

Data Driven Objects for Programming Multicore and Manycore chips

Laxmikant Kale

Dept. of Computer Science

University of Illinois at Urbana Champaign

<http://charm.cs.uiuc.edu>

kale@cs.uiuc.edu

The era of multicore and manycore chips has begun. Whether the market can sustain a continuously increasing numbers of cores over the years, just as it sustained chips with increasing frequencies over the past decades, depends on whether there are compelling CPU-hungry parallel applications for the desktop. In a significant measure, this depends on whether there is an effective programming model for such machines. Is there a model in sight that fits the bill?

I suggest that Charm++, along with its associated family of mini-languages, in development for over 12-15 years at Illinois, represents such a model. Charm++ has been very successful in science and engineering applications arena, yet it is probably even better suited for desktop parallelism, with a few extensions and elisions. This position paper describes Charm++, and outlines why I believe it is such a good match.

Charm++ programs are C++ programs, with the only extra requirement that some of the method declarations be provided in a separate file, specifying input and output parameters in a stylized manner for automatic marshaling. Charm++ defines a base class called a chare: only chare class objects can have their methods invoked via asynchronous method invocations from other processors.. A Charm++ computation consists of a number of chare objects, organized into multiple indexed collections. The Charm runtime system (RTS) assigns these objects to processors. Thus running a program on a different number of processors is accomplished without any extra programming effort. Further, the RTS can change the assignment of chares to processors dynamically, without user intervention. This allows the RTS to allocate resources to objects so as to handle dynamic variation in application behavior as well as environmental variations. E.g. if another program starts using a couple of cores, the Charm RTS can redistribute its objects accordingly.

Chares, as objects, provide encapsulation of data: each object typically accesses only its own data directly, accessing other object's data only via asynchronous method invocations. The locality of data accesses this engenders improves performance from desktops to petascale machines on which the program is run.

The intelligent runtime system is at the heart of Charm++ capabilities. The RTS mediates communication between objects, and schedules execution of objects, in addition to controlling their placement. Objects are anchored to individual processor cores. In fact, Charm++ uses one process (or system level thread) per core. Each process runs its

own user-level scheduler, which works with a pool of messages, each message corresponding to a single asynchronous method invocation. In the default case, the scheduler selects a message, delivers it to the chare involved, and allows it to run to completion. Therefore, each object executes its method atomically, without the need for a lock or a similar primitive. This message-driven paradigm allows Charm++ programs to be latency tolerant; Even more importantly in the manycore context, it allows the runtime system to accurately predict and prefetch the next few objects that are to execute. Since this is a much stronger predictive principle than the principle of locality that is at the heart of the cache hierarchy, the RTS can exploit scratchpad memories or prefetch capabilities effectively, as it does in case of the IBM cell processor.

Charm++ has been exceptionally successful for CSE applications. NAMD, a program for biomolecular simulations, has over 20,000 registered users, and has scaled effectively to machines with tens of thousands of processors. Yet, many of its users use it on small parallel machines as well as multicore desktops today. ChaNGa, for computational cosmology, and LeanCP (Quantum chemistry) or more recent collaborative applications and have been quite successful. 15 -- 20 percent of CPU cycles at two national centers were used on Charm++ applications (mostly NAMD) during a one year period.

C++ and message-driven execution were both alien concepts to the broader computational science and engineering community. (This is one reason why we developed adaptive AMPI on top of Charm++.) However, because of the event-driven programming used for GUI programming, both are natural for the desktop programming world.

Extensions

Charm++ is a very powerful system, which can be (and has been) used for a wide variety of applications. However, Charm++ also provides simpler specialized notations to facilitate easier expression of parallel programs in situations (algorithmic patterns, data access patterns) that are simpler in some ways. This is an important principle, because a large number of parallel data-sharing patterns fall within a few simple categories. We describe three such notations below.

Structured Dagger notation: the reactive message-driven style specification of an object's behavior can be simplified in some cases via a notation that cleanly specifies the life-cycle of a single object.

MSA (multiphase shared arrays) : supports a disciplined, deterministic, mechanism for sharing data structures such as arrays. Each array is accessed only in one mode (such as read-only, a write-exclusive, accumulate-only etc.) at a time, but the access modes change across phases at global synchronization points.

Charisma: this is an orchestration notation that specifies the behavior of a whole collection of objects. It provides a global view of control without sacrificing message-driven execution. It also cleanly separates parallel and sequential code, which is a style that is useful in many applications.

Tools:

*Charm++, as a mature system, comes with a sophisticated set of tools and libraries. *Projections* is a performance analysis and visualization tool. *LiveViz* is a tool that allows one to visualize a program's output even as the program is running (in contrast to the tradition in science and engineering applications, where the outputs are stored on files, and visualized later). Charm++ programs can run unchanged from single processors, multi-core chips, desktops consisting of multiple multi-core chips, all the way to petascale machines with tens of thousands of processors. (Of course, the application needs to have parallelism to keep all the processors busy). A tool called *BigSim* allows one to run a program in emulation mode on a small number of processors, collect coarse (method-invocation-level) traces, and run the traces on a simulator to predict performance on a different configuration.