Chapter 8.12

Abstractions and Middleware for Petascale Computing and Beyond

Ivo F. Sbalzarini

ETH Zurich, Switzerland

ABSTRACT

As high-performance computing moves to the petascale and beyond, a number of algorithmic and software challenges need to be addressed. This paper reviews the main performance-limiting factors in today’s high-performance computing software and outlines a possible new programming paradigm to address them. The proposed paradigm is based on abstract parallel data structures and operations that encapsulate much of the complexity of an application, but still make communication overhead explicit. The authors argue that all numerical simulations can be formulated in terms of the presented abstractions, which thus define an abstract semantic specification language for parallel numerical simulations. Simulations defined in this language can automatically be translated to source code containing the appropriate calls to a middleware that implements the underlying abstractions. Finally, the structure and functionality of such a middleware are outlined while demonstrating its feasibility on the example of the parallel particle-mesh library (PPM).

INTRODUCTION

Numerical simulations are well established as the third pillar of science, alongside theory and experiments. As numerical methods become more powerful, and more data and knowledge become available about complex real-world systems, the simulations become increasingly elaborate. The availability of parallel high-performance computing (HPC) systems has enabled simulations with unprecedented numbers of degrees of freedom. The corresponding simulation codes are mostly tightly coupled, which means that the different
processors of the HPC machine need to exchange
data (communicate) several times while solving
a problem, and not only at the beginning and the
end of the simulation. Minimizing the communica-
tion overhead is thus key to parallel efficiency. In
this article we propose a novel abstraction layer
that provides the proper level of granularity to
address some of the software challenges the field
is facing as we move beyond petascale systems.
We focus on tightly coupled simulations as they
occur in classical HPC applications in, e.g., ma-
terial science, fluid dynamics, astrophysics, or
computational chemistry, and in emerging user
fields such as biology, finance, or social science.

Despite the proliferation of HPC applications to
new areas of science, programming and using HPC
machines is becoming increasingly difficult. As
the performance of single processors has stopped
increasing, speedup can only be achieved through
parallelism. There is no more “free speedup”
for legacy codes. In addition, memory capacity
is growing faster than memory bandwidth (ag-
gravated by the fact that several processor cores
are sharing a memory bus), such that accessing
memory becomes increasingly expensive com-
pared to compute operations. Presently, it takes
about one to two orders of magnitude longer to
access the main memory than to perform a floating-
point multiplication. In GPUs and other emerging
heterogeneous cores this ratio is even higher. Ef-
cient codes should thus minimize memory access
counts rather than operation counts. This presents
challenges to both the traditional HPC user fields
as well as the emerging fields. In traditional fields,
well-tested and efficient codes have existed for
several decades. These codes are usually large
and of limited parallel scalability. They have
to be ported to heterogeneous multi-core HPC
systems or re-written altogether. For emerging
users in biology or social science, there is a high
entry hurdle into HPC since parallel program-
ing is notoriously difficult, requires experience,
and takes a long time. The predominantly used
message-passing paradigm resembles “commu-
nication assembly language” with every single
point-to-point communication explicitly coded
by the programmer. Together, these developments
cause several growing gaps in parallel HPC:

1. **The performance gap**: the actual sustained
   performance of scientific simulation codes
   is a decreasing fraction of the theoretical
   peak-performance of the hardware,

2. **The knowledge gap**: efficient use of HPC
   resources requires more and more special-
   ized knowledge and is restricted to a smaller
   and smaller community,

3. **The reliability gap**: as machines contain
   more and more processor cores, the mean-
time between failure drops below the typical
   runtime of a simulation, and

4. **The data gap**: storing, accessing, and analyz-
ing the peta-bytes of data generated by large
   simulations or experiments (such as those
   in astronomy or particle physics) becomes
   increasingly difficult.

Since the advent of multi-core CPUs, some
of these gaps even exist on the single-processor
level. Portable, generic scientific software libraries
such as GSL or “numerical recipes” are about one
order of magnitude slower than vendor-provided,
machine-specific libraries such as Intel’s IPP/
MKL. This is mainly due to the fact that the
latter are explicitly optimized and often contain
hand-tuned assembly language code for the
performance-critical sections, which, however,
limits their portability. Moreover, as memory is
becoming more expensive than processor cores,
the available memory per core decreases, caus-
ing bottlenecks due to memory contention when
independent heavy-weight processes (such as MPI
processes) are running on the different cores
(Sbalzarini, Walther, & Bergdorf et al., 2006).
Simulations are increasingly memory limited
and often use only as many cores per processor
as are needed to saturate the memory bandwidth,
disabling the rest of the cores or reducing their