Charm++
on the road to Exascale

Isaac Dooley

PEEPS @ VecPar

Berkeley, June 2010

ILLINOIS

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

PARALLEL PROGRAMMING LAB

DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS

PPL UIUC

Overview

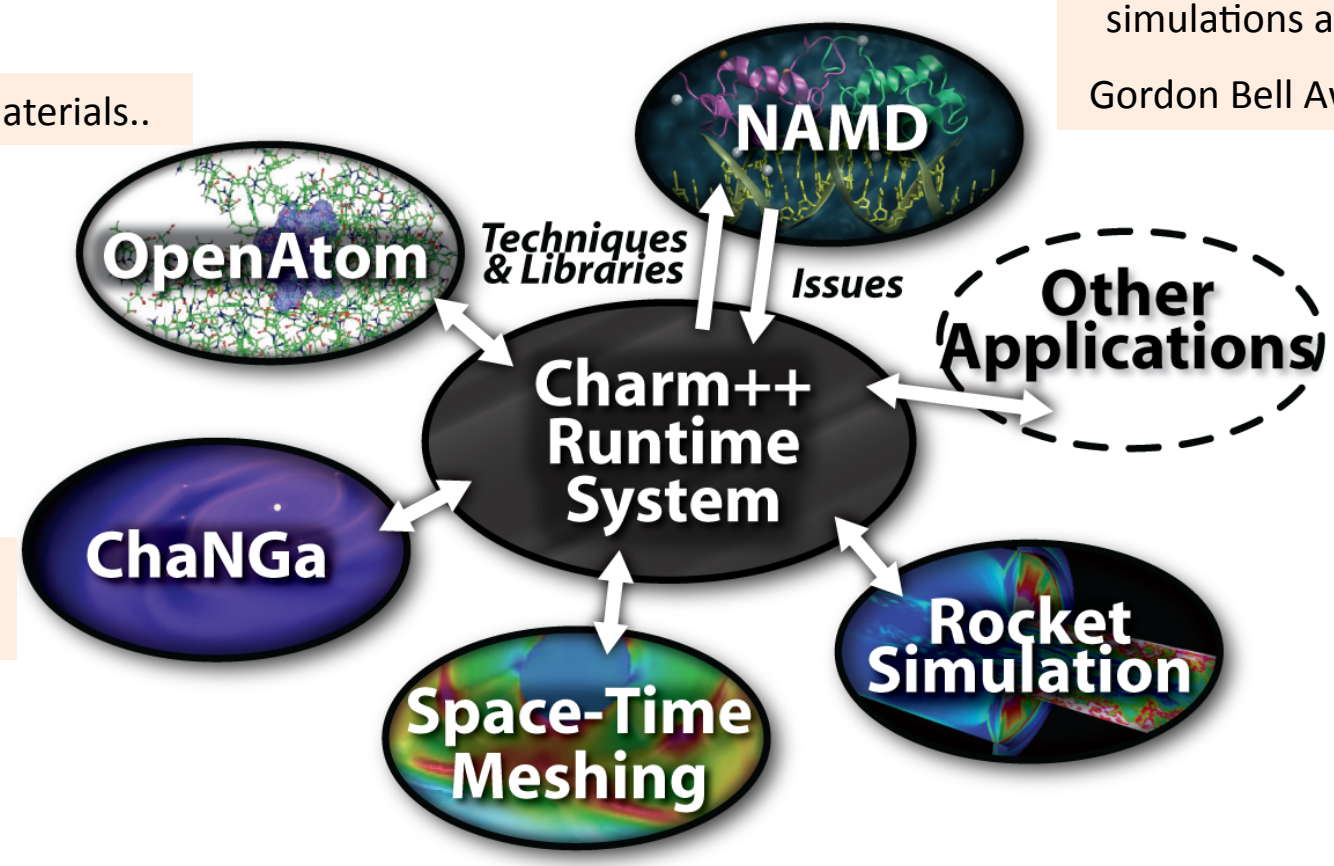
- Context
- History
- Philosophy
- Over-decomposition into Migratable Objects
- Some Current Research Efforts
  - Scalable Performance Analysis
  - Automatic Performance Tuning
  - New types of Runtime System Adaptation
  - Load Balancing
  - Topology Mapping
  - Accelerators
  - Incomplete but Useful Languages
- Promising Features in Charm++ for the Future

Context (so far)

Well-known molecular simulations application

Gordon Bell Award, 2002

Nano-Materials..



Computational Astronomy

A Glance at History

- 1987: Chare Kernel arose from parallel Prolog work
  - Dynamic load balancing for state-space search, Prolog, ..
- 1992: Charm++
- 1996: Charm++ in almost current form
  - Chare arrays, Measurement Based Dynamic Load balancing
- 1997: AMPI

But then things get complex: Multicore, GPU, Heterogeneous, Manycore, Large interconnects, RDMA on Infiniband

- 2008-2011: Blue Waters:
  - Charm++ and NAMD will scale to 1PFlop/s sustained application performance

PPL Mission and Approach

- To enhance *Performance and Productivity* in programming complex parallel applications
  - Performance: scalable to >100s of thousands of processors
  - Productivity: of human programmers
  - Complex: irregular structure, dynamic variations
- Approach: *Application Oriented yet CS centered research*
  - Develop enabling technology, for a wide collection of apps.
  - Develop, use and test it in the context of real applications

Some Guiding Principles

- No magic
  - Parallelizing compilers have achieved close to technical perfection, but are not enough
  - Sequential programs obscure too much information
- Seek an optimal division of labor between the system and the programmer
- Design abstractions based solidly on use-cases
  - Application-oriented yet computer-science centered approach

L. V. Kale, "Application Oriented and Computer Science Centered HPCC Research", Developing a Computer Science Agenda for High-Performance Computing, New York, NY, USA, 1994, ACM Press, pp. 98-105.

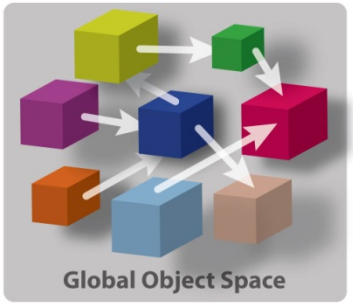
Migratable Objects (aka Processor Virtualization)

Programmer: [Over] decomposition into virtual processors / objects

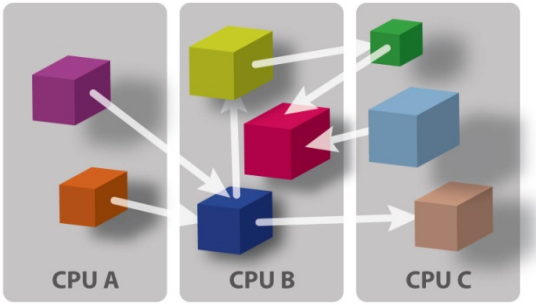
Runtime: Assigns VPs to processors

Enables *adaptive runtime strategies*

Implementations: Charm++, AMPI



User View

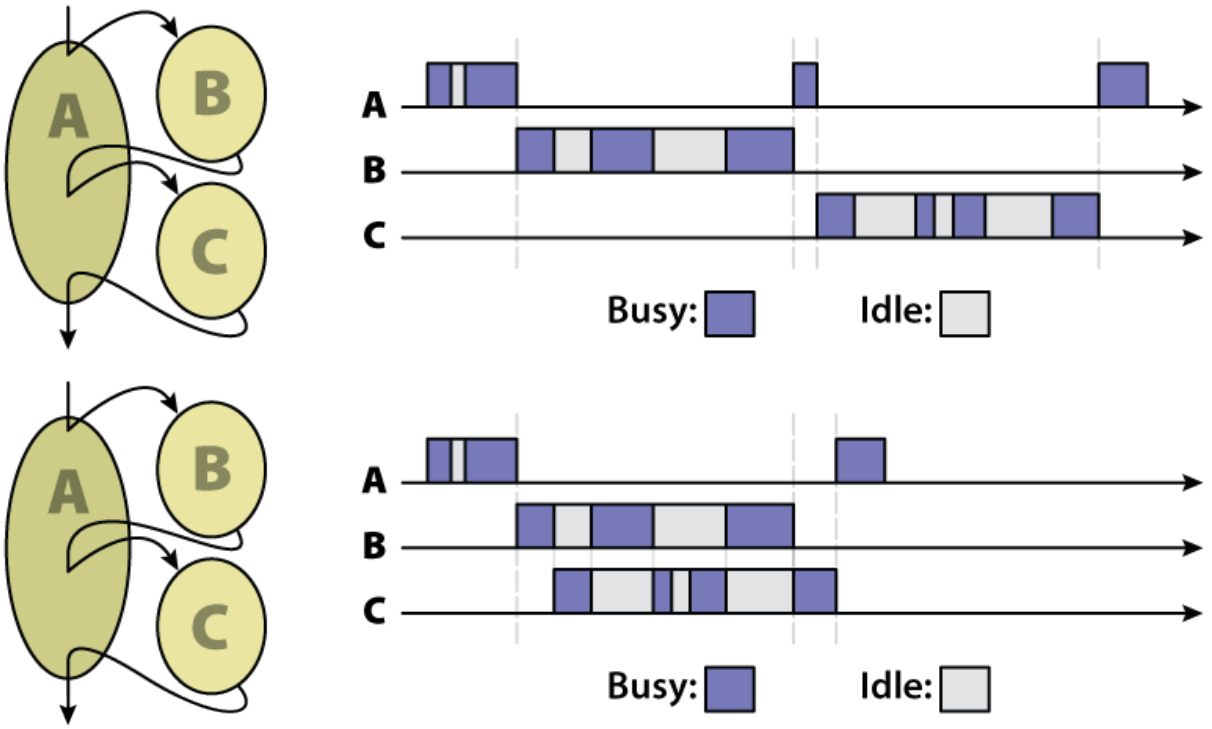


System View

Benefits

- Software engineering
  - Number of virtual processors can be independently controlled
  - Separate VPs for different modules
- Message driven execution
  - Adaptive overlap of communication
  - Predictability :
    - Automatic out-of-core
  - Asynchronous reductions
- Dynamic mapping
  - Heterogeneous clusters
    - Vacate, adjust to speed, share
  - Automatic checkpointing
  - Change set of processors used
  - Automatic dynamic load balancing
  - Communication optimization

Adaptive overlap and modules



SPMD and Message-Driven Modules

(From A. Gursoy, *Simplified expression of message-driven programs and quantification of their impact on performance*, Ph.D Thesis, Apr 1994)

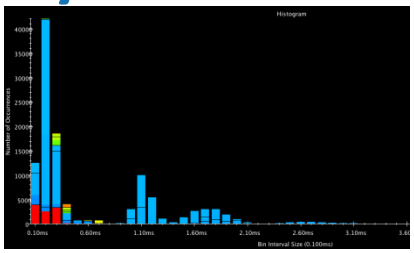
Modularity, Reuse, and Efficiency with Message-Driven Libraries: Proc. of the Seventh SIAM Conference on Parallel Processing for Scientific Computing, San Fransisco, 1995

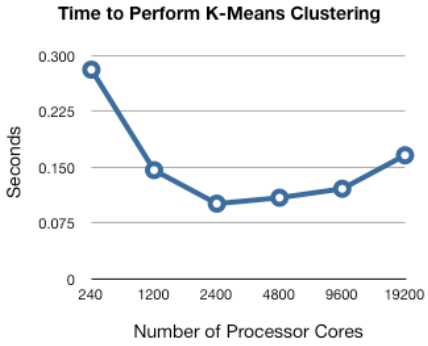
Overview

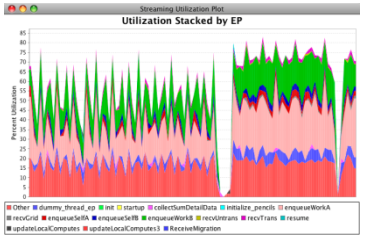
- Context
- History
- Philosophy
- Over-decomposition into Migratable Objects
- Some Current Research Efforts
  - Scalable Performance Analysis
  - Automatic Performance Tuning
  - New types of Runtime System Adaptation
  - Load Balancing
  - Topology Mapping
  - Accelerators
  - Incomplete but Useful Languages
- Promising Features in Charm++ for the Future

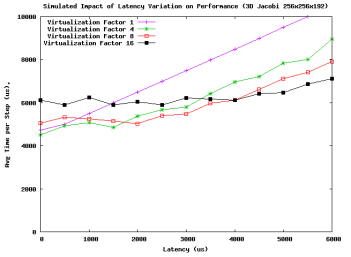
Scalable Performance Analysis

- Programming models must support **scalable** performance analysis or automatic tuning!
- Scalable performance analysis idioms
  - Must do in parallel
  - Use end-of-run when machine is available to you
  - E.g. parallel k-means clustering
- Live streaming of performance data
  - stream live performance data out-of-band in user-space to enable powerful analysis idioms



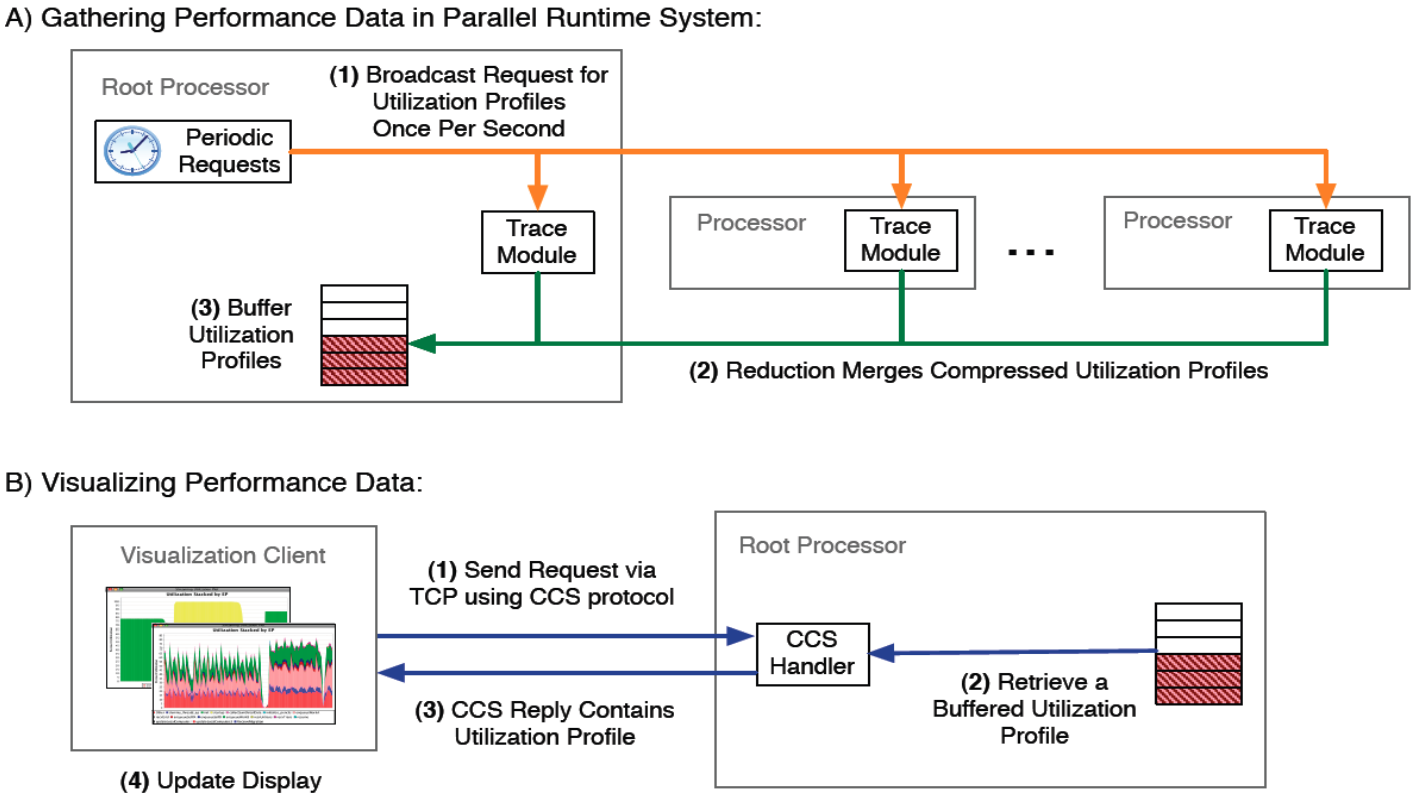






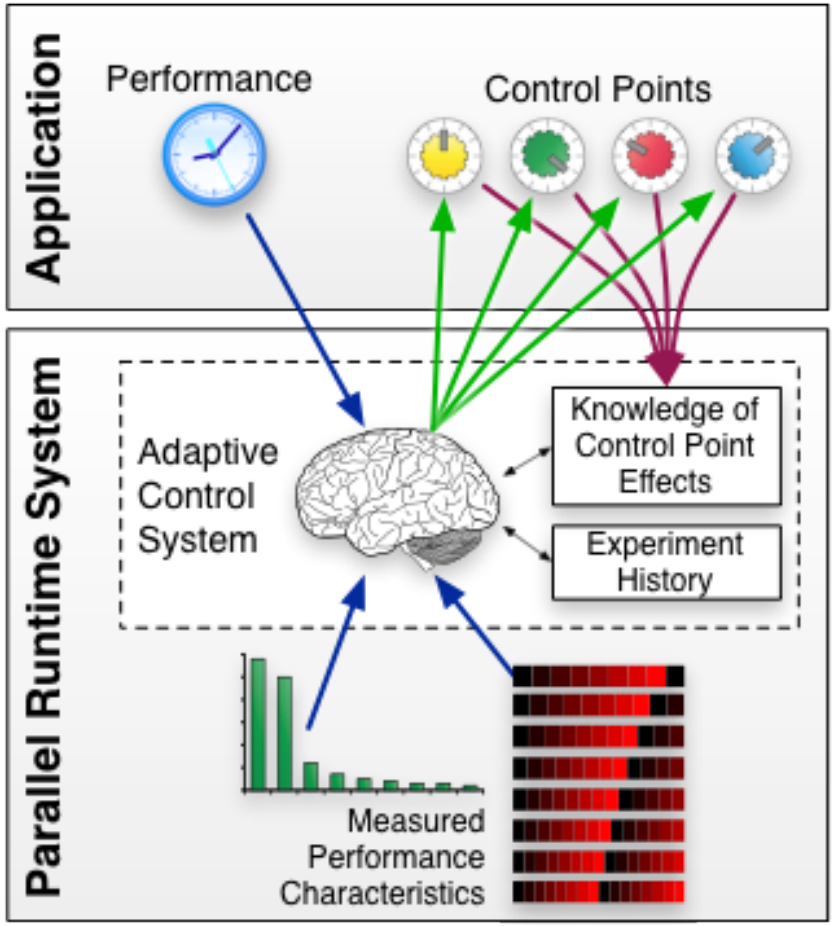
Live Streaming System Overview

- Interleave/Compose performance monitoring with application execution



Automatic Performance Tuning

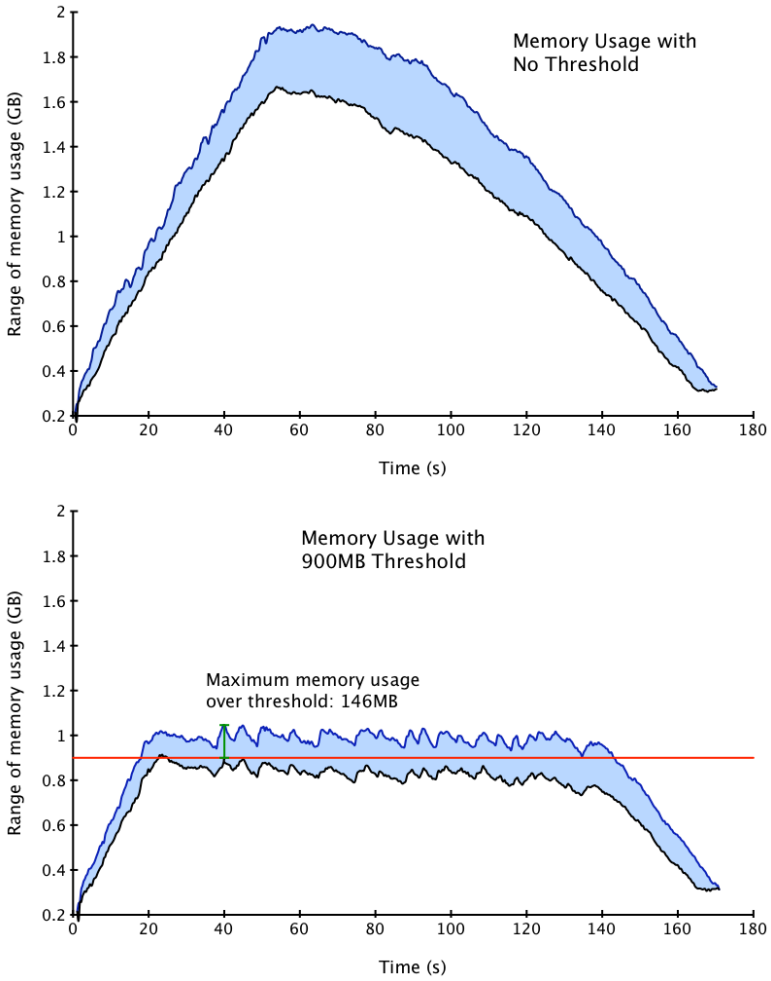
- The runtime system dynamically reconfigures applications
- Tuning/Steering is based on runtime observations :
  - Idle time, overhead time, grain size, # messages, critical paths, etc.
- Applications expose in structured manner: tunable parameters AND information about the parameters



Isaac Dooley, and Laxmikant V. Kale, *Detecting and Using Critical Paths at Runtime in Message Driven Parallel Programs,* 12th Workshop on Advances in Parallel and Distributed Computing Models (APDCM 2010) at IPDPS 2010.

New Types of Dynamic Adaptation

- Memory Aware Scheduling
- The Charm++ scheduler was modified to adapt its behavior.
- It can give preferential treatment to annotated entry methods when available memory is low.
- The memory usage for an LU Factorization program is reduced, enabling further scalability.



Isaac Dooley, Chao Mei, Jonathan Lifflander, and Laxmikant V. Kale, *A Study of Memory-Aware Scheduling in Message Driven Parallel Programs,* PPL Technical Report 2010

Scalability & Performance Summary

- We think dynamic adaptation is good.
- Automatic Adaptation Is Necessary
  - Manual tuning of even simple applications requires super-experts!
  - There are dynamically changing performance characteristics of application and of the system.
  - Need good support for intelligent automatic tuning?
  - Need good support for partially automated performance analysis?
- Need Composability of:
  - Applications
  - Performance Measuring
  - Intelligent Autotuning
  - Runtime system adaptation
  - …

Large Scale Parallelism: Load Balancing at Petascale

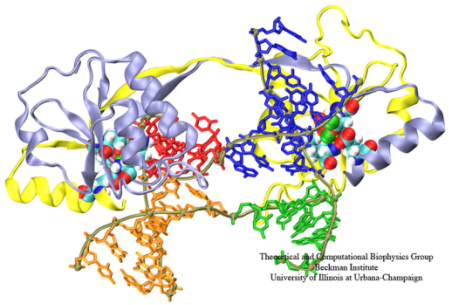
- Our older load balancing strategies don't scale on extremely large machines
  - Consider an application with 1M objects on 64K processors

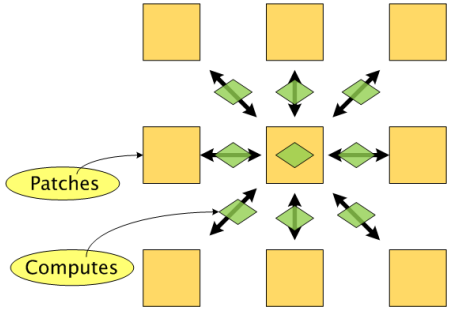
- Centralized
  - Object load data are sent to processor 0
  - Integrate to a complete object graph
  - Migration decision is broadcast from processor 0
  - Global barrier

- Distributed
  - Load balancing among neighboring processors
  - Build partial object graph
  - Migration decision is sent to its neighbors
  - No global barrier
- Topology-aware
  - On 3D Torus/Mesh topologies

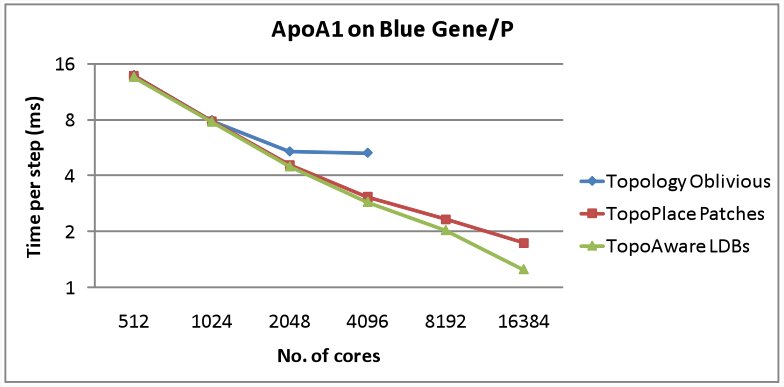
Mapping Objects onto Machine Interconnect Topology

Charm++ Applications

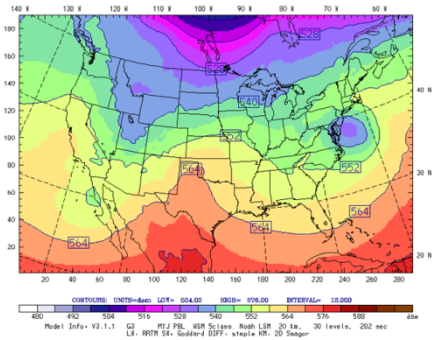


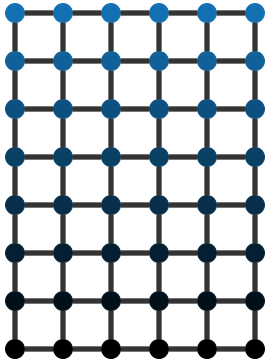


Molecular Dynamics - NAMD

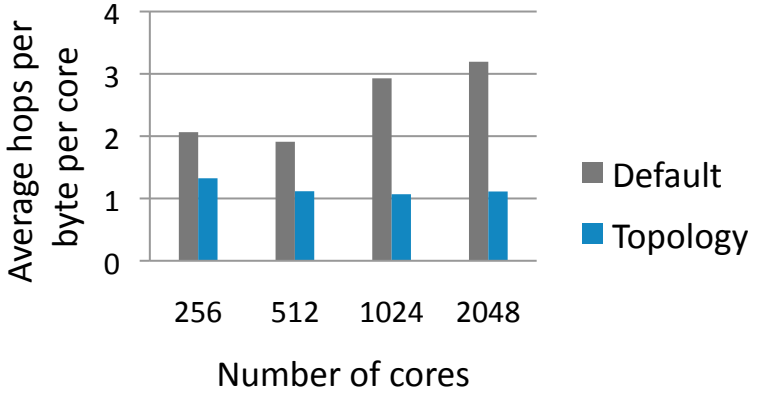


MPI Applications





Weather Research & Forecasting Model



Load Balancing Summary

- Need appropriate load balancing algorithms
- Many concerns:
  - Need excellent quality load balance (comm. & compute)
  - Machine interconnect topology
  - Complex hierarchy of processor cores
  - Cost of performing each load balancing operation
- Charm++ model, due to its encouraging of overdecomposing, provides many load balancing opportunities.

Accelerators and Heterogeneity

- GPUs, Larrabee, IBM Cell processor, ..
- It turns out that some of the Charm++ features are a good fit for these
  - Charm++ model already decomposes applications into small somewhat self-contained pieces.
- With minor changes to Charm++, tighter encapsulations of objects helps bridge barriers to multiple-memory-space platforms (or non-cache coherent systems)

Kunzman and Kale, *Towards a Framework for Abstracting Accelerators in Parallel Applications: Experience with Cell,* finalist for best student paper at SC09

Incomplete But Useful Languages

- Why use just one(or two) languages/paradigms?
- High level parallel languages:
  - Charisma: static data flow
  - SDAG: static & dynamic data flow
  - Multiphase Shared Arrays: shared arrays with access modes
  - Charj: Java-like version of Charm++ that supports better compiler generation of utility code (efficient object serialization, …)
- Multiple paradigms coexist with each other and can be interleaved in time (message driven scheduling)!

Overview

- Context
- History
- Philosophy
- Over-decomposition into Migratable Objects
- Some Current Research Efforts
  - Scalable Performance Analysis
  - Automatic Performance Tuning
  - New types of Runtime System Adaptation
  - Load Balancing
  - Topology Mapping
  - Accelerators
  - Incomplete but Useful Languages
- Promising Features in Charm++ for the Future

Promising Features in Charm++ for the Future

1. Overdecomposition
   - Portability to exotic architectures, better encapsulation
   - Greater ability to automatically manage (load balancing, fault tolerance, …)
2. Efficient composability of multiple modules
   - application, paradigms, libraries, performance analysis, autotuning, debugging, …
3. Automatic management where possible
   - Adaptation within runtime system, and in application

Exascale?

- Exascale machines will look different.
  - Power consumption, exotic/new architectures, …
- Charm++ looks like a promising candidate with sufficient margins & flexibility to fit onto a wide range of conceivable architectures.
- Charm++ model is complex, but then again so are many other choices *(which require super-experts to achieve good performance)*:
  - PGAS+threads+cuda+...
  - MPI+OpenMP+OpenCL+…

Final Summary

- Emerging parallel machines:
  - Big
  - Differing architectural designs
  - Increasingly hard to program
- Charm++ model has some interesting and promising features that address some of these difficulties:
  - Overdecomposition
  - Efficient composition

Questions?

Charm++: on the road to Exascale

Isaac Dooley

PEEPS @ VecPar

Berkeley, June 2010

ILLINOIS

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

PARALLEL PROGRAMMING LAB

DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS