newsletter of the parallel programming laboratory

inside this issue

PPL UIUC

Parallel Views

Energy and Fault Tolerance

Estimates show that the combined energy consumption for all data centers in the world is equivalent to 235 billion KWh and it accounts for more than 1.3% of the world's overall electricity consumption. Half of the energy consumed by a data center can be attributed to cooling. Although machine room temperature should be between 18°C-27°C, some data centers operate machine rooms at temperatures as low as 13°C due to the fear of increased node failures at higher temperatures. If processor operation at acceptable temperatures could be ensured, data centers can run at higher machine room temperatures. This is what motivates the temperature-restraining component of our work.

Modern day microprocessors support dynamic voltage and frequency scaling (DVFS), a means of changing the voltage and frequency at which a processor operates. The motivation for running processors at a lower frequency-voltage pair is to reduce power dissipation and, hence, control temperature. A component of the runtime system can periodically check processor temperatures and decrease frequency when the processor heats up beyond a user-defined threshold.

While most research focuses on minimizing core power, PPL's work started by focusing on cooling energy, which is often overlooked. The paper "A Cool Load Balancer for Parallel Application", published at Supercomputing 2011, proposed and evaluated a runtime technique that restrains core temperatures to reduce energy consumption. Most of the reduction in energy consumption was from cooling energy. The scheme was evaluated on a real testbed of 128 cores with a dedicated, controllable air-conditioning unit. The scheme was assessed on three metrics: ability to avoid hot spots by processor temperature control, execution overhead, and reduction in cooling energy. Results showed reductions of up to 63% in cooling energy consumption, with time overhead of only 2-23%. Moreover, this scheme was able to constrain core temperatures within the specified limits for all the experiments.

Recently, PPL has started focusing on reducing machine energy consumption as well. The paper "Efficient Cool Down of Parallel Applications," accepted at Power-Aware Systems and Architectures (PASA) 2012, proposes and evaluates a runtime technique for MPI that reduces both machine and cooling energy consumption. By dividing computation into different parts depending on sensitivity to frequency and running them at different frequencies, the technique achieved 17% reduction in machine energy consumption with as little as 0.9% increase in execution time while constraining core temperatures below 60°C.

In addition to reducing energy consumption, lower processor temperatures can also improve reliability of a machine. Past work shows that the failure rate of a compute node doubles with every 10°C increase in temperature. The recent paper, "A 'Cool' Way of Improving the Reliability of HPC Machines," to be presented at Supercomputing 13, proposes, implements, and evaluates a novel approach that combines temperature restraint, load balancing, and checkpoint/restart to increase Mean Time Between Failures (MTBF), while reducing total execution time for an application. At exascale, for a 350K-socket machine where regular checkpoint/restart fails to make progress (less than 1% efficiency), this

continued on page 4

From the Director's Desk

Prof. Laxmikant (Sanjay) Kale

Power, energy, temperature! These issues dominate discussions of extreme scale parallel computing these days. Many of us remember the plots shown by Intel's Shekhar Borkar in the late nineties, showing that chip temperatures, simply extrapolated, were set to increase to match those of rocket nozzles, and even the surface of the sun! That was stopped by limiting clock frequencies. Yet, the power consumption is in the 10+ MW range for modern petascale supercomputers. How can we get to 100-1000 fold more powerful machines with a much smaller increment in power consumption? Also, with process variability coming from smaller feature sizes and possibly near-threshold-voltage computing, thermal and speed variation issues will plague performance and reliability on large computers. The main article in this issue addresses how adaptive runtime systems, such as Charm++, can help optimize multiple metrics in the presence of multiple constraints. To be sure, hardware advances will be needed to close much of the gap, but runtime techniques are necessary to make the best of what the hardware will provide.

Two short articles describe progress on technical issues: The first deals with higher level parallel languages, including Charisma, and interoperability between them, in spite of very different interaction APIs used by each. The second deals with techniques to automate when and how to do load balancing, in an asynchronous manner. Then, we have an article on Episimdemics and the collaboration with Madhav Marathe and his team at Virginia Tech.

PPL will be there in force at SC13, with four papers in the main conference, and a Charm++ BOF on Tuesday evening. Hope to see you there!

PPL @ SC 2013

As in years past, the Parallel Programming Laboratory will be well represented at Supercomputing 2013 in Denver, Colorado. PPL members will present research spanning load balancing, performance tuning, resilience, and power management.

Several PPLers will be recognized at this year's conference. Abhishek Gupta will be presenting his research on high-performance computing in the cloud at this year's Dissertation Research Showcase. Despite the advantages of the pay-as-you-go model, elasticity, and virtualization in clouds, it is often difficult to use cost and resourse-utilization oriented clouds for HPC. Abhishek will showcase his research techniques for performing cost-effective and efficient HPC in the cloud. Ehsan Totoni will be presenting a poster on parallel algorithms for solving sparse triangular systems for the ACM Student Research Competition. PPL swept the George Michael HPC Fellowships, with both current PPLer Jonathan Lifflander and PPL alumnus Edgar Solomonik (now at UC Berkeley) winning this year.

PPL will host a Birds-of-a-Feather Session on the Charm++ ecosystem - its runtime, programming tools, and applications. Participants will hear about how these components are improving in performance, features, ease of use, portability, and interoperability. This is an ideal opportunity to learn more about how Charm++ is used in practice and to see how it is growing. The session will take place from 5:30PM to 7:00PM on Tuesday, November 19.

This year, PPL reached a new high, with a record four papers accepted at SC13.

A 'Cool' Way of Improving the Reliability of HPC Machines

Osman Sarood, Esteban Meneses, Laxmikant Kale

Tuesday, Nov. 19th at 1:30 PM

Reports predict that overall reliability at the exascale level could be so bad that failures might become the norm rather than the exception. In this work, we leverage both hardware and software aspects to improve the reliability of HPC machines. It has been postulated that fault rates in machines double for every 10°C increase in core temperatures. The paper exploits this idea to demonstrate the use of constraining core temperatures combined with load balancing, to improve the reliability of parallel machines as well as reduce the total execution time required by the application. Experimental results show that this method can improve the reliability of a machine by 2.5x and reduce the execution time by as much as 15%.

Distributed Dynamic Load Balancing for Iterative Applications

Harshitha Menon, Laxmikant Kale

Tuesday, Nov. 19th at 2:30 PM

This paper describes a fully distributed algorithm which uses partial information about the global state of the system for dynamic load balancing. The algorithm, referred to as "GrapevineLB," includes lightweight information propagation based on the gossip protocol for obtaining information about the system. Using this partial information, GrapevineLB probabilistically transfers work units to obtain high quality load balance with less overhead. The paper demonstrates the effectiveness of GrapevineLB for two applications on up to 131,072 cores of BlueGene/Q.

ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection

Xiang Ni, Esteban Meneses, Nikhil Jain, Laxmikant Kale

Wednesday, Nov. 20th at 1:30 PM

As machines increase in scale, many researchers have predicted a corresponding increase in failure rates. Soft errors do not inhibit execution, but may silently generate incorrect results, hence they must be handled to maintain correctness. In this paper we present a holistic methodology "ACR" which intelligently performs application replication along with checkpoint/restart for soft and hard error protection. ACR can automatically adapt the checkpoint period using online information about the current failure rate to decrease interventions to applications. By injecting failures that follow different distributions for five applications this paper shows that ACR incurs low overhead when scaled to 131,072 cores.

Predicting application performance using supervised learning on communication features

Nikhil Jain, Abhinav Bhatele, Michael Robson, Todd Gamblin, Laxmikant Kale

Thursday, Nov. 21th at 4:00 PM

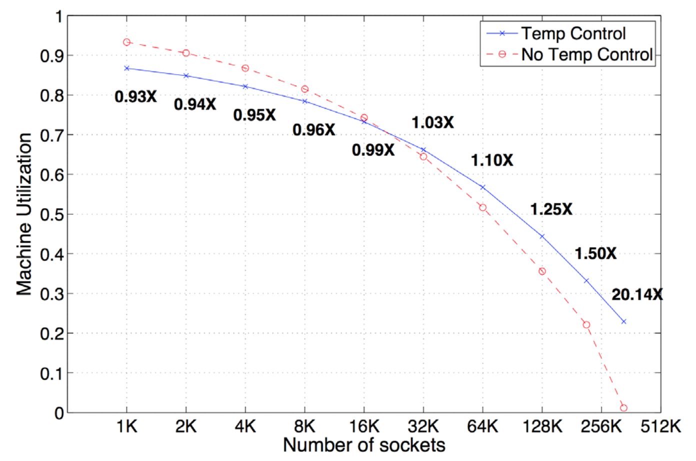
In order to eliminate performance bottlenecks caused due to communication in applications, it is vital to first study the factors that determine network behavior. This paper attempts to model the performance of an application by using communication data, such as the communication graph and network hardware counters. Supervised learning algorithms are used to correlate performance with combinations of metrics. For three different communication patterns and a production application, the paper demonstrates a very strong correlation between the new proposed metrics and the execution time of these codes.

continued from page 1

validated model predicts an efficiency of 20% by improving the MTBF by a factor of up to 2.29.

With current infrastructure, another big challenge for exascale machines is the strict power budget. Recent PPL research focuses on scheduling problems that involve efficiently running an overprovisioned HPC application under a given strict power budget. Given application characteristics, the challenge is to come up with an optimal schedule, the ideal machine room temperature, and the best power level for each processor. Preliminary analysis is encouraging, as it suggests the possibility of reducing both execution time and energy consumption while remaining under a strict power budget for an overprovisioned HPC data center.

The figure on the right compares the machine utilization of a stencil application using our scheme to baseline checkpoint restart without temperature control. The labels on the figure show the estimated 'times' improvement our scheme gets over the baseline checkpoint restart.



Grants

After going through a stretch of time with no new funding in sight, PPL is smiling again! There are two major new grants and several smaller sources of funding to report. It is interesting to note that almost all of these were awarded a few weeks ago, around September 2013!

PPL is proud to participate in a DOE-funded joint project called Argo (well, not the movie) to design the OS/runtime prototype for exascale machines! The project is led by Argonne National Laboratory, and other participants include PNNL, LLNL, the University of Tennessee, the University of Oregon, the University of Chicago, and Boston University. Many constructs and ideas in the Charm++ stack are expected to find their way in the Argo stack, or influence constructs in it.

PPL received a major grant from NSF's SI2-SSI program for developing HPC software for electronic structure methods, applicable to predictive simulations of nanoscale organic solar power systems and fast, low power, post-CMOS devices. This project builds upon our work on OpenAtom program, developed in collaboration with Dr. Glenn Martyna (IBM Research). It now includes Prof. Ismail-Beigi (Yale University), in addition to Dr. Martyna.

PPL is part of another large project awarded to a team at the University of Illinois, led by Bill Gropp: "Center for Exascale Simulation of Plasma-Coupled Combustion", funded by DOE's PSAAP2 program. We expect this project to begin by December 2013.

PPL's collaboration with Prof. Tom Quinn (University of Washington) on ChaNGa, the computational astronomy code, received a small boost, in the form of a collaborative grant to support the scaling work on "Evolution of the High Redshift Galaxy and AGN Populations," via NSF's CDS&E program. This will allow PPL to continue work on ChaNGa, which is a highly scalable and adaptive tree code. Last year, ChaNGa was selected to run on the Blue Waters system by NSF, as a PRAC (Petascale Computing Resource Allocation) awardee.

Some programs get evaluated and renewed each year. PPL has a collaborative project with Prof. Udatta Palekar on stochastic optimization, funded by the MITRE corporation. This is expected to be renewed for this new financial year. Also, an LLNL-funded project in collaboration with Dr. Abhinav Bhatele on topology-aware mapping is getting renewed for 2013-14. Gifts are especially neat, as they provide relatively unrestricted funds while allowing us to explore specific areas of interest to us. PPL is pleased to report a gift from Disney, encouraging our work on collision detection and load balancing for animation.

So, as you can see, PPL is back in the black! After losing some staff over the past two years, **PPL is looking for new staff (research programmer, postdoctoral associate) again! Encourage good candidates to apply at http://charm.cs.illinois.edu/.**

Visit the Charm++ website for more information

EpiSimdemics

co-written by Jae-Seung Yeom

Accurate simulations can be useful to understand and combat contagions such as epidemics of communicable disease among the population of a state, country, or ultimately, the entire planet. Contagion is used here broadly to mean transmitted phenomena such as diseases, opinions, trends, norms, packet diffusion, worm propagation in computer networks, database replication in sensor networks, spread of social movements, and influence among peers to purchase music videos or go to movies. However, an increase in input sizes and accuracy requirements, coupled with strict deadlines for simulations have resulted in simple computational approaches being insufficient. For example, the analysis necessary during the 2009 outbreak of the avian flu (H5N1) required a result within 24 hours to integrate the simulation results into the federal government's decision cycle. Techniques based on simplified probabilistic systems may not generate results accurate enough to be useful. In addition, they may not easily extend to large-scale realistic networks due to the absence of individuality and heterogeneity. Furthermore, many of the underlying problems related to dynamical properties become computationally intractable. Thus, HPC is a natural choice to study contagion diffusion via agent-based simulations.

Contagion diffusion can be viewed as interaction over a graph. Nodes in the graph represent interactors (computers, sensors, people, agents, or any other abstract entities). An edge between two nodes represents an interaction in an abstract form, labeled with its properties such as the time of interaction. With social contact networks, the graph has spatial structure with a small number of long-range edges and a significant amount of clustering among neighbors. One of the interesting problems is to understand the dynamics and evolution of the network with various interventions, such as public policy measures to mitigate the spread. Developing computational models to simulate such a phenomenon is complicated and scientifically challenging for at least following reasons:

1. The size and the scale of the network is extremely large, e.g. at global scale, there are over 7 billion interactors.
2. The interaction graph is unstructured where behavior of every interactor is distinct from the rest.
3. Interactors and the network of interactors co-evolve while providing feedback to evolve intervention.
4. One may need to simulate diffusion of multiple contagions simultaneously.
5. Simulation of contagion diffusion is typically done for many scenarios, whose combined outcome is useful for analysis.

In order to develop a simulation framework that is capable of handling the complex unstructured and evolving nature of contagion diffusion, one requires a rich programming model that is equally capable. EpiSimdemics is an agent-based simulation framework for contagion simulation that has been co-designed by domain specialists from Virginia Tech and PPL to meet the aforementioned requirements. It can be used to model a wide range of epidemic scenarios as well as the impact of counter measures. The social contact network data used in EpiSimdemics are synthesized to match the statistical properties at the street block level using the information available from various sources such as the US Census Bureau, American Community Survey, NAVTEQ, OSM, and so on. It is built on top of Charm++, which provides strong support for efficiently handling dynamic unstructured application environments. Based on the notion of over-decomposed, asynchronous message driven objects, Charm++ has a powerful runtime system that is apt for meeting the aforementioned requirements of contagion simulation.

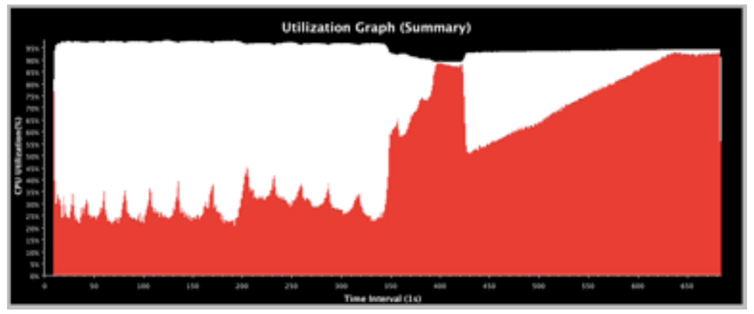
Use of an individual-based model for diffusion in EpiSimdemics makes the idea of "interaction" explicit, which allows for studying many interesting aspects of the network. However, this pushes the scalability requirement of the simulation to its limit with the growth in the volume of data and the semi-real-time demand. Initial attempts to meet these challenges focused on an implementation of EpiSimdemics in MPI. Scalability for EpiSimdemics in MPI was limited to only 376 processing elements (for a population of 100 million), far from sufficient for current needs. Since then, EpiSimdemics has been reimplemented in Charm++, taking advantage of its highly productive programming environment, abstract machine model that transparently optimize for SMP configurations and communication layers, various libraries for synchronization and fine-grained message handling, and interoperation with MPI-IO. As a result, the scalability of EpiSimdemics has been greatly improved and demonstrated for up to 352K processing elements on Blue Waters when simulating a population of 280 million. This is currently the largest social contact network data available while the global population data is under preparation.

In EpiSimdemics, the overhead of synchronization may grow quickly if not implemented carefully as there are two explicit global synchronizations per simulation iteration in addition to a reduction. These are necessary for detecting the completion of message deliveries, which are dynamic and data dependent. The Quiescence Detection (QD) mechanism in Charm++ offers a scalable solution to these problems. Further complications arise with the addition of support for branching of simulation for running multiple simulations simultaneously. Charm++'s Completion Detection provides an apt tool to handle these complications. It prevents the synchronization operation from interfering with the execution of the non-participants, and hence avoids a scalability bottleneck. The Topological Routing and Aggregation Module (TRAM) in Charm++, which offers transparent aggregation of small messages and can be easily configured to take advantage of underlying interconnection topology, is yet another example of a feature in Charm++ that is useful in EpiSimdemics.

The close association between the application scientists at Virginia Tech and systems researchers at Illinois has resulted in mutual benefits. EpiSimdemics performance has been significantly improved through use of Charm++ features and detailed performance analysis using PPL's Projections tool. For example, by profiling the application and then performing post-mortem visual analysis, load imbalance and synchronization overhead were identified as two dominant scaling bottlenecks. In turn, EpiSimdemics has fueled the research and development of some new parallel runtime features, and extensions of some existing ones. Using TRAM for many-to-many communication and interoperation between MPI and Charm++ are notable examples.

MetaBalancer

Load imbalance is a key factor that affects the performance of an application. As we move towards large systems, the chance that one of the processors has a load significantly greater than the average system load is high. However, performing load balance incurs overhead which includes the time spent in finding the new placement of work units and the time spent in moving the work units. Due to the cost of load balancing, it is important to determine if invoking the load balancer is profitable at all. In addition, the load characteristics of the application may change due the dynamic nature of the application. All this makes it difficult to identify when and how often to balance load. Currently, the user is responsible for identifying a load balancing period which is done using educated guesses and by extensively studying the application. A common practice is to use a fixed period for invoking the load balancer. This approach may not be efficient as it prevents the load balancing from adapting to the dynamic nature of the application.



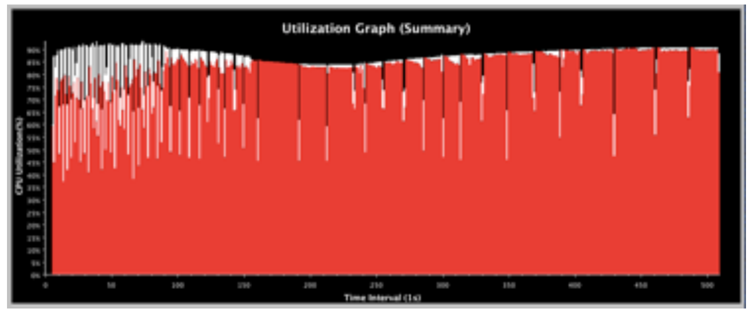


Figure 1 and 2: Comparing utilization without and with the Metabalancer.

MetaBalancer is an adaptive control mechanism to automate some of the decisions related to load balancing. It is a part of the run-time system which monitors the application characteristics and based on some guiding principles invokes the load balancer. MetaBalancer collects information about the application characteristics aggressively to determine if load balancing is required. At every fixed small interval, it aggregates some statistics which includes the maximum load, average load and the minimum utilization of all the processors in the system. In order to aggregate the information with less overhead, it collects minimal set of statistics and uses asynchronous reduction mechanism. Once the load statistics is collected, MetaBalancer determines whether there is load imbalance using the ratio of maximum load to the average load. If there is load imbalance in the system, it will lead to performance loss. However, the presence of load imbalance does not mandate load balancing as it may not be profitable due to the overhead incurred in balancing load. An ideal load balancing period is calculated using a linear model to predict the load characteristics of the system. The load balancing period is chosen so that the benefit of performing load balancing overshoots the cost incurred. The load model is updated and the LB period is refined as the application makes progress. To minimize the overhead of this process, the load balancing decisions are taken asynchronously. A distributed consensus scheme is used to identify the ideal load balancing period and ensure that all the objects enter the load balancing phase.

To demonstrate the adaptive nature of MetaBalancer we use it to perform load balance decisions for `Fractography3D'. Fractography3D is an application used to study the fracture surface of materials. When an external force is applied to the material, the initial elastic response of the material may change to plastic as stress increases which results in high load concentration in that region. Figure 1 shows the average processor utilization as the execution progress. It can be seen that Fractography3D has a large variation in processor utilization. Figure 3 shows the total application run time for a range of fixed LB periods. If the load balancer is invoked frequently, the overhead of load balancing overshoots the gains and results in bad performance. On the other hand, if load balancing is performed infrequently, the load imbalance in the system reduces the application performance. When using MetaBalancer, it adapts to the application characteristics as shown in Figure 2. An interesting thing to note is the frequent invocation of load balancing by Meta-Balancer in the first quarter of the execution as seen by the vertical notches in the plot. Thereafter, when the load variation decreases, the frequency of load balancing goes down.

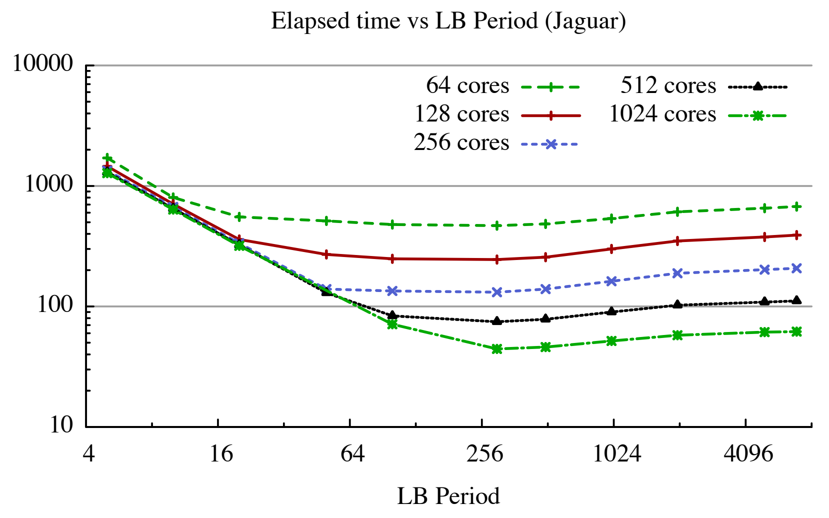
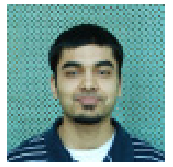


Figure 3: Total application run time for a range of fixed LB periods.

MetaBalancer automates the process of when and how often to balance load depending on the application characteristics and the load balancing overhead. We have shown that MetaBalancer is able to identify the ideal load balancing period and extract the best performance without any help from the user.

Transitions



Pritish Jetley is interested in the design and implementation of higher-level and specialized languages for the productive, performance-oriented programming of modern parallel computers. He enjoys studying aspects of application performance at scale, and the emergence of intelligent functions from the dynamics of simple, neural units. Since leaving PPL, he has been a part of the Qualcomm Zeroth project, which aims to create biologically inspired, intelligent machines.

Esteban Meneses is a Research Assistant Professor in the Center for Simulation and Modeling (SaM) at the University of Pittsburgh. He is currently extending a computational fluid dynamics application by adding a dynamic load balancing mechanism and several kernels to use accelerators. His responsibilities at SaM also include providing support in HPC to the scientific computing community at the university. He is still interested in fault tolerance strategies for extreme-scale systems. Currently, he is participating in various projects related to resilience for HPC.

Ramprasad Venkataraman recently transitioned to an engineering role at Google. Ram joined PPL in 2008 as a research staff engineer. He is interested in scalable and productive programming models, parallel application design, collective communication optimizations and runtime assisted adaptivity in the context of parallel algorithms for novel domains. Ram worked on the Charm++ runtime system and was part of the core group that maintained and evolved the runtime capabilities. He also worked on applications and collaborations that used Charm++, notably: OpenAtom (quantum chemistry), JetAlloc (stochastic optimization) and CharmLU (LU benchmark). He contributed to, and organized the team that submitted and won the HPC Challenge award at SC in 2011. Ram plans to continue his involvement with large-scale HPC in the coming years.

Domain-Specific Languages

It is believed that currently dominant (e.g. MPI, OpenMP) programming paradigms suffer from productivity (e.g. message passing, as embodied by MPI) or scalability (thread-based fork-join parallelism, as embodied by OpenMP and Cilk) pitfalls. PPL (and others, e.g. the designers of PGAS/APGAS languages such as X10, UPC, CAF, etc.) believes that parallel programming can be made easier through the provision of higher-level, abstract languages for programming.

In particular, PPL is interested in the productive and performance-oriented construction of scalable HPC applications of interest to the scientific and engineering communities. Previous research has shown that most HPC applications can be categorized into one of a handful of communication patterns.

In recent work, PPL has been creating a programming system that aims to exploit this regularity of expression in the realm of HPC. This system is intended to provide a means for the modular, productive and performance-oriented construction of parallel HPC applications. Specifically, PPL has identified three broad application classes that are of interest to the HPC community:

- Programs with data-independent communication patterns (*Charisma*)
- Generatively recursive programs on distributed memory machines (*Divcon*)
- Programs based on distributed trees (*Distree*)

Moreover, a higher-level, specialized mini-language was designed for each application class. The price of language specialization, however, is that of completeness of expression. That is, not all types of parallel computation can be expressed in any one of our languages. Indeed, we expect the programmer to identify the language best suited for the expression of each application (or module thereof). Moreover, if none of the specialized paradigms suffice, the programmer must express the computation in a general-purpose programming paradigm, namely Charm++.

This work enables interoperability between differently expressed parallel modules, thereby allowing the programmer to phrase her parallel application as the composition of a number of relatively independent, productively-expressed modules. Each module of the application is expressed in the paradigm that most closely matches its semantics. Moreover, all of the specialized paradigms as well as the base language are based on the *object-based, message-driven* execution model. This allows module execution to be automatically interleaved, based on availability of data for each module.

To demonstrate this multi-paradigm approach, PPL has developed a Barnes-Hut simulator that comprises modules written in Charisma, Divcon, Distree, and Charm++. Specifically, data-independent control and data flows are expressed in Charisma; ORB domain decomposition is expressed in Divcon; and the highly irregular, data-dependent data flows of the tree traversal are expressed in Distree and Charm++. Importantly, the performance of this multi-paradigm program is comparable to a highly optimized Charm++ code with similar functionality.

Charm++ Apps Book

At Supercomputing 2013, the first book on parallel applications developed using Charm++ will be released. *Parallel Science and Engineering Applications: The Charm++ Approach* surveys a diverse collection of scalable science and engineering applications, most of which are used regularly on supercomputers by scientists. The book is co-edited by Laxmikant Kale and Abhinav Bhatele and is composed of contributed chapters on different Charm++ applications.

The first few chapters provide a brief introduction to Charm++ and associated tools, and design considerations for writing a Charm++ program. Then, the book presents several parallel codes written in the Charm++ model and their underlying scientific and numerical formulations, explaining their parallelization strategies and performance. These chapters demonstrate the versatility of Charm++ and its utility for a wide variety of applications, including molecular dynamics (NAMD), cosmology (ChaNGa), fracture simulations (Fractography3D), quantum chemistry (OpenAtom), agent-based simulations (EpiSimdemics), and weather modeling (BRAMS).

The book is intended for a wide audience of people, both in academia and industry. Application developers and users will find this book interesting as an introduction to Charm++ and to developing parallel applications in an asynchronous, message-driven model. It will also be a useful reference for undergraduate and graduate courses on parallel computing in computer science and other engineering disciplines.

In the words of Horst Simon (LBNL), "Laxmikant (Sanjay) Kale and Abhinav Bhatele have developed an eminently readable and comprehensive book that provides the very first in depth introduction in book form to the research that Sanjay Kale has pioneered in the last 20 years with his research group at the University of Illinois [...] It succeeds perfectly and combines for the first time both Charm++ and significant application development in a single volume. It will provide a solid foundation for anyone who is considering using the most recent tools for developing applications for future Exascale platforms."

Department of Computer Science
University of Illinois
College of Engineering
201 North Goodwin Avenue
Urbana, IL 61801-2302

Non-profit Organization
US POSTAGE
PAID
Permit No. 75
CHAMPAIGN IL 61820