: Projections Meets TAU. In *Proceedings of The 38th International Conference on Parallel Processing (ICPP)*, pages 140–147, Vienna, Austria, September 2009.

[24] Keith Bisset, Ashwin Aji, Madhav Marathe, and Wu-chun Feng. High-performance biocomputing for simulating the spread of contagion over large contact networks. *BMC Genomics*, 13(Suppl 2):S3, 2012.

[25] E. Bohm, A. Bhatele, L.V. Kale, M.E. Tuckerman, S. Kumar, J.A. Gunnels, and G.J. Martyna. Fine-grained parallelization of the Car-Parrinello ab initio molecular dynamics method on the Blue Gene/L supercomputer. *IBM J. Res. Dev.*, 52 1/2:159–176, (2008).

[26] Eric Bohm, Abhinav Bhatele, Laxmikant V. Kale, Mark E. Tuckerman, Sameer Kumar, John A. Gunnels, and Glenn J. Martyna. Fine Grained Parallelization of the Car-Parrinello ab initio MD Method on Blue Gene/L. *IBM Journal of Research and Development: Applications of Massively Parallel Systems*, 52(1/2):159–174, 2008.

[27] Kevin J. Bowers, Edmond Chow, Huafeng Xu, Ron O. Dror, Michael P. Eastwood, Brent A. Gregersen, John L. Klepeis, Istvan Kolossvary, Mark A. Moraes, Federico D. Sacerdoti, John K. Salmon, Yibing Shan, and David E. Shaw. Molecular dynamics—scalable algorithms for molecular dynamics simulations on commodity clusters. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 84, New York, NY, USA, 2006. ACM Press.

[28] Kevin J. Bowers, Edmond Chow, Huafeng Xu, Ron O. Dror, Michael P. Eastwood, Brent A. Gregersen, John L. Klepeis, Istvan Kolossvary, Mark A. Moraes, Federico D. Sacerdoti, John K. Salmon, Yibing Shan, and David E. Shaw. Scalable algorithms for molecular dynamics simulations on commodity clusters. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2006. ACM Press.

[29] BRAMS. `http://www.cptec.inpe.br/brams`, 2009.

[30] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proceedings of Supercomputing'00*, Dallas, Texas, 2000.

[31] Robert K. Brunner and Laxmikant V. Kalé. Adapting to load on workstation clusters. In *The Seventh Symposium on the Frontiers of Massively Parallel Computation*, pages 106–112. IEEE Computer Society Press, February 1999.

[32] Robert K. Brunner and Laxmikant V. Kalé. Handling application-induced load imbalance using parallel objects. In *Parallel and Distributed Computing for Symbolic and Irregular Applications*, pages 167–181. World Scientific Publishing, 2000.

[33] G. T. Camacho and M. Ortiz. Computational modeling of impact damage in brittle materials. *Int. J. Solids Struct.*, 33:2899–2938, 1996.

[34] R. Car and M. Parrinello. Unified approach for molecular dynamics and density functional theory. *Phys. Rev. Lett.*, 55:2471, (1985).

[35] C. Cavazzoni, G.L. Chiarotti, S. Scandolo, E. Tosatti, M. Bernasconi, and M. Parrinello. Superionic and Metallic States of Water and Ammonia at Giant Planet Conditions. *Science*, 283:44, (1999).

[36] Sayantan Chakravorty and L. V. Kale. A fault tolerant protocol for massively parallel machines. In *FTPDS Workshop for IPDPS 2004*. IEEE Press, 2004.

[37] Sayantan Chakravorty and L. V. Kale. A fault tolerance protocol with fast fault recovery. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*. IEEE Press, 2007.

[38] K. Channakeshava, K. Bisset, M. Marathe, A. Vullikanti, and S. Yardi. High performance scalable and expressive modeling environment to study mobile malware in large dynamic networks. In *Proceedings of 25th IEEE International Parallel & Distributed Processing Symposium*, 2011.

[39] Karthik Channakeshava, Deepti Chafekar, Keith Bisset, Anil Vullikanti, and Madhav Marathe. EpiNet: A simulation framework to study the spread of malware in wireless networks. In *SIMUTools09*. ICST Press, March 2009. Rome, Italy.

[40] K.L. Chung, Y.L. Huang, and Y.W. Liu. Efficient algorithms for coding Hilbert curve of arbitrary-sized image and application to window query. *Information Sciences*, 177(10):2130–2151, 2007.

[41] A.J. Cohen, Paula Mori-Sanchez, and Weitao Yang. Insights into current limitations of density functional theory. *Science*, 321:792, (2008).

[42] P. Colella, D.T. Graves, T.J. Ligocki, D.F. Martin, D. Modiano, D.B. Serafini, and B. Van Straalen. Chombo Software Package for AMR Applications Design Document, 2003. `http://seesar.lbl.gov/anag/chombo/ChomboDesign-1.4.pdf`.

[43] M. C.Payne, M.P. Teter, D.C. Allan, T.A. Arias, and J.D. Joannopoulos. Iterative minimization techniques for ab initio total-energy calculations: molecular dynamics and conjugate gradients. *Rev. Mod. Phys.*, 64:1045, (1992).

[44] T.A. Darden, D.M. York, and L.G. Pedersen. Particle mesh Ewald. An N·log(N) method for Ewald sums in large systems. *JCP*, 98:10089–10092, 1993.

[45] M. Davis, G. Efstathiou, C. S. Frenk, and S. D. M. White. The evolution of large-scale structure in a universe dominated by cold dark matter. *Astrophys. J.*, 292:371–394, May 1985.

[46] M. Davis, G. Efstathiou, C. S. Frenk, and S. D. M. White. The evolution of large-scale structure in a universe dominated by cold dark matter. *Astrophys. J.*, 292:371–394, May 1985.

[47] W. Dehnen. Towards optimal softening in three-dimensional N-body codes - I. Minimizing the force error. *MNRAS*, 324:273–291, June 2001.

[48] S.W. deLeeuw, J.W. Perram, and E.R. Smith. Simulation of Electrostatic Systems in Periodic Boundary Conditions. I. Lattice Sums and Dielectric Constants. *Proc. R. Soc. London A*, 373:27, 1980.

[49] Department of Computer Science,University of Illinois at Urbana-Champaign, Urbana, IL. *The CHARM (5.9) programming language manual*, 2006.

[50] Department of Computer Science,University of Illinois at Urbana-Champaign, Urbana, IL. *The CONVERSE programming language manual*, 2006.

[51] Jayant DeSouza and Laxmikant V. Kalé. MSA: Multiphase specifically shared arrays. In *Proceedings of the 17th International Workshop on Languages and Compilers for Parallel Computing*, West Lafayette, Indiana, USA, September 2004.

[52] K. Devine, B. Hendrickson, E. Boman, M. St. John, and C. Vaughan. Design of Dynamic Load-Balancing Tools for Parallel Applications. In *Proc. Intl. Conf. Supercomputing*, May 2000.

[53] Karen D. Devine, Erik G. Boman, Robert T. Heaphy, Bruce A. Hendrickson, James D. Teresco, Jamal Faik, Joseph E. Flaherty, and Luis G. Gervasio. New challenges in dynamic load balancing. *Appl. Numer. Math.*, 52(2–3):133–152, 2005.

[54] J. Diemand, M. Kuhlen, P. Madau, M. Zemp, B. Moore, D. Potter, and J. Stadel. Clumps and streams in the local dark matter distribution. *Nature*, 454:735–738, August 2008.

[55] H.-Q. Ding, N. Karasawa, and W. A. Goddard, III. The reduced cell multipole method for Coulomb interactions in periodic systems with million-atom unit cells. *Chemical Physics Letters*, 196:6–10, August 1992.

[56] P. Domingos and M. Richardson. Mining the Network Value of Customers. In *Proc. ACM KDD*, pages 57–61, 2001.

[57] Isaac Dooley. *Intelligent Runtime Tuning of Parallel Applications With Control Points.* PhD thesis, Dept. of Computer Science, University of Illinois, 2010. http://charm.cs.uiuc.edu/papers/DooleyPhDThesis10.shtml.

[58] D. J. Earl and M.W. Deem. Parallel tempering: Theory, applications, and new perspectives. *Phys. Chem. Chem. Phys.*, 7:3910–3916, (2005).

[59] D. Easley and J. Kleinberg. *Networks, Crowds and Markets: Reasoning About A Highly Connected World.* Cambridge University Press, New York, NY, 2010.

[60] G. Efstathiou, M. Davis, S. D. M. White, and C. S. Frenk. Numerical techniques for large cosmological N-body simulations. *Astrophys. J. Supp.*, 57:241–260, February 1985.

[61] S.N. Eliane, E. Araújo, W. Cirne, G. Wagner, N. Oliveira, E.P. Souza, C.O. Galvão, and E.S. Martins. The SegHidro Experience: Using the Grid to Empower a HydroMeteorological. In *Proceedings of the First International Converence on e-Science and Grid Computing (e-Science/05)*, pages 64–71, 2005.

[62] S. Eubank, H. Guclu, V. S. Anil Kumar, M. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429:180–184, 2004.

[63] A. E. Evrard. Beyond N-body - 3D cosmological gas dynamics. *MNRAS*, 235:911–934, December 1988.

[64] P. P. Ewald. Die Berechnung optischer und elektrostatischer Gitterpotentiale. *Annalen der Physik*, 369:253–287, 1921.

[65] A. L. Fazenda, J. Panetta, P. Navaux, L. F. Rodrigues, D. M. Katsurayama, and L. F Motta. Escalabilidade de aplicação operacional em ambiente massivamente paralelo. In *Anais do X Simpósio em Sistemas Computacionais (WSCAD-SCC)*, pages 27–34, 2009.

[66] R.P. Feynman. *Statistical Mechanics.* Benjamin, Reading, (1972).

[67] B. Fitch, R. Germain, M. Mendell, J. Pitera, M. Pitman, A. Rayshubskiy, Y. Sham, F. Suits, W. Swope, T. Ward, Y. Zhestkov, and R. Zhou. Blue Matter, an application framework for molecular simulation on Blue Gene. *Journal of Parallel and Distributed Computing*, 63:759–773, 2003.

[68] Blake G. Fitch, Aleksandr Rayshubskiy, Maria Eleftheriou, T. J. Christopher Ward, Mark Giampapa, Michael C. Pitman, and Robert S. Germain. Molecular dynamics—blue matter: approaching the limits of concurrency for classical molecular dynamics. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 87, New York, NY, USA, 2006. ACM Press.

[69] IT Foster and BR Toonen. Load-balancing algorithms for climate models. In *Proceedings of Scalable High-Performance Computing Conference*, pages 674–681, 1994.

[70] Peter L. Freddolino, Anton S. Arkhipov, Steven B. Larson, Alexander McPherson, and Klaus Schulten. Molecular dynamics simulations of the complete satellite tobacco mosaic virus. *Structure*, 14:437–449, 2006.

[71] SR Freitas, KM Longo, MAF Silva Dias, R. Chatfield, P. Silva Dias, P. Artaxo, MO Andreae, G. Grell, LF Rodrigues, A. Fazenda, et al. The Coupled Aerosol and Tracer Transport model to the Brazilian developments on the Regional Atmospheric Modeling System (CATT-BRAMS). *Atmospheric Chemistry and Physics*, 9(8):2843–2861, 2009.

[72] C. S. Frenk, S. D. M. White, P. Bode, J. R. Bond, G. L. Bryan, R. Cen, H. M. P. Couchman, A. E. Evrard, N. Gnedin, A. Jenkins, A. M. Khokhlov, A. Klypin, J. F. Navarro, M. L. Norman, J. P. Ostriker, J. M. Owen, F. R. Pearce, U.-L. Pen, M. Steinmetz, P. A. Thomas, J. V. Villumsen, J. W. Wadsley, M. S. Warren, G. Xu, and G. Yepes. The Santa Barbara Cluster Comparison Project: A Comparison of Cosmological Hydrodynamics Solutions. *Astrophys. J.*, 525:554–582, November 1999.

[73] D. Frenkel and B. Smit. *Understanding Molecular Simulation.* Academic Press, 1996.

[74] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[75] George Karypis and Vipin Kumar. Multilevel k-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 48:96–129 , 1998.

[76] T. C. Germann, K. Kadau, I. M. Longini, Jr., and C. A. Macken. Mitigation strategies for pandemic influenza in the United States. *Proc. of National Academy of Sciences*, 103(15):5935–5940, April11 2006.

[77] P. H. Geubelle and J. Baylor. Impact-induced delamination of composites: a 2d simulation. *Composites B*, 29(B):589–602, 1998.

[78] R. Gevaerd, S. R. Freitas, and K. M. Longo. Numerical simulation of biomass burning emission and trasportation during 1998 roraima fires. In *Proceedings of International Conference on Southern Hemisphere Meteorology and Oceanography (ICSHMO) 8*, 2006.

[79] S. Ghan, X. Bian, A. Hunt, and A. Coleman. The thermodynamic influence of subgrid orography in a global climate model. *Climate Dynamics*, 20(1):31–44, 2002.

[80] S. Ghan and T. Shippert. Load balancing and scalability of a subgrid orography scheme in a global climate model. *International Journal of High Performance Computing Applications*, 19(3):237, 2005.

[81] D.S. Ginley and D. Cahen. *Fundamentals of Materials for Energy and Environmental Sustainability.* Cambridge University Press, Cambridge, UK.

[82] Filippo Gioachin and Laxmikant V. Kalé. Dynamic High-Level Scripting in Parallel Applications. In *In Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Rome, Italy, May 2009.

[83] Filippo Gioachin, Amit Sharma, Sayantan Chakravorty, Celso Mendes, Laxmikant V. Kale, and Thomas R. Quinn. Scalable cosmology simulations on parallel machines. In *VECPAR 2006, LNCS 4395, pp. 476-489*, 2007.

[84] Filippo Gioachin, Gengbin Zheng, and Laxmikant V. Kalé. Debugging Large Scale Applications in a Virtualized Environment. In *Proceedings of the 23rd International Workshop on Languages and Compilers for Parallel Computing (LCPC2010)*, number 10-11, Houston, TX (USA), October 2010.

[85] Filippo Gioachin, Gengbin Zheng, and Laxmikant V. Kalé. Robust Record-Replay with Processor Extraction. In *PADTAD '10: Proceedings of the 8th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging*, pages 9–19. ACM, July 2010.

[86] E Goldstein, A Apolloni, B Lewis, J C Miller, M Macauley, S Eubank, M Lipsitch, and J Wallinga. Distribution of vaccine/antivirals and the 'least spread line'; in a stratified population. *J R Soc Interface*, 7(46):755–64, 2010.

[87] R. Gould. Collective action and network structure. *American Sociological Review*, 58:182–196, 1993.

[88] F. Governato, C. Brook, L. Mayer, A. Brooks, G. Rhee, J. Wadsley, P. Jonsson, B. Willman, G. Stinson, T. Quinn, and P. Madau. Bulgeless dwarf galaxies and dark matter cores from supernova-driven outflows. *Nature*, 463:203–206, January 2010.

[89] F. Governato, B. Willman, L. Mayer, A. Brooks, G. Stinson, O. Valenzuela, J. Wadsley, and T. Quinn. Forming disc galaxies in ΛCDM simulations. *MNRAS*, 374:1479–1494, February 2007.

[90] S. L . Graham, P. B. Kessler, and M. K. McKusick. GPROF: a call graph execution profiler. *SIGPLAN 1982 Symposium on Compiler Construction*, pages 120–126, June 1982.

[91] M. Granovetter. Threshold Models of Collective Behavior. *American J. Sociology*, 83(6):1420–1443, 1978.

[92] L. Greengard. *The rapid evaluation of potential fields in particle systems.* PhD thesis, MIT, Cambridge, MA, USA., 1988.

[93] G. Grell and D. Devenyi. A generalized approach to parameterizing convection combining ensemble and data assimilation techniques. *Geophysical Research Letters*, 29(14):38–1, 2002.

[94] G. Grimmett. *Percolation.* Springer, 1989.

[95] A. Gursoy, L.V. Kale, and S.P. Vanka. Unsteady fluid flow calculations using a machine independent parallel programming environment. In R. B. Pelz, A. Ecer, and J. Hauser, editors, *Parallel Computational Fluid Dynamics '92*, pages 175–185. North-Holland, 1993.

[96] A. Haldane and R. May. Systemic risk in banking ecosystems. *Nature*, 469:351–355, 2011.

[97] M. Halloran, N. Ferguson, I. Longini S. Eubank, D. Cummings, B. Lewis, S Xu, C. Fraser, A. Vullikanti, T. Germann, D. Wagener, R. Beckman, K. Kadau, C. Barrett, C. Macken, D. Burke, and P. Cooley. Modeling targeted layered containment of an influenza pandemic in the united states. *PNAS*, 105(12):4639–4644, 2008.

[98] M. Elizabeth Halloran, Neil M. Ferguson, Stephen Eubank, Ira M. Longini, Derek A. T. Cummings, Bryan Lewis, Shufu Xu, Christophe

Fraser, Anil Vullikanti, Timothy C. Germann, Diane Wagener, Richard Beckman, Kai Kadau, Chris Barrett, Catherine A. Macken, Donald S. Burke, and Philip Cooley. Modeling targeted layered containment of an influenza pandemic in the united states. *Proceedings of the National Academy of Sciences*, 105(12):4639–4644, March 2008.

[99] R. Halstead. Multilisp: A Language for Concurrent Symbolic Computation. *ACM Transactions on Programming Languages and Systems*, October 1985.

[100] Tsuyoshi Hamada, Tetsu Narumi, Rio Yokota, Kenji Yasuoka, Keigo Nitadori, and Makoto Taiji. 42 tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 62:1–62:12, New York, NY, USA, 2009. ACM.

[101] Richard Hamming. *Numerical Analysis for Scientists and Engineers*. 1973.

[102] K. Heitmann, P. M. Ricker, M. S. Warren, and S. Habib. Robustness of Cosmological Simulations. I. Large-Scale Structure. *Astrophys. J. Supp.*, 160:28–58, September 2005.

[103] L. Hernquist, F. R. Bouchet, and Y. Suto. Application of the Ewald method to cosmological N-body simulations. *Astrophys. J. Supp.*, 75:231–240, February 1991.

[104] L. Hernquist and N. Katz. TREESPH - A unification of SPH with the hierarchical tree method. *Astrophys. J. Supp.*, 70:419–446, June 1989.

[105] D. Hilbert. Über die stetige abbildung einer linie auf ein flächenstück. *Mathematische Annalen*, 38:459–460, 1891.

[106] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. New York: McGraw-Hill, 1981.

[107] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136:B864, 1964.

[108] Chao Huang and Laxmikant V. Kale. Charisma: Orchestrating migratable parallel objects. In *Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC)*, July 2007.

[109] Chao Huang, Orion Lawlor, and L. V. Kalé. Adaptive MPI. In *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003), LNCS 2958*, pages 306–322, College Station, Texas, October 2003.

[110] Chao Huang, Gengbin Zheng, Sameer Kumar, and Laxmikant V. Kalé. Performance Evaluation of Adaptive MPI. In *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2006*, March 2006.

[111] J. JáJá. *An introduction to parallel algorithms*. Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 1992.

[112] Pritish Jetley, Filippo Gioachin, Celso Mendes, Laxmikant V. Kale, and Thomas R. Quinn. Massively parallel cosmological simulations with ChaNGa. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2008*, 2008.

[113] Pritish Jetley, Lukasz Wesolowski, Filippo Gioachin, Laxmikant V. Kalé, and Thomas R. Quinn. Scaling hierarchical n-body simulations on gpu clusters. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, Washington, DC, USA, 2010. IEEE Computer Society.

[114] Xiangmin Jiao, Gengbin Zheng, Phillip A. Alexander, Michael T. Campbell, Orion S. Lawlor, John Norris, Andreas Haselbacher, and Michael T. Heath. A system integration framework for coupled multiphysics simulations. *Engineering with Computers*, 22(3):293–309, 2006.

[115] J. M. Jiménez, B. L. Lewis, and S. Eubank. Hospitals as complex social systems: agent-based simulation of hospital-acquired infections. In *Proceedings of 2nd International Conference on Complex Sciences: Theory and Applications*, 2012.

[116] John A. Board Jr., Laxmikant V. Kale, Klaus Schulten, Robert D. Skeel, , and Tamar Schlick. Modeling biomolecules: Large sclaes, longer durations. *IEEE Computational Science & Engineering*, 1:19–30, Winter 1994.

[117] Rashmi Jyothi, Orion Sky Lawlor, and L. V. Kale. Debugging support for Charm++. In *PADTAD Workshop for IPDPS 2004*, page 294. IEEE Press, 2004.

[118] L. V. Kale. Application oriented and computer science centered HPCC research. pages 98–105, 1994.

[119] L. V. Kale and Milind Bhandarkar. Structured Dagger: A Coordination Language for Message-Driven Programming. In *Proceedings of Second International Euro-Par Conference*, volume 1123-1124 of *Lecture Notes in Computer Science*, pages 646–653, September 1996.

[120] L. V. Kale and Sanjeev Krishnan. A comparison based parallel sorting algorithm. In *Proceedings of the 22nd International Conference on Parallel Processing*, pages 196–200, St. Charles, IL, August 1993.

[121] L. V. Kale and Sanjeev Krishnan. A comparison based parallel sorting algorithm. In *Proceedings of the 22nd International Conference on Parallel Processing*, pages 196–200, St. Charles, IL, August 1993.

[122] L. V. Kale and Sanjeev Krishnan. Charm++: Parallel Programming with Message-Driven Objects. In Gregory V. Wilson and Paul Lu, editors, *Parallel Programming using C++*, pages 175–213. MIT Press, 1996.

[123] L. V. Kale, B. H. Richards, and T. D. Allen. Efficient parallel graph coloring with prioritization. In *Lecture Notes in Computer Science*, volume 1068, pages 190–208. Springer-Verlag, August 1995.

[124] Laxmikant Kale, Anshu Arya, Abhinav Bhatele, Abhishek Gupta, Nikhil Jain, Pritish Jetley, Jonathan Lifflander, Phil Miller, Yanhua Sun, Ramprasad Venkataraman, Lukasz Wesolowski, and Gengbin Zheng. Charm++ for productivity and performance: A submission to the 2011 HPC class II challenge. Technical Report 11-49, Parallel Programming Laboratory, November 2011.

[125] Laxmikant Kale, Anshu Arya, Nikhil Jain, Akhil Langer, Jonathan Lifflander, Harshitha Menon, Xiang Ni, Yanhua Sun, Ehsan Totoni, Ramprasad Venkataraman, and Lukasz Wesolowski. Migratable objects + active messages + adaptive runtime = productivity + performance a submission to 2012 HPC class II challenge. Technical Report 12-47, Parallel Programming Laboratory, November 2012.

[126] Laxmikant Kalé, Robert Skeel, Milind Bhandarkar, Robert Brunner, Attila Gursoy, Neal Krawetz, James Phillips, Aritomo Shinozaki, Krishnan Varadarajan, and Klaus Schulten. NAMD2: Greater scalability for parallel molecular dynamics. *Journal of Computational Physics*, 151:283–312, 1999.

[127] Laxmikant V. Kalé. The virtualization model of parallel programming : Runtime optimizations and the state of art. In *LACSI 2002*, Albuquerque, October 2002.

[128] Laxmikant V. Kalé. Performance and productivity in parallel programming via processor virtualization. In *Proc. of the First Intl. Workshop on Productivity and Performance in High-End Computing (at HPCA 10)*, Madrid, Spain, February 2004.

[129] Laxmikant V. Kalé, Sameer Kumar, Gengbin Zheng, and Chee Wai Lee. Scaling molecular dynamics to 3000 processors with projections: A performance analysis case study. In *Terascale Performance Analysis Workshop, International Conference on Computational Science(ICCS)*, Melbourne, Australia, June 2003.

[130] Laxmikant V. Kale, Gengbin Zheng, Chee Wai Lee, and Sameer Kumar. Scaling applications to massively parallel machines using projections performance analysis tool. In *Future Generation Computer Systems Special Issue on: Large-Scale System Performance Modeling and Analysis*, volume 22, pages 347–358, February 2006.

[131] L.V. Kalé and S. Krishnan. CHARM++: A Portable Concurrent Object Oriented System Based on C++. In A. Paepcke, editor, *Proceedings of OOPSLA'93*, pages 91–108. ACM Press, September 1993.

[132] L.V. Kalé and Amitabh Sinha. Projections: A preliminary performance tool for charm. In *Parallel Systems Fair, International Parallel Processing Symposium*, pages 108–114, Newport Beach, CA, April 1993.

[133] S.A. Kalogirou. *Solar Energy Engineering: Processes and Systems*. Academic Press, Waltham, MA USA.

[134] George Karypis and Vipin Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system. University of Minnesota, 1995.

[135] George Karypis and Vipin Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, page 35, 1996.

[136] Amal Kasry, Marcelo A. Kuroda, Glenn J. Martyna, George S. Tulevski, and Ageeth A. Bol. Chemical doping of large-area stacked graphene films for use as transparent, conducting electrodes. *ACS Nano*, 4(7):3839–3844, 2010.

[137] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the Spread of Influence Through a Social Network. In *Proc. ACM KDD*, pages 137–146, 2003.

[138] D. Kempe, J. Kleinberg, and E. Tardos. Influential Nodes in a Diffusion Model for Social Networks. In *Proc. ICALP*, pages 1127–1138, 2005.

[139] C.H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L. Steele Jr., and M.E. Zosel. *The High Performance Fortran Handbook*. MIT Press, 1994.

[140] W. Kohn and L.J. Sham. Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140:A1133, 1965.

[141] C. Koziar, R. Reilein, and G. Runger. Load imbalance aspects in atmosphere simulations. *International Journal of Computational Science and Engineering*, 1(2):215–225, 2005.

[142] Sanjeev Krishnan and L. V. Kale. A parallel adaptive fast multipole algorithm for n-body problems. In *Proceedings of the International Conference on Parallel Processing*, pages III 46 – III 50, August 1995.

[143] Rick Kufrin. Perfsuite: An Accessible, Open Source Performance Analysis Environment for Linux. In *In Proceedings of the Linux Cluster Conference*, 2005.

[144] Sameer Kumar. *Optimizing Communication for Massively Parallel Processing*. PhD thesis, University of Illinois at Urbana-Champaign, May 2005.

[145] Sameer Kumar, Chao Huang, Gheorghe Almasi, and Laxmikant V. Kalé. Achieving strong scaling with NAMD on Blue Gene/L. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006*, April 2006.

[146] Sameer Kumar, Yanhua Sun, and L. V. Kale. Acceleration of an asynchronous message driven programming paradigm on ibm blue gene/q. In *Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Boston, USA, May 2013.

[147] V. Kumar. *Introduction to parallel computing*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2002.

[148] Akhil Langer, Jonathan Lifflander, Phil Miller, Kuo-Chuan Pan, , Laxmikant V. Kale, and Paul Ricker. Scalable Algorithms for Distributed-Memory Adaptive Mesh Refinement. In *Proceedings of the 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2012)*, New York, USA, October 2012.

[149] Ilya Lashuk, Aparna Chandramowlishwaran, Harper Langston, Tuan-Anh Nguyen, Rahul Sampath, Aashay Shringarpure, Richard Vuduc, Lexing Ying, Denis Zorin, and George Biros. A massively parallel adaptive fast-multipole method on heterogeneous architectures. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009. ACM.

[150] Orion Lawlor, Sayantan Chakravorty, Terry Wilmarth, Nilesh Choudhury, Isaac Dooley, Gengbin Zheng, and Laxmikant Kale. Parfum: A parallel framework for unstructured meshes for scalable dynamic physics applications. *Engineering with Computers*, 22(3-4):215–235, September 2006.

[151] Orion Lawlor, Hari Govind, Isaac Dooley, Michael Breitenfeld, and Laxmikant Kale. Performance degradation in the presence of subnormal floating-point values. In *Proceedings of the International Workshop*

on Operating System Interference in High Performance Applications, September 2005.

[152] Orion Sky Lawlor. *Impostors for Parallel Interactive Computer Graphics*. PhD thesis, University of Illinois at Urbana-Champaign, December 2004.

[153] Orion Sky Lawlor and L. V. Kalé. Supporting dynamic parallel object arrays. *Concurrency and Computation: Practice and Experience*, 15:371–393, 2003.

[154] D. Lea and W. Gloger. A memory allocator. `http://web.mit.edu/sage/export/singular-3-0-4-3.old/omalloc/Misc/dlmalloc/malloc.ps`, 2000.

[155] C. Lee, W. Yang, and R.G. Parr. Development of the Calle-Salvetti correlation energy into a functional of the electron density. *Phys. Rev. B*, 37:785, (1988).

[156] Chee Wai Lee. *Techniques in Scalable and Effective Parallel Performance Analysis*. PhD thesis, Department of Computer Science, University of Illinois, Urbana-Champaign, December 2009.

[157] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46(3):259–271, 1990.

[158] J. Leskovec, L. Adamic, and B. Huberman. The Dynamics of Viral Marketing. *ACM Trans. on the Web*, 1(1), 2007.

[159] J.R. Levine. *Linkers and Loaders*. Morgan-Kauffman, 2000.

[160] Bryan Lewis, Stephen Eubank, Allyson M Abrams, and Ken Kleinman. In silico surveillance: evaluating outbreak detection with simulation models. *BMC medical informatics and decision making*, 13(1):12, January 2013.

[161] G. F. Lewis, A. Babul, N. Katz, T. Quinn, L. Hernquist, and D. H. Weinberg. The Effects of Gasdynamics, Cooling, Star Formation, and Numerical Resolution in Simulations of Cluster Formation. *Astrophys. J.*, 536:623–644, June 2000.

[162] X. Li, W. Cai, J. An, S. Kim, J. Nah, D. Yang, R. Piner, A. Velamakanni, I. Jung, E. Tutuc, S.K. Banerjee, L. Colombo, and R.S. Ruoff. Large-Area Synthesis of High-Quality and Uniform Graphene Films on Copper Foils. *Science*, 324:1312, (2009).

[163] X. Liu and G. Schrack. Encoding and decoding the Hilbert order. *Software, practice & experience*, 26(12):1335–1346, 1996.

[164] Kwan-Liu Ma, Greg Schussman, Brett Wilson, Kwok Ko, Ji Qiang, and Robert Ryne. Advanced visualization technology for terascale particle accelerator simulations. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[165] Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 95–ff. ACM Press, 1995.

[166] Sandhya Mangala, Terry Wilmarth, Sayantan Chakravorty, Nilesh Choudhury, Laxmikant V. Kale, and Philippe H. Geubelle. Parallel adaptive simulations of dynamic fracture events. *Engineering with Computers*, 24:341–358, December 2007.

[167] Achla Marathe, Bryan Lewis, Christopher Barrett, Jiangzhuo Chen, Madhav Marathe, Stephen Eubank, and Yifei Ma. Comparing effectiveness of top-down and bottom-up strategies in containing influenza. *PloS one*, 6(9):e25149, 2011.

[168] Achla Marathe, Bryan Lewis, Jiangzhuo Chen, and Stephen Eubank. Sensitivity of household transmission to household contact structure and size. *PloS one*, 6(8):e22461, 2011.

[169] Dominik Marx, Mark E. Tuckerman, and M. Parrinello. The nature of the hydrated excess proton in water. *Nature*, 601:397, (1999).

[170] L. Mayer, T. Quinn, J. Wadsley, and J. Stadel. Formation of Giant Planets by Fragmentation of Protoplanetary Disks. *Science*, 298:1756–1759, November 2002.

[171] M. McPherson, L. Smith-Lovin, and J. Cook. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, 27:415–444, 2001.

[172] Vikas Mehta. LeanMD: A Charm++ framework for high performance molecular dynamics simulation on large parallel machines. Master's thesis, University of Illinois at Urbana-Champaign, 2004.

[173] Chao Mei, Yanhua Sun, Gengbin Zheng, Eric J. Bohm, Laxmikant V. Kalé, James C.Phillips, and Chris Harrison. Enabling and scaling biomolecular simulations of 100 million atoms on petascale machines with a multicore-optimized message-driven runtime. In *Proceedings of the 2011 ACM/IEEE conference on Supercomputing*, Seattle, WA, November 2011.

[174] Chao Mei, Gengbin Zheng, Filippo Gioachin, and Laxmikant V. Kalé. Optimizing a Parallel Runtime System for Multicore Clusters: A Case Study. In *TeraGrid'10*, number 10-13, Pittsburgh, PA, USA, August 2010.

[175] Esteban Meneses, Greg Bronevetsky, and Laxmikant V. Kale. Dynamic load balance for optimized message logging in fault tolerant HPC applications. In *IEEE International Conference on Cluster Computing (Cluster) 2011*, September 2011.

[176] Esteban Meneses, Celso L. Mendes, and Laxmikant V. Kale. Team-based message logging: Preliminary results. In *3rd Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids (CCGRID 2010).*, May 2010.

[177] J. Michalakes. MM90: a scalable parallel implementation of the Penn State/NCAR Mesoscale Model (MM5). *Parallel Computing*, 23(14):2173–2186, 1997.

[178] John Michalakes, Josh Hacker, Richard Loft, Michael O. McCracken, Allan Snavely, Nicholas J. Wright, Tom Spelce, Brent Gorda, and Robert Walkup. Wrf nature run. In *Proceedings of SuperComputing*, pages 1–6, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[179] Phil Miller, Aaron Becker, and Laxmikant Kal. Using shared arrays in message-driven parallel programs. *Parallel Computing*, 38(12):66 – 74, 2012.

[180] J. Minkel. The 2003 northeast blackout–five years later. *Scientific American*, 2008. 13 August 2008, `http://www.scientificamerican.com/article.cfm?id=2003-blackout-five-years-later.`

[181] M.Levy. Universal variational functionals of electron densities, first-order density matrices, and natural spin-orbitals and solution of the v-representability problem. *Proc. Natl. Acad. Sci. U.S.A.*, 76:6062, (1979).

[182] P. R. Monge and N. S. Contractor. *Theories of Communication Networks*. Oxford University Press, USA, 2003.

[183] B. Moore, F. Governato, T. Quinn, J. Stadel, and G. Lake. Resolving the Structure of Cold Dark Matter Halos. *Astrophys. J. Lett.*, 499:L5–+, May 1998.

[184] E. Moretti. Social learning and peer effects in consumption: Evidence from movie sales. *Review of Economic Studies*, 78:356–393, 2011.

[185] H. Mortveit and C. Reidys. *An Introduction to Sequential Dynamical Systems*. Springer, New York, NY, 2007.

[186] Martin Mundhenk, Judy Goldsmith, Christopher Lusena, and Eric Allender. Complexity of finite-horizon markov decision process problems. *JACM*, 47(4):681–720, July 2000.

[187] National Institutes of Health, 2009. `http://www.nigms.nih.gov/Initiatives/MIDAS`.

[188] J. F. Navarro, C. S. Frenk, and S. D. M. White. A Universal Density Profile from Hierarchical Clustering. *Astrophys. J.*, 490:493, December 1997.

[189] NDSSL. Synthetic data products for societal infrastructures and proto-populations: Data set 2.0. Technical Report NDSSL-TR-07-003, NDSSL, Virginia Polytechnic Institute and State University, Blacksburg, VA, 24061, 2007.

[190] M. Newman. The structure and function of complex networks. *SIAM Review*, 45, 2003.

[191] M.E. Newman. Spread of epidemic disease on networks. *Phys. Rev. E*, 2002.

[192] D.M. Newns, B.G. Elmegreen, X.-H. Liu, and G.J. Martyna. High Response Piezoelectric and Piezoresistive Materials for Fast, Low Voltage Switching: Simulation and Theory of Transduction Physics at the Nanometer-Scale. *Adv. Mat.*, 24:3672, 2012.

[193] D.M. Newns, B.G. Elmegreen, X.-H. Liu, and G.J. Martyna. High Response Piezoelectric and Piezoresistive Materials for Fast, Low Voltage Switching: Simulation and Theory of Transduction Physics at the Nanometer-Scale. *Adv. Mat.*, 24:3672, 2012.

[194] D.M. Newns, B.G. Elmegreen, X.-H. Liu, and G.J. Martyna. The piezoelectronic transistor:A nanoactuator-based post-CMOS digital switch with high speed and low power. *MRS Bulletin*, 37:1071, 2012.

[195] D.M. Newns, J.A. Misewich, A. Gupta C.C. Tsuei, B.A. Scott, and A. Schrott. Mott Transition Field Effect Transistor. *Appl. Phys. Lett.*, 73:780, (1998).

[196] R. Nistor, D.M. Newns, and G.J. Martyna. Understanding the doping mechanism in graphene-based electronics: The role of chemistry. *ACS Nano*, 5:3096, (2011).

[197] A. Odell1, A. Delin1, B. Johansson, N. Bock, M. Challacombe, and A. M. N. Niklasson. Higher-order symplectic integration in Born.Oppenheimer molecular dynamics. *J. Chem. Phys.*, 131:244106, (2009).

[198] Committee on Modeling Community Containment for Pandemic Influenza and Institute of Medicine. *Modeling Community Containment for Pandemic Influenza: A Letter Report.* The National Academies Press, Washington D.C., 2006.

[199] J. P. Ostriker and P. J. E. Peebles. A Numerical Study of the Stability of Flattened Galaxies: or, can Cold Galaxies Survive? *Astrophys. J.*, 186:467–480, December 1973.

[200] Douglas Z. Pan and Mark A. Linton. Supporting reverse execution for parallel programs. *SIGPLAN Not.*, 24(1):124–129, 1989.

[201] J. P. Perdew, K. Burke, and M. Ernzerhof. Generalized Gradient Approximation Made Simple. *Phys. Rev. B*, 77:386, (1996).

[202] P. Perzyna. Fundamental problems in viscoplasticity. *Advances in applied mechanics*, 9(C):243–377, 1966.

[203] James C. Phillips, Gengbin Zheng, Sameer Kumar, and Laxmikant V. Kalé. NAMD: Biomolecular simulation on thousands of processors. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–18, Baltimore, MD, September 2002.

[204] Planck Collaboration, N. Aghanim, M. Arnaud, M. Ashdown, F. Atrio-Barandela, J. Aumont, C. Baccigalupi, A. Balbi, A. J. Banday, R. B. Barreiro, J. G. Bartlett, E. Battaner, K. Benabed, J.-P. Bernard, M. Bersanelli, H. Böhringer, A. Bonaldi, J. R. Bond, J. Borrill, F. R. Bouchet, H. Bourdin, M. L. Brown, C. Burigana, R. C. Butler, P. Cabella, J.-F. Cardoso, P. Carvalho, A. Catalano, L. Cayón, A. Chamballu, R.-R. Chary, L.-Y. Chiang, G. Chon, P. R. Christensen, D. L. Clements, S. Colafrancesco, S. Colombi, A. Coulais, B. P. Crill, F. Cuttaia, A. Da Silva, H. Dahle, R. J. Davis, P. de Bernardis, G. de Gasperis, G. de Zotti, J. Delabrouille, J. Démoclès, F.-X. Désert, J. M. Diego, K. Dolag, H. Dole, S. Donzelli, O. Doré, M. Douspis, X. Dupac, T. A. Enßlin, H. K. Eriksen, F. Finelli, I. Flores-Cacho, O. Forni, P. Fosalba, M. Frailis, S. Fromenteau, S. Galeotta, K. Ganga, R. T. Génova-Santos, M. Giard, J. González-Nuevo, R. González-Riestra, K. M. Górski, A. Gregorio, A. Gruppuso, F. K. Hansen, D. Harrison, A. Hempel, C. Hernández-Monteagudo, D. Herranz, S. R. Hildebrandt, A. Hornstrup, K. M. Huffenberger, G. Hurier, T. Jagemann, J. Jasche, M. Juvela, E. Keihänen, R. Keskitalo, T. S. Kisner, R. Kneissl, J. Knoche, L. Knox, H. Kurki-Suonio, G. Lagache, A. Lähteenmäki, J.-M. Lamarre, A. Lasenby, C. R. Lawrence, S. Leach, R. Leonardi, A. Liddle, P. B. Lilje, M. López-Caniego, G. Luzzi, J. F. Macías-Pérez, D. Maino, N. Mandolesi, R. Mann, F. Marleau, D. J. Marshall, E. Martínez-González, S. Masi, M. Massardi, S. Matarrese, F. Matthai, P. Mazzotta, P. R. Meinhold, A. Melchiorri, J.-B. Melin, L. Mendes, A. Mennella, M.-A. Miville-Deschênes, A. Moneti, L. Montier, G. Morgante, D. Mortlock, D. Munshi, P. Naselsky, P. Natoli, H. U. Nørgaard-Nielsen, F. Noviello, S. Osborne, F. Pasian, G. Patanchon, O. Perdereau, F. Perrotta, F. Piacentini, E. Pierpaoli, S. Plaszczynski, P. Platania, E. Pointecouteau, G. Polenta, N. Ponthieu, L. Popa, T. Poutanen, G. W. Pratt, J.-L. Puget, J. P. Rachen, R. Rebolo, M. Reinecke, M. Remazeilles, C. Renault, S. Ricciardi, T. Riller, I. Ristorcelli, G. Rocha, C. Rosset, M. Rossetti, J. A. Rubiño-Martín, B. Rusholme, M. Sandri, G. Savini, B. M.

Schaefer, D. Scott, G. F. Smoot, J.-L. Starck, F. Stivoli, R. Sunyaev, D. Sutton, J.-F. Sygnet, J. A. Tauber, L. Terenzi, L. Toffolatti, M. Tomasi, M. Tristram, L. Valenziano, B. Van Tent, P. Vielva, F. Villa, N. Vittorio, B. D. Wandelt, J. Weller, S. D. M. White, D. Yvon, A. Zacchei, and A. Zonca. Planck intermediate results. I. Further validation of new Planck clusters with XMM-Newton. *Astronomy and Astrophysics*, 543:A102, July 2012.

[205] S. J. Plimpton and B. A. Hendrickson. A new parallel method for molecular-dynamics simulation of macromolecular systems. *J Comp Chem*, 17:326–337, 1996.

[206] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.*, 117(1):1–19, 1995.

[207] C. Power, J. F. Navarro, A. Jenkins, C. S. Frenk, S. D. M. White, V. Springel, J. Stadel, and T. Quinn. The inner structure of ΛCDM haloes - I. A numerical convergence study. *Monthly Notices of the Royal Astronomical Society*, 338:14–34, January 2003.

[208] D. Reed, J. Gardner, T. Quinn, J. Stadel, M. Fardal, G. Lake, and F. Governato. Evolution of the mass function of dark matter haloes. *MNRAS*, 346:565–572, December 2003.

[209] D.K. Remler and P.A. Madden. Molecular Dynamics without effective potentials via the Car-Parrinello approach. *Mol. Phys.*, 70:921, (1990).

[210] E. R. Rodrigues, P. O. A. Navaux, J Panetta, and C. L. Mendes. A new technique for data privatization in user-level threads and its use in parallel applications. In *ACM 25th Symposium On Applied Computing (SAC), Sierre, Switzerland*, 2010.

[211] Eduardo R. Rodrigues, Philippe O. A. Navaux, Jairo Panetta, Alvaro Fazenda, Celso L. Mendes, and Laxmikant V. Kalé. A comparative analysis of load balancing algorithms applied to a weather forecast model. In *Proceedings of 22nd IEEE International Symposium on Computer Architecture and High Performance Computing*, Petrópolis - Brazil, 2010.

[212] Eduardo R. Rodrigues, Philippe O. A. Navaux, Jairo Panetta, Celso L. Mendes, and Laxmikant V. Kalé. Optimizing an MPI weather forecasting model via processor virtualization. In *Proceedings of IEEE International Conference on High Performance Computing (HiPC 2010)*, Goa - India, 2010.

[213] D. Romero, B. Meeder, and J. Kleinberg. Differences in the Mechanics of Information Diffusion Across Topics: Idioms, Political Hashtags, and Complex Contagion on Twitter. In *Proceedings of the 20th International World Wide Web Conference (WWW 2011)*, 2011.

[214] Michiel Ronsse and Koen De Bosschere. RecPlay: a fully integrated practical record/replay system. *ACM Trans. Comput. Syst.*, 17(2):133–152, 1999.

[215] H.G. Rotithor. Taxonomy of dynamic task scheduling schemes in distributed computing systems. In *Proceedings of IEE: Computers and Digital Techniques*, volume 141, pages 1–10, 1994.

[216] J. J. Ruan, T. R. Quinn, and A. Babul. The Observable Thermal and Kinetic Sunyaev-Zel'dovich Effect in Merging Galaxy Clusters. *ArXiv e-prints*, April 2013.

[217] Ruth Rutter. Run-length encoding on graphics hardware. Master's thesis, University of Alaska at Fairbanks, 2011.

[218] J. K. Salmon and M. S. Warren. Skeletons from the treecode closet. *Journal of Computational Physics*, 111:136–155, March 1994.

[219] Osman Sarood and Laxmikant V. Kalé. A 'cool' load balancer for parallel applications. In *Proceedings of the 2011 ACM/IEEE conference on Supercomputing*, Seattle, WA, November 2011.

[220] Osman Sarood, Phil Miller, Ehsan Totoni, and L. V. Kale. 'Cool' Load Balancing for High Performance Computing Data Centers. In *IEEE Transactions on Computer - SI (Energy Efficient Computing)*, September 2012.

[221] Martin Schulz, Jim Galarowicz, Don Maghrak, William Hachfeld, David Montoya, and Scott Cranford. Open|speedshop: An open source infrastructure for parallel performance analysis. *Scientific Programming*, 16(2-3):105–121, 2008.

[222] Melih Sener, Johan Strumpfer, John A. Timney, Arvi Freiberg, C. Neil Hunter, and Klaus Schulten. Photosynthetic vesicle architecture and constraints on efficient energy harvesting. *Biophysical Journal*, 99:67–75, 2010.

[223] D. Shakhvorostov, R.A. Nistor, L. Krusin-Elbaum, G.J. Martyna, D.M. Newns, B.G. Elmegreen, X. Liu, Z.E. Hughesa, S. Paul, C. Cabral, S. Raoux, D.B. Shrekenhamerd, D.N. Basovd, Y. Songe, and M.H. Mueser. Evidence for electronic gap-driven metal-semiconductor transition in phase-change materials. *PNAS.*, 106:10907–10911, (2009).

[224] S. Shende and A. D. Malony. The TAU Parallel Performance System. *International Journal of High Performance Computing Applications*, 20(2):287–331, Summer 2006.

[225] S.A. Shevlin, A. Curioni, and W. Andreoni. Ab Initio Design of High-k Dielectrics: $La_xY_{1-x}AlO_3$. *Phys. Rev. Lett.*, 94:146401, (2005).

[226] S. Shingu, H. Takahara, H. Fuchigami, M. Yamada, Y. Tsuda, W. Ohfuchi, Y. Sasaki, K. Kobayashi, T. Hagiwara, S. Habata, et al. A 26.58 tflops global atmospheric simulation with the spectral transform method on the earth simulator. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–19. IEEE Computer Society Press, 2002.

[227] D. Siegel. Social networks and collective action. *Americal Journal of Political Science*, 53:122–138, 2009.

[228] A. Sinha and L.V. Kalé. Information Sharing Mechanisms in Parallel Programs. In H.J. Siegel, editor, *Proceedings of the 8th International Parallel Processing Symposium*, pages 461–468, Cancun, Mexico, April 1994.

[229] Marc Snir. A note on n-body computations with cutoffs. *Theory of Computing Systems*, 37:295–318, 2004.

[230] Edgar Solomonik and Laxmikant V. Kale. Highly Scalable Parallel Sorting. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2010.

[231] R. Souto, RB Avila, POA Navaux, MX Py, N. Maillard, T. Diverio, HC Velho, S. Stephany, AJ Preto, J. Panetta, et al. Processing mesoscale climatology in a grid environment. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid–CCGrid*, 2007.

[232] V. Springel. The cosmological simulation code GADGET-2. *MNRAS*, 364:1105–1134, December 2005.

[233] V. Springel, J. Wang, M. Vogelsberger, A. Ludlow, A. Jenkins, A. Helmi, J. F. Navarro, C. S. Frenk, and S. D. M. White. The Aquarius Project: the subhaloes of galactic haloes. *MNRAS*, 391:1685–1711, December 2008.

[234] V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, and F. Pearce. Simulations of the formation, evolution and clustering of galaxies and quasars. *Nature*, 435:629–636, June 2005.

[235] J. Stadel, D. Potter, B. Moore, J. Diemand, P. Madau, M. Zemp, M. Kuhlen, and V. Quilis. Quantifying the heart of darkness with GHALO - a multibillion particle simulation of a galactic halo. *MNRAS*, 398:L21–L25, September 2009.

[236] J. G. Stadel. *Cosmological N-body Simulations and their Analysis*. PhD thesis, Department of Astronomy, University of Washington, March 2001.

[237] Yanhua Sun, Gengbin Zheng, Chao Mei Eric J. Bohm, Terry Jones, Laxmikant V. Kalé, and James C.Phillips. Optimizing fine-grained communication in a biomolecular simulation application on cray xk6. In *Proceedings of the 2012 ACM/IEEE conference on Supercomputing*, Salt Lake City, Utah, November 2012.

[238] Yanhua Sun, Gengbin Zheng, L. V. Kale, Terry R. Jones, and Ryan Olson. A uGNI-based Asynchronous Message-driven Runtime System for Cray Supercomputers with Gemini Interconnect. In *Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Shanghai, China, May 2012.

[239] Emad Tajkhorshid, Aleksij Aksimentiev, Ilya Balabin, Mu Gao, Barry Isralewitz, James C. Phillips, Fangqiang Zhu, and Klaus Schulten. Large scale simulation of protein mechanics and function. In Frederic M. Richards, David S. Eisenberg, and John Kuriyan, editors, *Advances in Protein Chemistry*, volume 66, pages 195–247. Elsevier Academic Press, New York, 2003.

[240] Emad Tajkhorshid, Peter Nollert, Morten Ø. Jensen, Larry J. W. Miercke, Joseph O'Connell, Robert M. Stroud, and Klaus Schulten. Control of the selectivity of the aquaporin water channel family by global orientational tuning. *Science*, 296:525–530, 2002.

[241] Claudia Taylor, Achla Marathe, and Richard Beckman. Same influenza vaccination strategies but different outcomes across us cities? *International Journal of Infectious Diseases*, 14(9):e792 – e795, 2010.

[242] T.N. Theis and P.M. Solomon. In Quest of the "Next Switch": Prospects for Greatly Reduced Power Dissipation in a Successor to the Silicon Field-Effect Transistor. *Proceedings of the IEEE*, 98:2005, (2010).

[243] GJ Tripoli and WR Cotton. The Colorado State University three-dimensional cloud/mesoscale model. Technical Report 3, Atmos, 1982.

[244] M. Tuckerman, G. Martyna, M.L. Klein, and B.J. Berne. Efficient Molecular Dynamics and Hybrid Monte Carlo Algorithms for Path Integrals. *J. Chem. Phys.*, 99:2796, (1993).

[245] Ramkumar V. Vadali, Yan Shi, Sameer Kumar, L. V. Kale, Mark E. Tuckerman, and Glenn J. Martyna. Scalable fine-grained parallelization of plane-wave-based ab initio molecular dynamics for large supercomputers. *Journal of Comptational Chemistry*, 25(16):2006–2022, Oct. 2004.

[246] J. W. Wadsley, J. Stadel, and T. Quinn. Gasoline: a flexible, parallel implementation of TreeSPH. *New Astronomy*, 9:137–158, February 2004.

[247] R.L. Walko, L.E. Band, J. Baron, T.G.F. Kittel, R. Lammers, T.J. Lee, D. Ojima, R.A. Pielke Sr, C. Taylor, C. Tague, et al. Coupled atmosphere–biophysics–hydrology models for environmental modeling. *Journal of Applied Meteorology*, 39(6), 2000.

[248] Yuhe Wang and John Killough. A new approach to load balance for parallel compositional simulation based on reservoir model over-decomposition. In *2013 SPE Reservoir Simulation Symposium*, 2013.

[249] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, Supercomputing '93, pages 12–21, New York, NY, USA, 1993. ACM.

[250] M.S. Warren, J.K. Salmon, D.J. Becker, M.P. Goda, T. Sterling, and W. Winckelmans. Pentium pro inside: I. a treecode at 430 gigaflops on asci red, ii. price/performance of $50/mflop on loki and hyglac. In *Supercomputing, ACM/IEEE 1997 Conference*, page 61, nov. 1997.

[251] S. D. M. White, C. S. Frenk, and M. Davis. Clustering in a neutrino-dominated universe. *Astrophys. J. Lett.*, 274:L1–L5, November 1983.

[252] X.-P. Xu and A. Needleman. Numerical simulation of fast crack growth in brittle solids. *Journal of the Mechanics and Physics of Solids*, 42:1397–1434, 1994.

[253] M. Xue, K.K. Droegemeier, and D. Weber. Numerical Prediction of High-Impact Local Weather: A Driver for Petascale Computing. *Petascale Computing: Algorithms and Applications*, pages 103–124, 2007.

[254] Jae-Seung Yeom, Abhinav Bhatele, Keith Bisset, Eric Bohm, Abhishek Gupta, Laxmikant V. Kale, Madhav Marathe, Dimitrios S. Nikolopoulos, Martin Schulz, and Lukasz Wesolowski. Overcoming the scalability challenges of contagion simulations on Blue Waters. Technical Report 13-057, NDSSL, Virginia Bioinformatics Institute at Virginia Tech, 2013.

[255] Y. B. Zeldovich and R. A. Sunyaev. The Interaction of Matter and Radiation in a Hot-Model Universe. *Astrophysics & Space Science*, 4:301–316, July 1969.

[256] Gongpu Zhao, Juan R. Perilla, Ernest L. Yufenyuy, Xin Meng, Bo Chen, Jiying Ning, Jinwoo Ahn, Angela M. Gronenborn, Klaus Schulten, Christopher Aiken, and Peijun Zhang. Mature HIV-1 capsid structure by cryo-electron microscopy and all-atom molecular dynamics. *Nature*, 497:643–646, 2013. doi:10.1038/nature12162.

[257] Gengbin Zheng. *Achieving high performance on extremely large parallel machines: performance prediction and load balancing.* PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.

[258] Gengbin Zheng, Abhinav Bhatele, Esteban Meneses, and Laxmikant V. Kale. Periodic Hierarchical Load Balancing for Large Supercomputers. *International Journal of High Performance Computing Applications (IJHPCA)*, March 2011.

[259] Gengbin Zheng, Orion Sky Lawlor, and Laxmikant V. Kalé. Multiple flows of control in migratable parallel programs. In *2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, pages 435–444, Columbus, Ohio, August 2006. IEEE Computer Society.

[260] Gengbin Zheng, Xiang Ni, and L. V. Kale. A Scalable Double In-memory Checkpoint and Restart Scheme towards Exascale. In *Proceedings of the 2nd Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS)*, Boston, USA, June 2012.

[261] Gengbin Zheng, Lixia Shi, and Laxmikant V. Kalé. FTC-Charm++: An In-Memory Checkpoint-Based Fault Tolerant Runtime for Charm++ and MPI. In *2004 IEEE Cluster*, pages 93–103, San Diego, CA, September 2004.

[262] Gengbin Zheng, Terry Wilmarth, Praveen Jagadishprasad, and Laxmikant V. Kalé. Simulation-based performance prediction for large parallel machines. In *International Journal of Parallel Programming*, volume 33, pages 183–207, 2005.