Parallel Programming with Charm++

Phil Miller, Ramprasad Venkataraman, Laxmikant Kalé

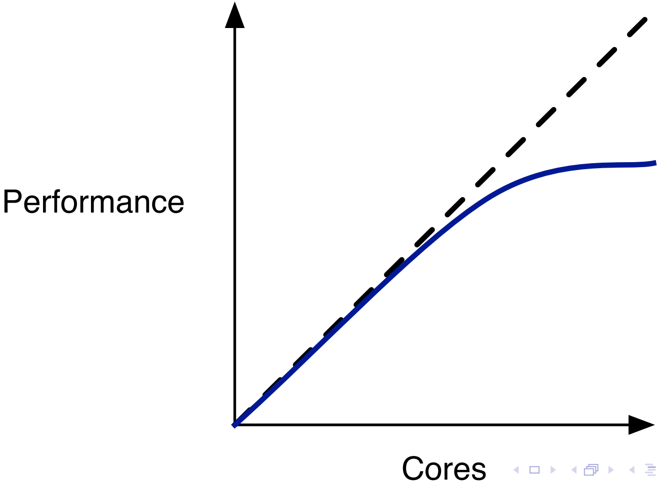
Parallel Programming Lab
University of Illinois
charmplusplus.org

May 14, 2012

Audience Poll: Challenges of Parallel Programming

How many of you have written parallel programs that suffer from:

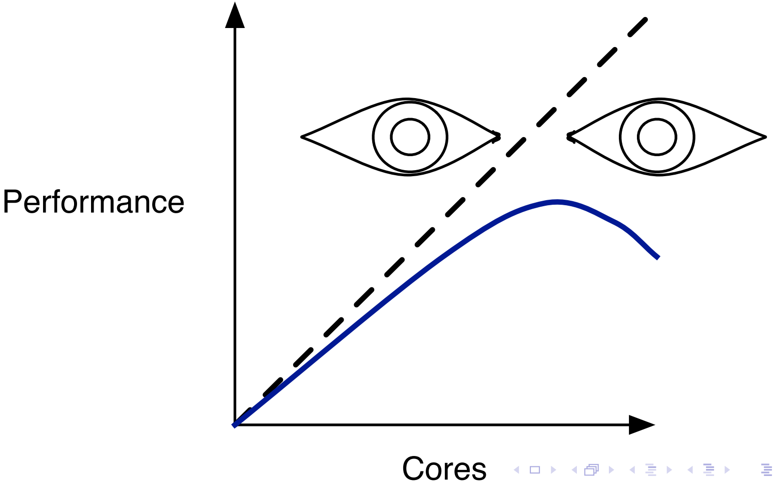
- Bad scaling?



Audience Poll: Challenges of Parallel Programming

How many of you have written parallel programs that suffer from:

- Sad scaling?



Audience Poll: Challenges of Parallel Programming

How many of you have written parallel programs that suffer from:

- Bad scaling?
- Races, deadlocks, etc: gremlins of shared state?

Audience Poll: Challenges of Parallel Programming

How many of you have written parallel programs that suffer from:

- Bad scaling?
- Races, deadlocks, etc: gremlins of shared state?
- Limited to shared memory? GPU? No sharing allowed?

Audience Poll: Challenges of Parallel Programming

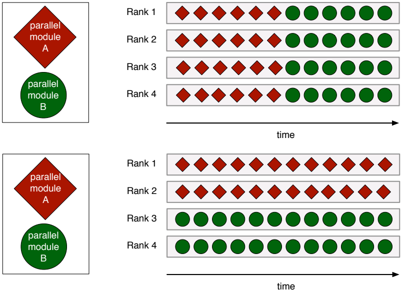
How many of you have written parallel programs that suffer from:

- Bad scaling?
- Races, deadlocks, etc: gremlins of shared state?
- Limited to shared memory? GPU? No sharing allowed?
- Coded to match core count?

Audience Poll: Challenges of Parallel Programming

How many of you have written parallel programs that suffer from:

- Bad scaling?
- Races, deadlocks, etc: gremlins of shared state?
- Limited to shared memory? GPU? No sharing allowed?
- Coded to match core count?
- Independent tasks serialized or badly split across resources?



Audience Poll: Challenges of Parallel Programming

How many of you have written parallel programs that suffer from:

- Bad scaling?
- Races, deadlocks, etc: gremlins of shared state?
- Limited to shared memory? GPU? No sharing allowed?
- Coded to match core count?
- Independent tasks serialized or badly split across resources?
- Application logic interwoven with parallelism optimizations?

Audience Poll: Challenges of Parallel Programming

How many of you have written parallel programs that suffer from:

- Bad scaling?
- Races, deadlocks, etc: gremlins of shared state?
- Limited to shared memory? GPU? No sharing allowed?
- Coded to match core count?
- Independent tasks serialized or badly split across resources?
- Application logic interwoven with parallelism optimizations?
- Wasted energy?

Audience Poll: Challenges of Parallel Programming

How many of you have written parallel programs that suffer from:

- Bad scaling?
- Races, deadlocks, etc: gremlins of shared state?
- Limited to shared memory? GPU? No sharing allowed?
- Coded to match core count?
- Independent tasks serialized or badly split across resources?
- Application logic interwoven with parallelism optimizations?
- Wasted energy?
- Square-peg logic in round-hole framework abstractions?

Charm++

Parallel ...

- ... programming model
- ... programming framework
- ... runtime system

Charm++

Parallel ...

- ... programming model
- ... programming framework
- ... runtime system

- General-purpose
- Macro Dataflow
- Unified data and task parallelism
- Unified handling of shared and distributed memory
- Parallel algorithm independent of available processors
- Seamless parallel composability of modular components

Charm++

Parallel ...

- ... programming model
- ... programming framework
- ... runtime system

- Code generation, Base classes, utility functions and other API
- Multi-paradigm parallel code (procedural, object oriented, generic)
- Rich ecosystem of tools
- Separation of roles and concerns

Charm++

Parallel ...

- ... programming model
- ... programming framework
- ... runtime system

- Managed parallel execution
- Measurement-based performance introspection
- Adaptive response for better performance
  - Fault tolerance
  - Dynamic load balancing
  - Energy management

Charm++: Portability

Environments

- Embedded ARM: CARMA dev boards, cell phones
- Commodity x86: servers, desktops, laptops, tablets
- Clusters: commodity, with a network
- Supercomputers: IBM Blue Gene and POWER, Cray

Charm++: Portability

Environments

- Embedded ARM: CARMA dev boards, cell phones
- Commodity x86: servers, desktops, laptops, tablets
- Clusters: commodity, with a network
- Supercomputers: IBM Blue Gene and POWER, Cray

Operating Systems

- Linux
- Mac OS X
- Windows
- Proprietary Cray & IBM

Charm++: Portability

Environments

- Embedded ARM: CARMA dev boards, cell phones
- Commodity x86: servers, desktops, laptops, tablets
- Clusters: commodity, with a network
- Supercomputers: IBM Blue Gene and POWER, Cray

Operating Systems

- Linux
- Mac OS X
- Windows
- Proprietary Cray & IBM

Network Interfaces

- TCP, UDP
- Shared memory
- MPI
- Infiniband Verbs
- IBM BlueGene P,Q (DCMF, PAMI)
- Cray Gemini and Aries (uGNI)

Charm++: Portability

Environments

- Embedded ARM: CARMA dev boards, cell phones
- Commodity x86: servers, desktops, laptops, tablets
- Clusters: commodity, with a network
- Supercomputers: IBM Blue Gene and POWER, Cray

Operating Systems

- Linux
- Mac OS X
- Windows
- Proprietary Cray & IBM

Compilers

- GCC
- Clang
- Microsoft VC++
- IBM XL
- Intel
- Portland Group (PGI)
- Cray
- Fujitsu

Charm++: Pedigree

- 1987: Chare Kernel arose from parallel Prolog work
- 1992: Initial C++-based Charm++
- 1994-1996: NAMD developed
- 1997: Application-facing abstractions reach near-current form
- 1997: Adaptive MPI (AMPI) built atop Charm++
- 2000-present: More applications developed, runtime facilities extended, scaling with new machines

Charm++: Pedigree

- 1987: Chare Kernel arose from parallel Prolog work
- 1992: Initial C++-based Charm++
- 1994-1996: NAMD developed
- 1997: Application-facing abstractions reach near-current form
- 1997: Adaptive MPI (AMPI) built atop Charm++
- 2000-present: More applications developed, runtime facilities extended, scaling with new machines

Award-winning

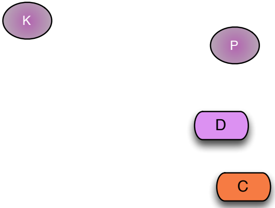
Gordon Bell award in 2002
HPC Challenge award in 2011
Sidney Fernbach award for Kalé in 2012

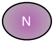
several best papers

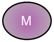
Express parallel algo independent of processors

Use units natural to domain

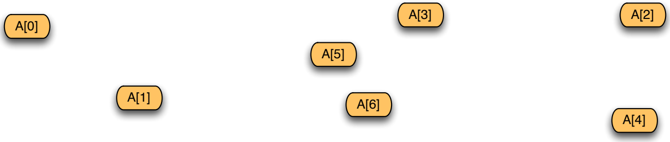
- matrix block
- tile of an image
- slice of a computation's work
- volume of simulation space
- partition of a graph, tree or other data structures





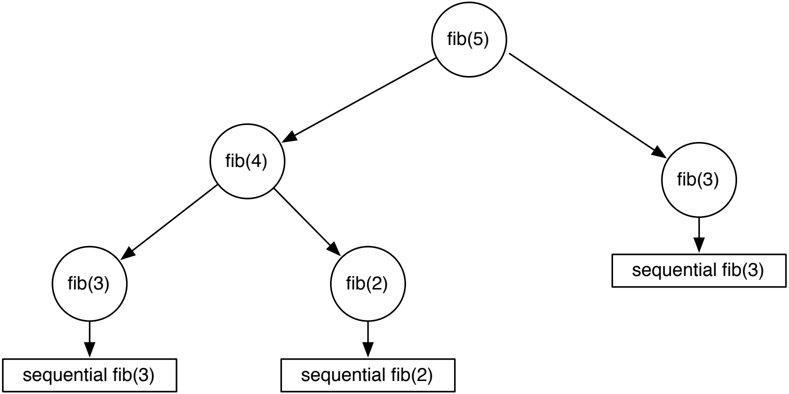


Data decomposition: via an object collection

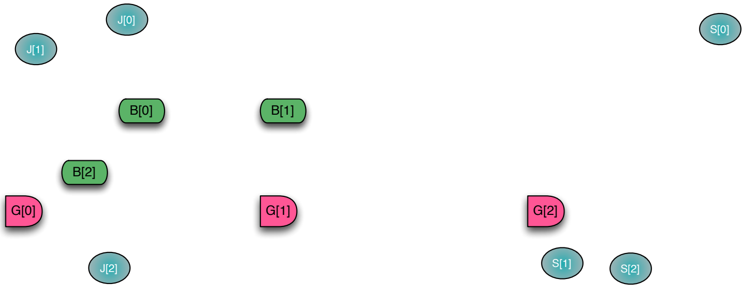


Multiple data parallel collections

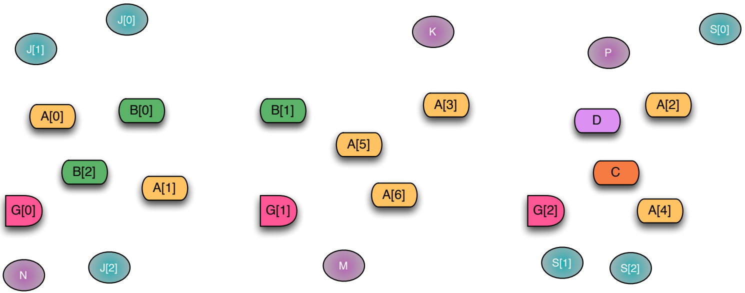
Work decomposition: also via objects / collections



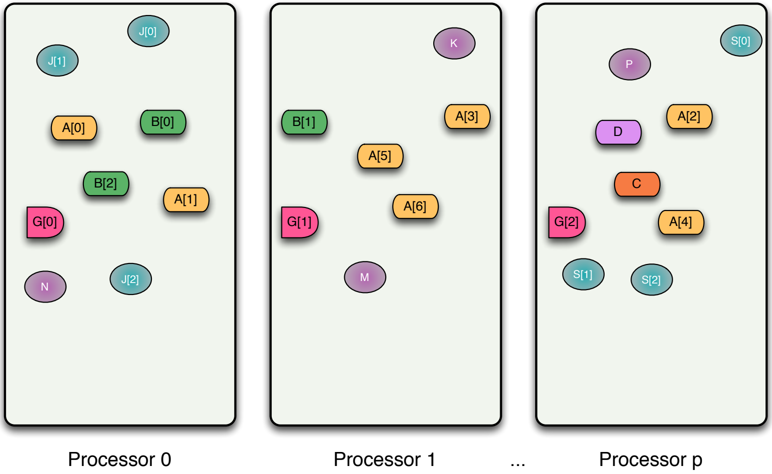
Functional decomposition: via multiple classes



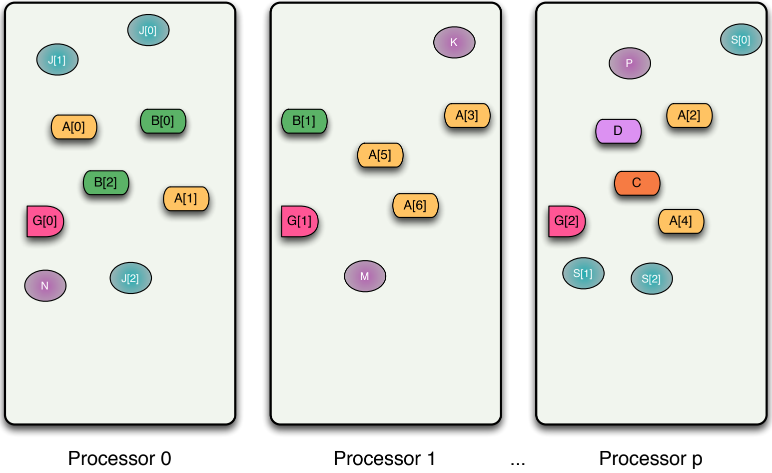
App logic: via classes and object collections



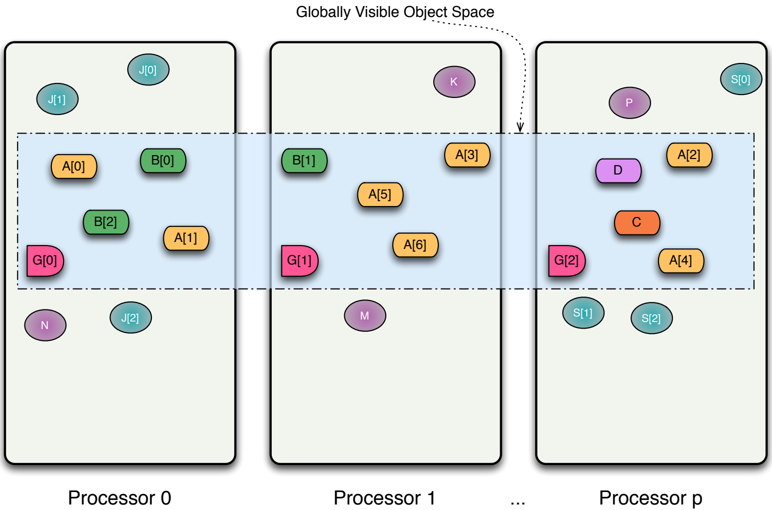
Concurrency requires placing objects on all processors



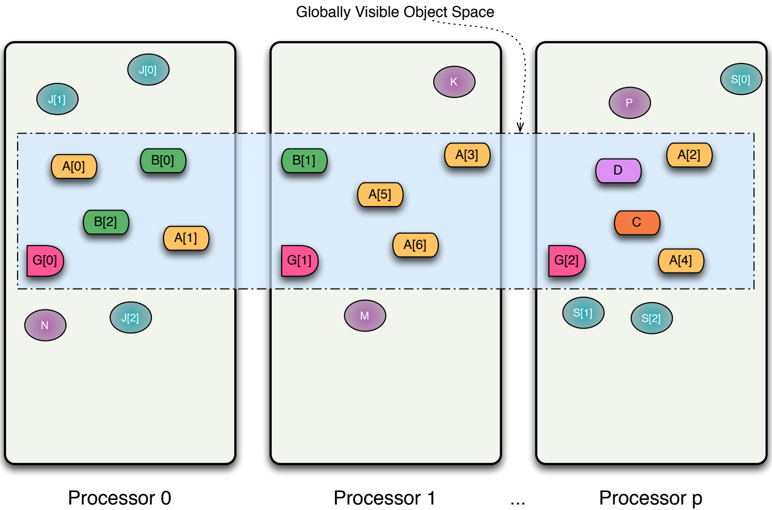
However, do not burden programmer with this view



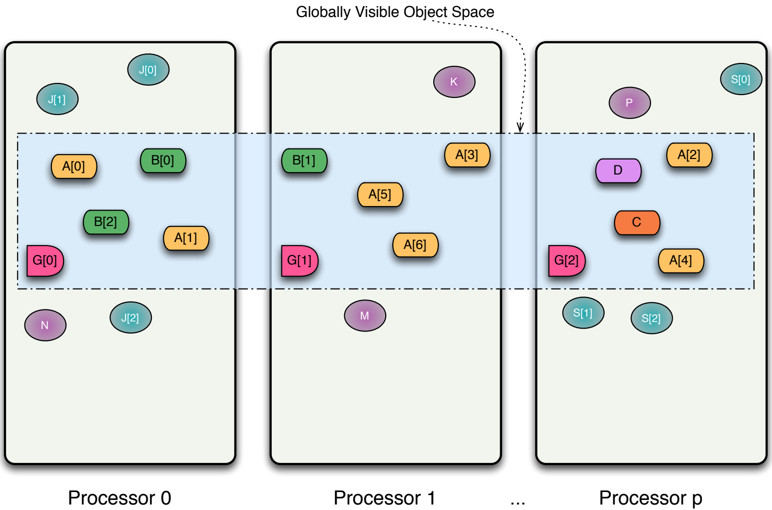
Elevate some objects to global visibility



Globally visible objects = chares



Globally visible object collections = chare arrays



Annotating classes to enable global visibility

In foo.ci

```
module foo_module {
  array [2D] Foo {
    // . . .
  };
}
```

In foo.C

```
#include "foo.h"

// . . .

#include "foo_module.def.h"
```

In foo.h

```
#include "foo_module.decl
    .h"

class Foo : public
    CBase_Foo {
  // . . .
};
```

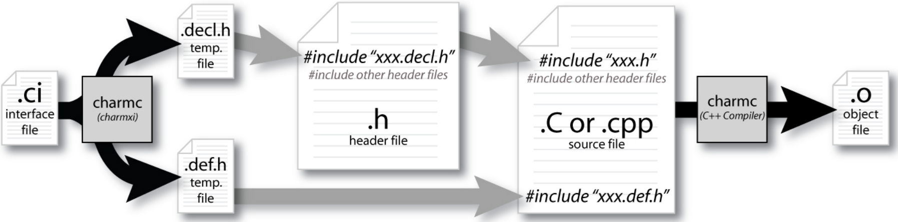
Annotating classes to enable global visibility

In foo.ci

```
module foo_module {
  array [2D] Foo {
    // . . .
  };
}
```

In foo.C

```
#include "foo.h"

// . . .

#include "foo_module.def.h"
```



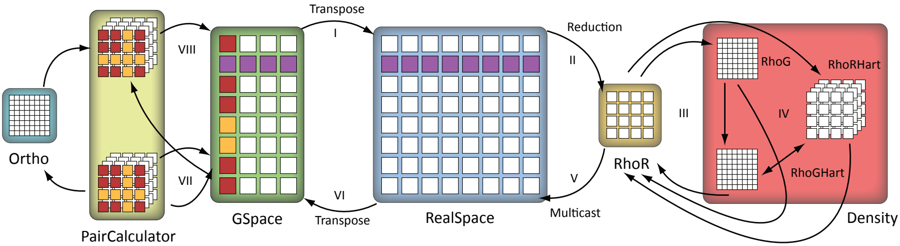
Indexing into Object Collections

In foo.ci

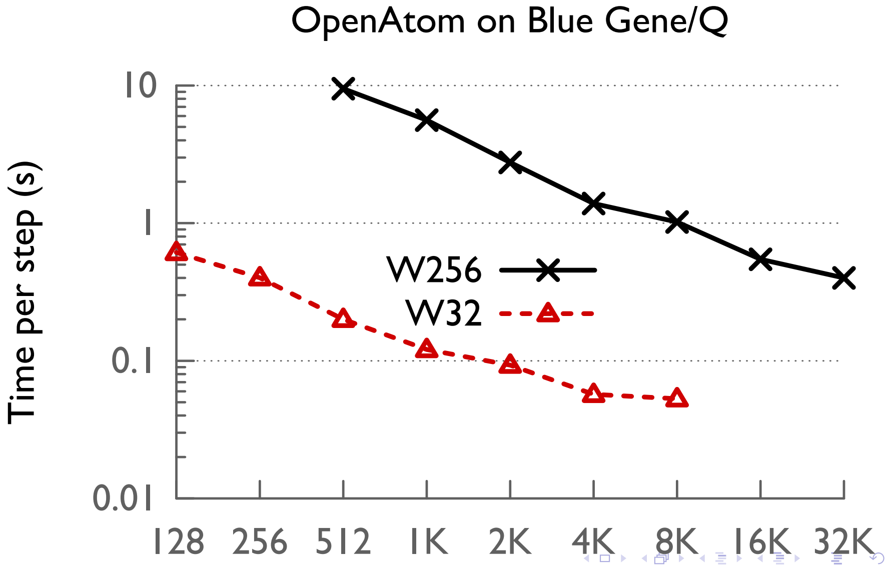
```
module foo_module {
  array [2D] Foo {
    // ...
  };
}
```

- multidimensional, integer (1D .. 6D)
  - Dense
  - Sparse
- anything hashable (strings, bitvectors)
- Static
- Dynamic (elements come and go)

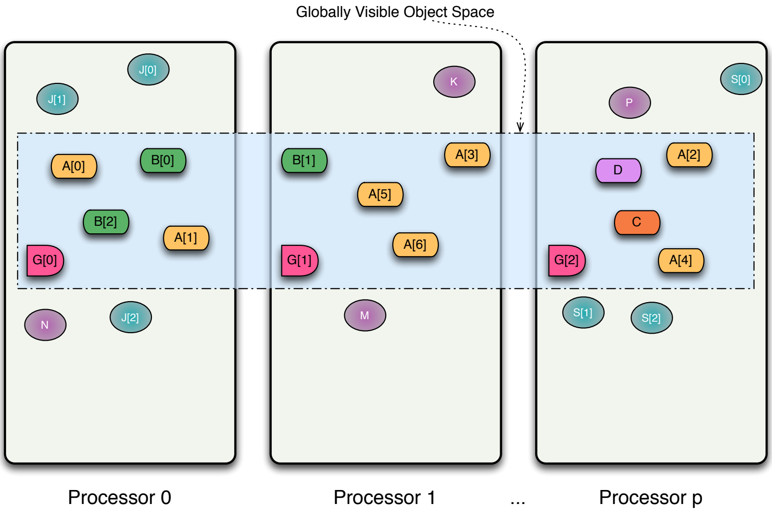
Quantum Chemistry: OpenAtom



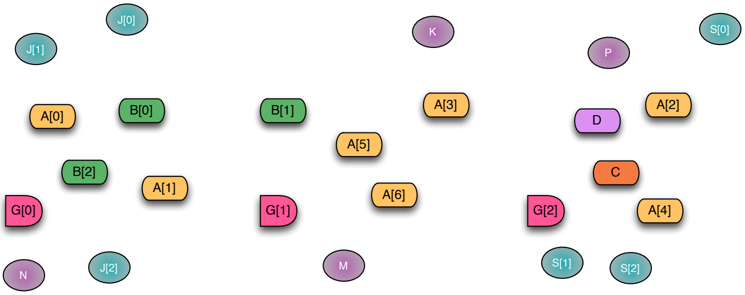
Quantum Chemistry: OpenAtom



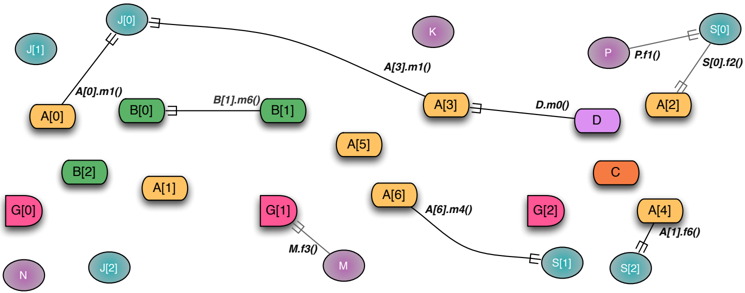
Object collections maketh not a parallel program



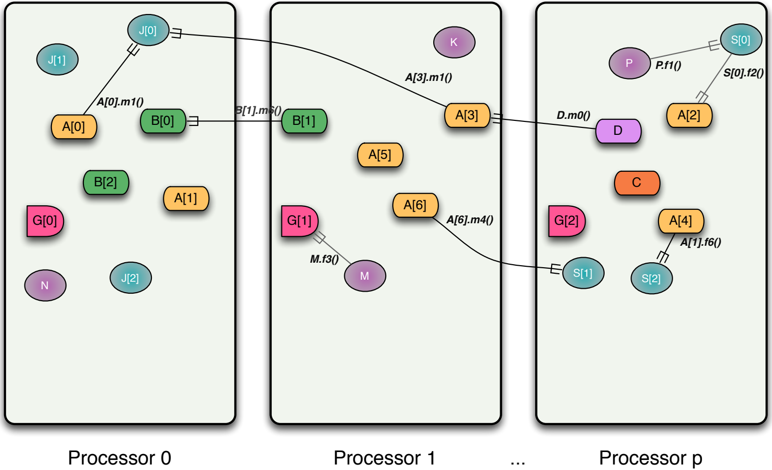
Object interactions



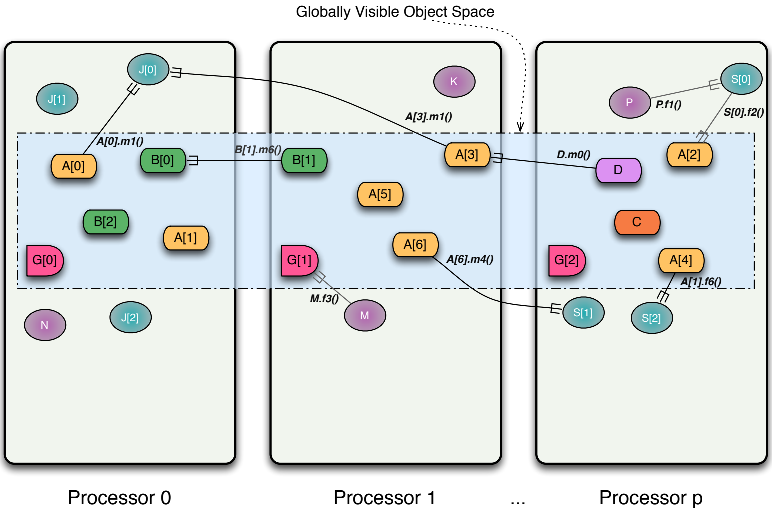
Object interactions



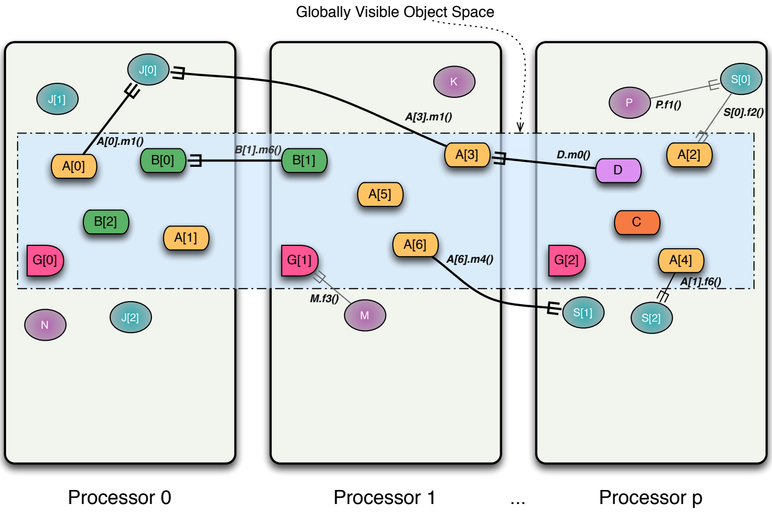
Object interactions ... via remote method invocations



1. Not every object is remotely invocable



2. Not every method is remotely invocable

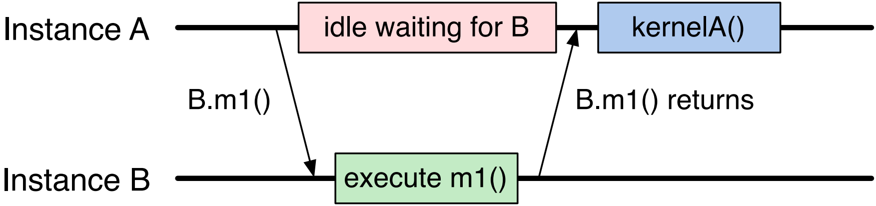


3. Remote methods are of void return type

What happens if an object waits for a return value from a method invocation?

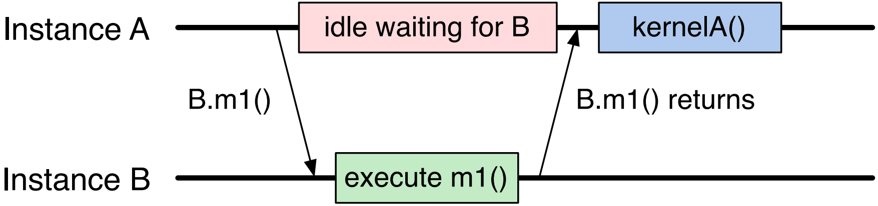
3. Remote methods are of void return type

What happens if an object waits for a return value from a method invocation?



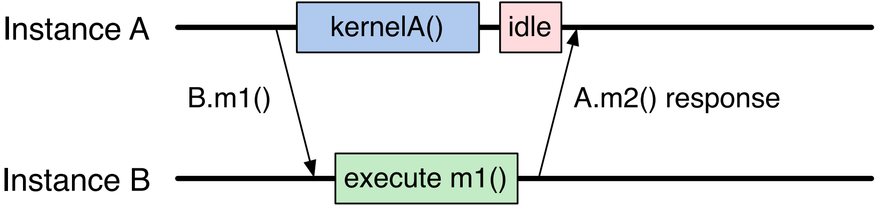
3. Remote methods are of void return type

What happens if an object waits for a return value from a method invocation?

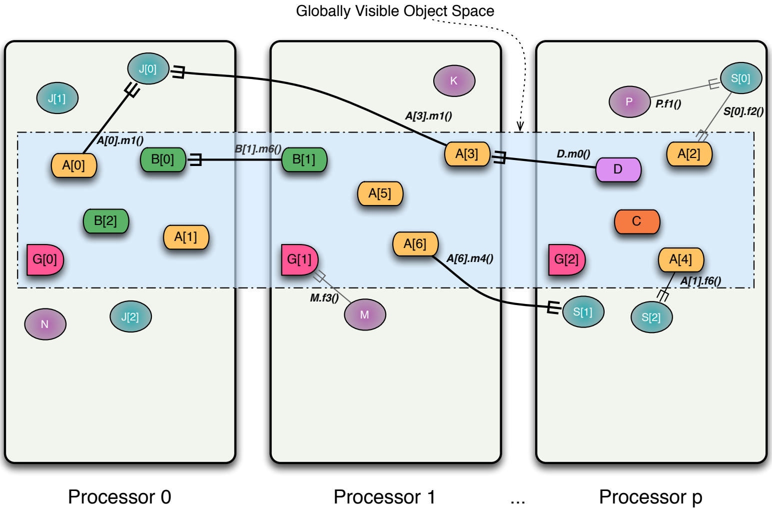


- Performance
- Latency
- Reasoning about correctness

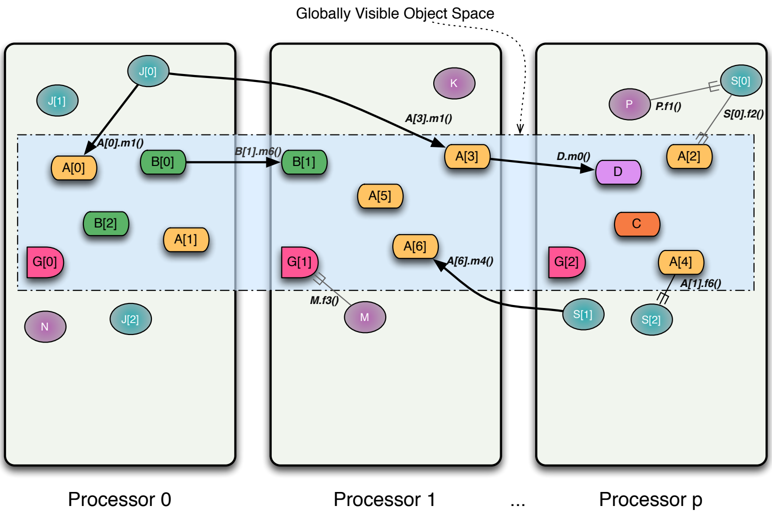
3. Remote methods are of void return type



- Hence, method invocations should be asynchronous
  - No return values
- Computations are driven by the incoming data
  - Initiated by the sender or method caller

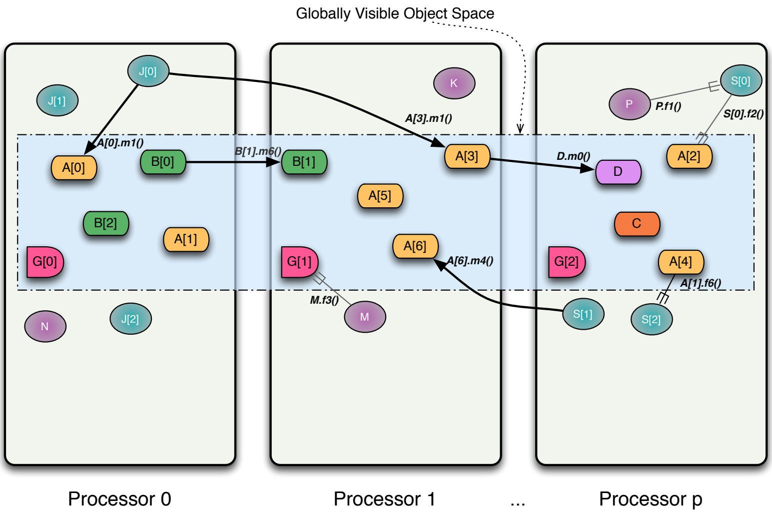

Globally Visible Object Space
J[0]
J[1]
A[0].m1()
A[0]
B[0]
B[1].m6()
B[1]
B[2]
A[1]
G[0]
N
J[2]
K
A[3].m1()
A[3]
A[5]
G[1]
A[6]
A[6].m4()
M.f3()
M
P
P.f1()
S[0]
S[0].f2()
D.m0()
D
A[2]
C
G[2]
A[4]
A[1].f6()
S[1]
S[2]
Processor 0
Processor 1
...
Processor p

Asynchronous, non-blocking remote method invocations on chares



Entry Methods

Asynchronous, non-blocking remote method invocations on chares



Globally visible entry methods

In `foo.ci`

```
array [2D] Foo {
  entry Foo(int c, double d);
  entry void compute(int count, double[count] data);
};
```

In `foo.h`

```
class Foo : public CBase_Foo {
  int c_; double d_;
public:
  Foo(int c, double d);
  void compute(int count, double * data);
};
```

In `foo.C`

```
Foo::Foo(int c, double d) : c_(c), d_(d) { }
void Foo::compute(int count, double * data)
{ /* . . . */ }
```

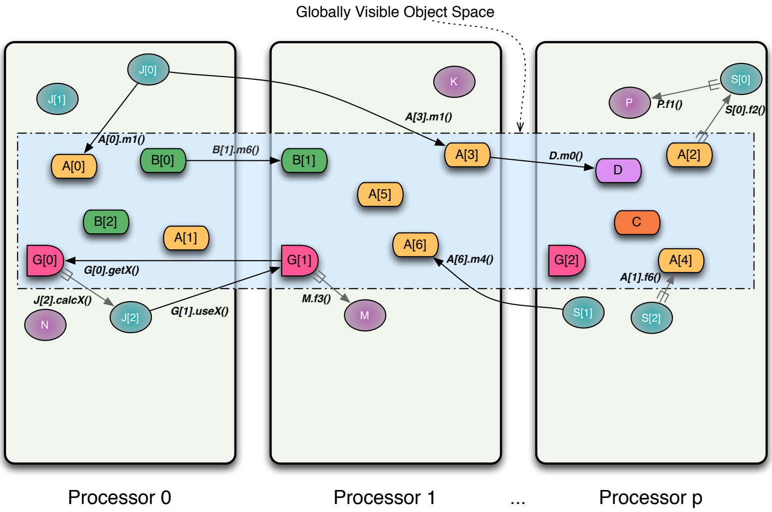
Calling Entry Methods: Proxy Objects

```
// Construct a 10*10 array of Foo chares, each initialized with {42, 2.7}
CProxy_Foo f = CProxy_Foo::ckNew(10, 10, 42, 2.7);

double d[7] = {0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6};

// Call Foo::compute(7, d) on the object at (1, 2) in the collection
f(1, 2).compute(7, d);
```

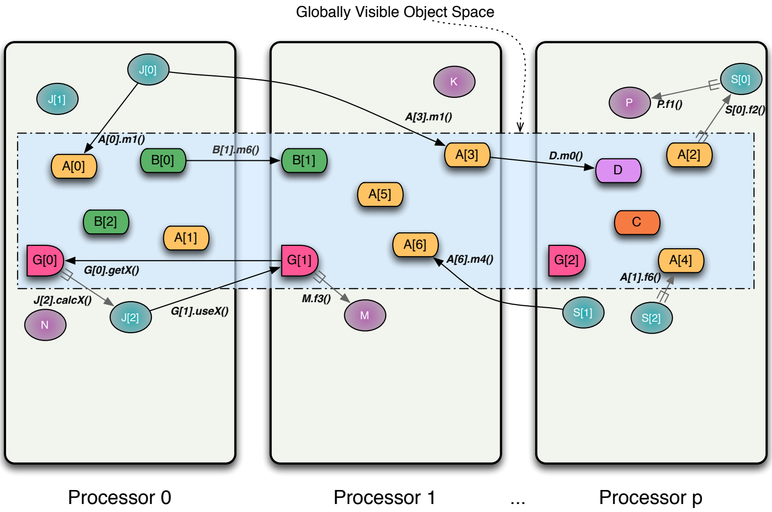
Tenet: Do not hide locality information from developer

Many RMI implementations try to hide remote-ness.
Ours draws attention to potential expense of non-local operations.
Proxy objects are explicitly visible to client code. Any invocations via proxies are potentially remote.

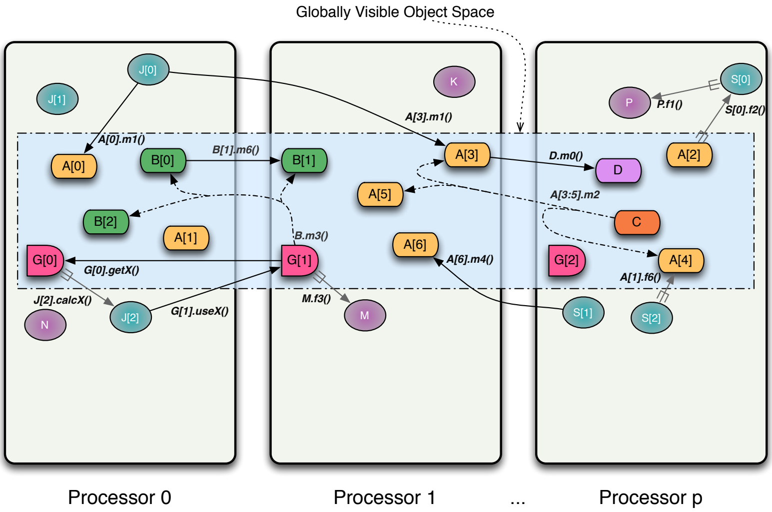
How do you get return values



Method invocation on object collections



Method invocation on object collections

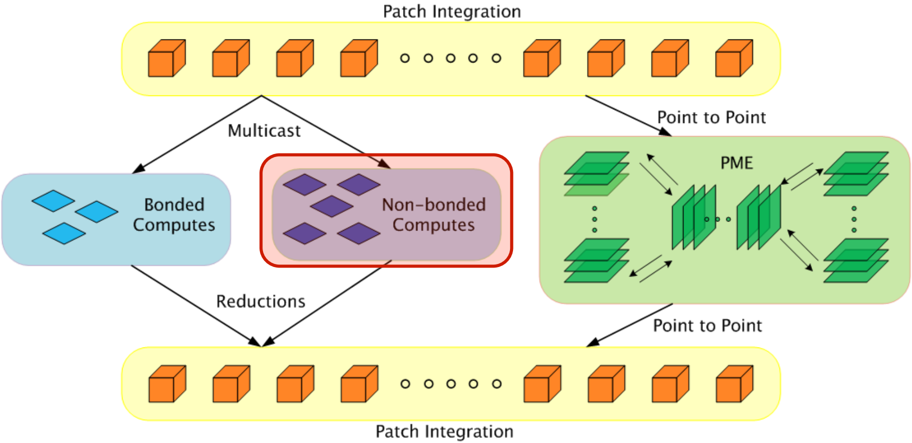


Entry methods and Dataflow

- void return types imply one-way information transfer
- signal application's intent to perform (possibly) remote task
- carry required input data for remote task
- express parallel dependencies

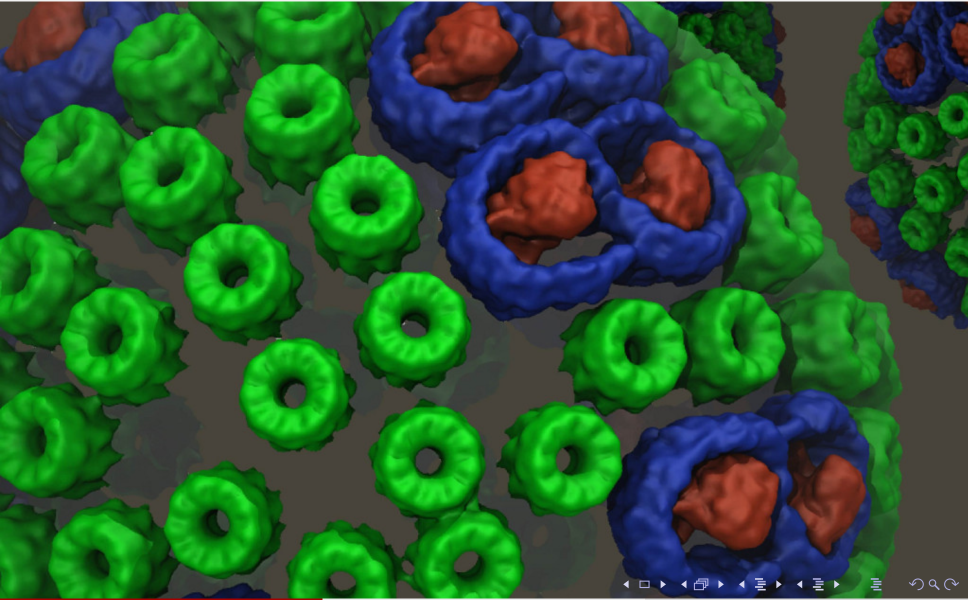
Biomolecular Physics: NAMD

Parallel decomposition and dependencies



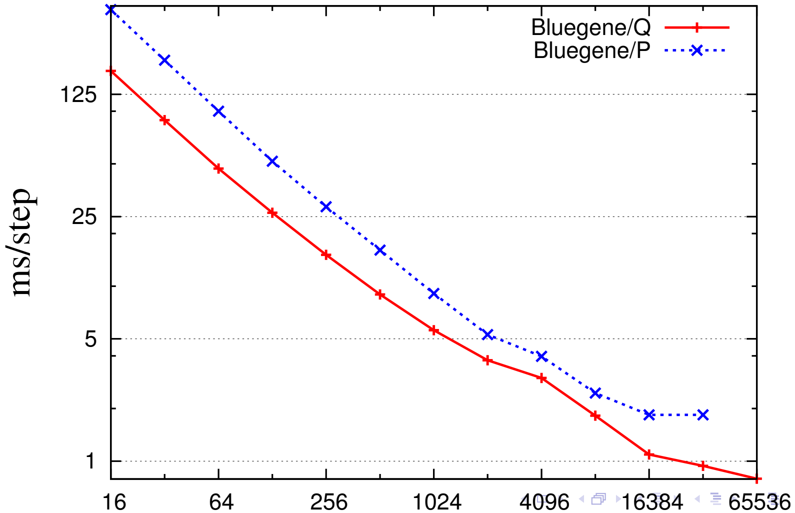
Biomolecular Physics: NAMD

Chromatophore vesicle in purple bacteria



Biomolecular Physics: NAMD

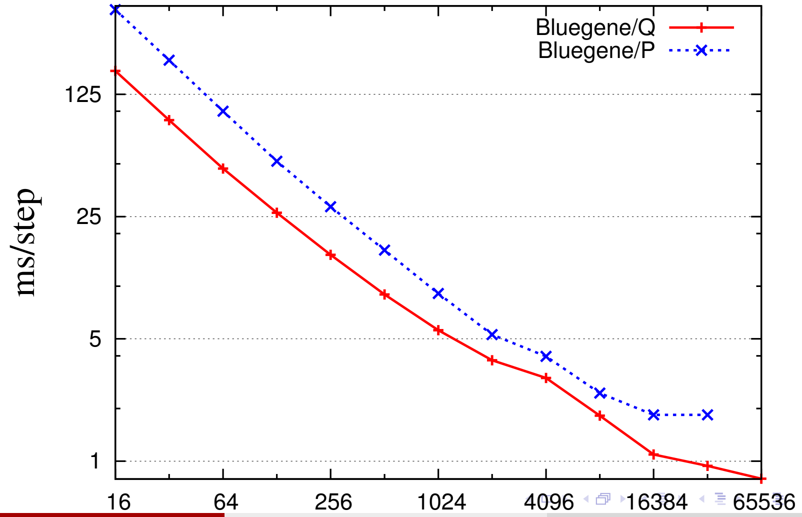
ApoA1 on IBM BlueGene P/Q (Intrepid/Mira)



Biomolecular Physics: NAMD

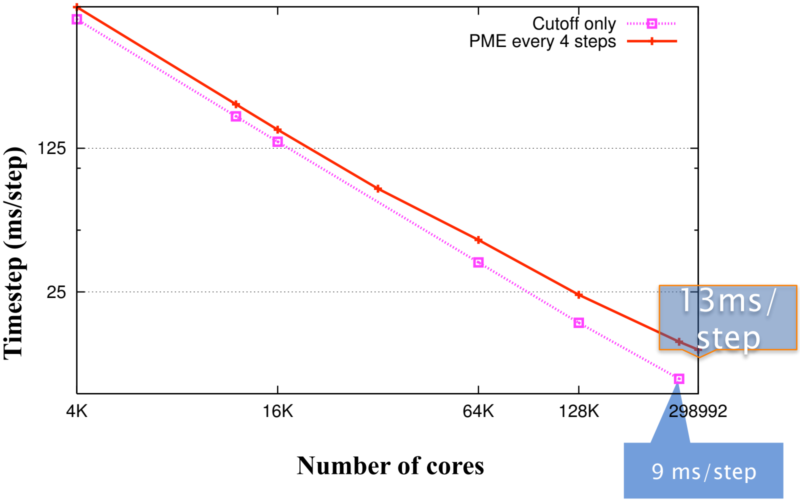
ApoA1 on IBM BlueGene P/Q (Intrepid/Mira)

794 us / step

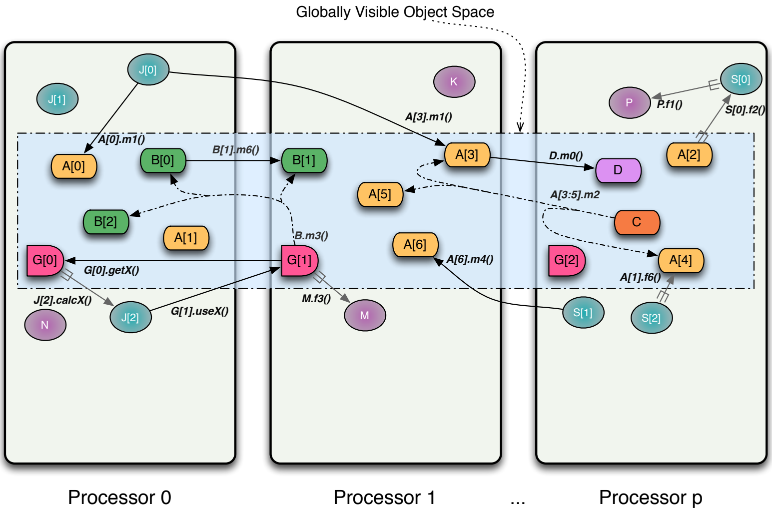


Biomolecular Physics: NAMD

100M atom STMV on Cray XK6 (Titan)



RMI → Messages



Remember the void return types?

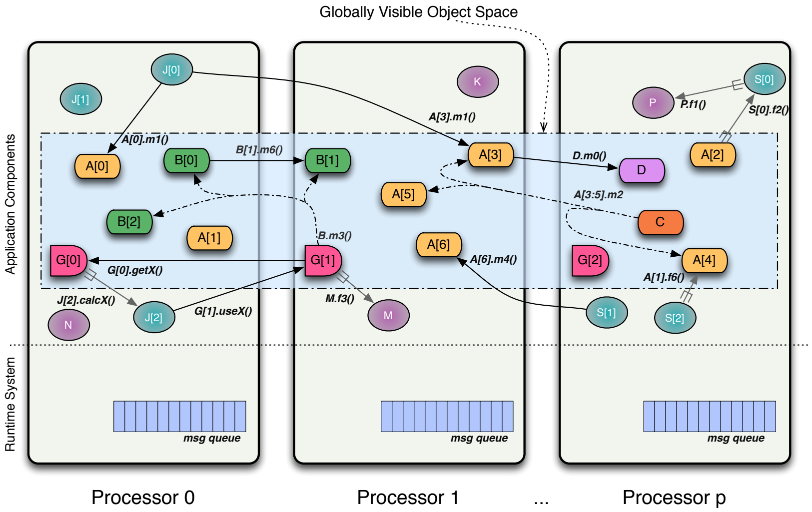
- void return types imply one-way information transfer
- signal application's intent to perform (possibly) remote task
- carry required input data for remote task
- express parallel dependencies

Remember the void return types?

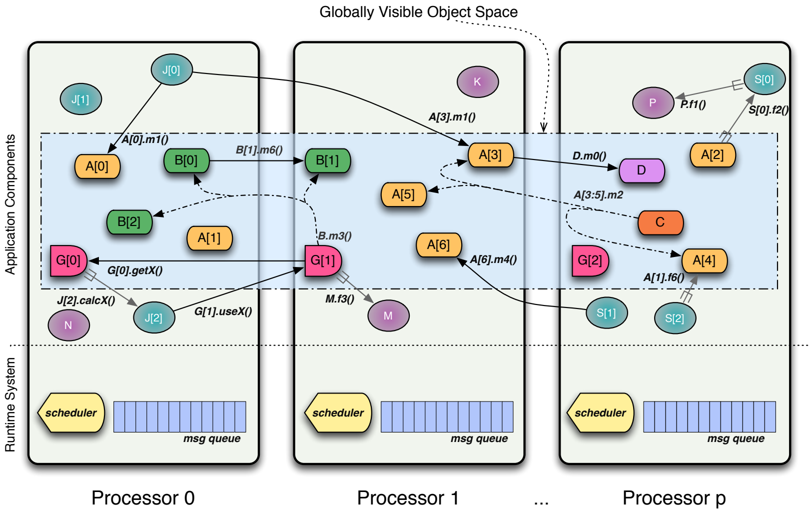
- void return types imply one-way information transfer
- signal application's intent to perform (possibly) remote task
- carry required input data for remote task
- express parallel dependencies

Entry methods express when something can execute.
Not when something should execute.

Message queues



Scheduler



Globally Visible Object Space
Application Components
Runtime System
J[0]
J[1]
K
S[0]
P
P.f1()
S[0].f2()
A[3].m1()
A[0].m1()
A[0]
B[0]
B[1].m6()
B[1]
A[3]
D.m0()
D
A[2]
A[5]
A[3:5].m2
B[2]
A[1]
C
B.m3()
A[6]
G[0]
G[1]
A[6].m4()
G[2]
A[4]
G[0].getX()
A[1].f6()
J[2].calcX()
M.f3()
S[1]
S[2]
N
J[2]
G[1].useX()
M
scheduler
msg queue
scheduler
msg queue
scheduler
msg queue
Processor 0
Processor 1
...
Processor p

Charm++

- objects = fundamental unit of state / functionality
- methods = fundamental unit of execution

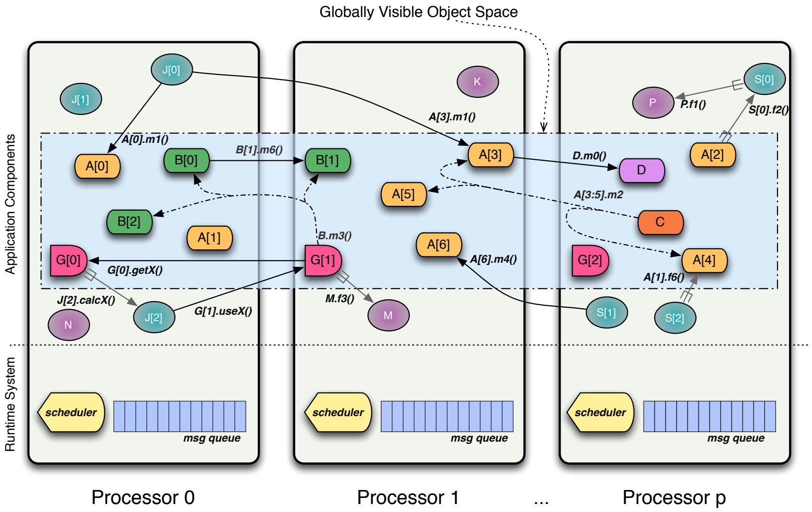
Charm++

- objects = fundamental unit of state / functionality
- methods = fundamental unit of execution

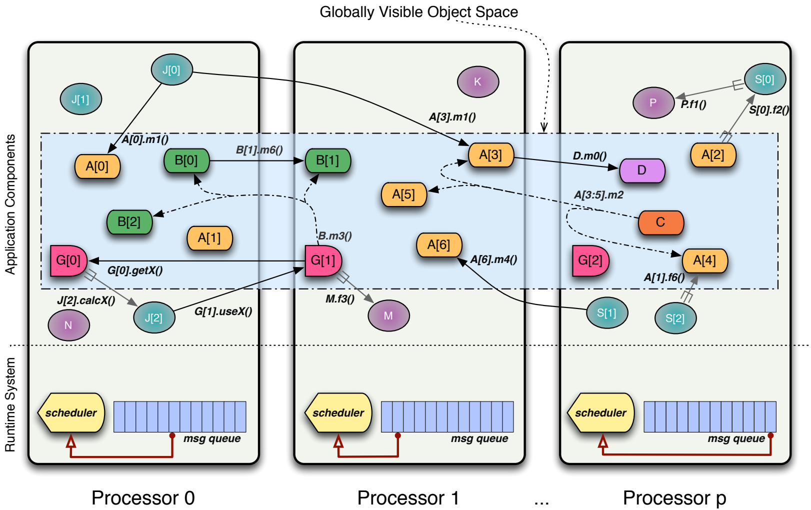
Entry Methods ...

- are *scheduled* for execution
- are not preempted
- are not reentrant
- have unspecified delivery order
- do not require threading / locking mechanisms (typically)

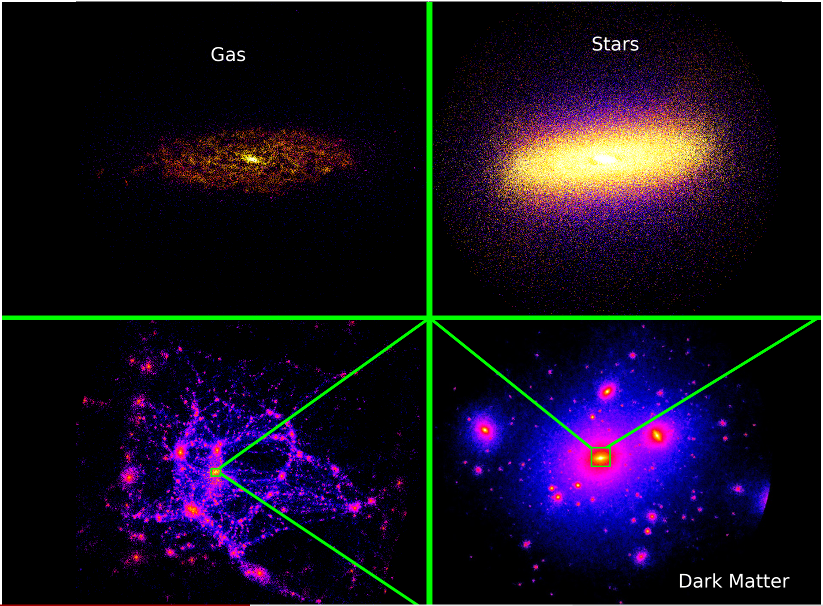
Prioritized Execution



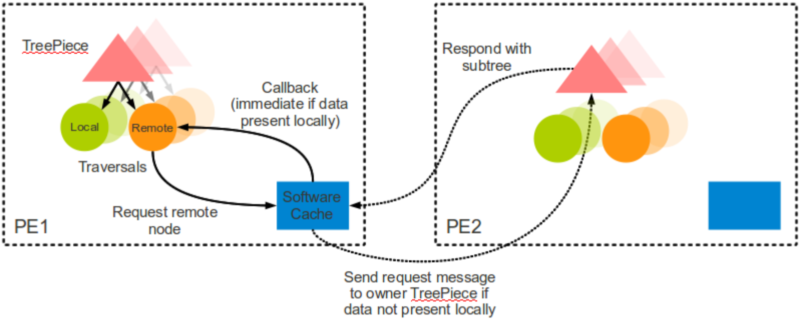
Prioritized Execution



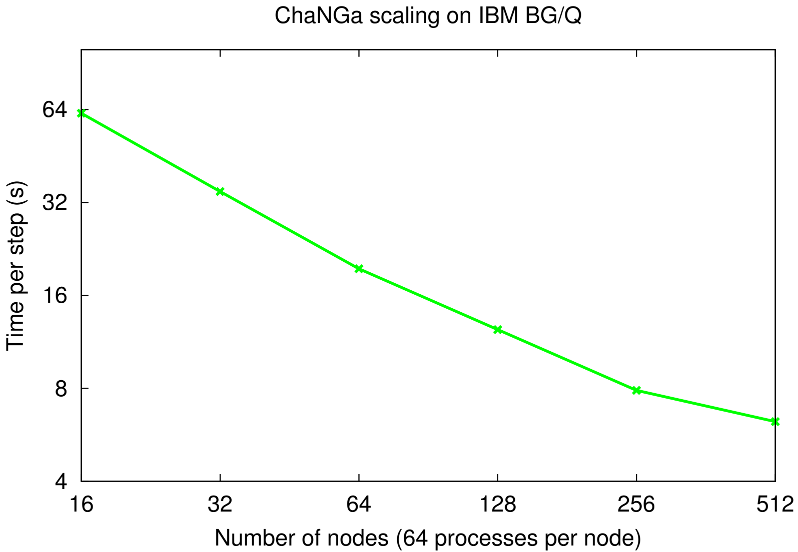
Cosmology: ChaNGa



Cosmology: ChaNGa



Cosmology: ChaNGa



Parallel Decomposition

Recap

- Data or Task parallelism encoded in objects
- Object count independent of processors
- How many objects, then? How big?

Parallel Decomposition

Overdecomposition

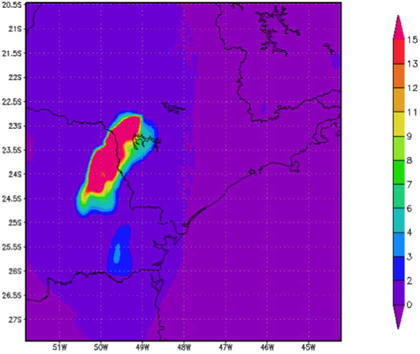
Want *several* objects per processor

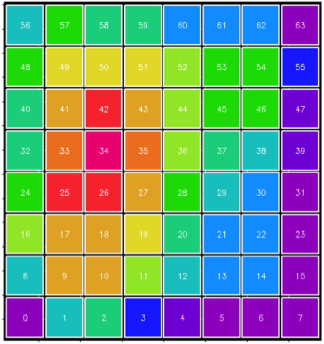
- Increase chance that one will have work available
- Overlap communication of one with computation of another
- Important for later optimizations

Parallel Decomposition

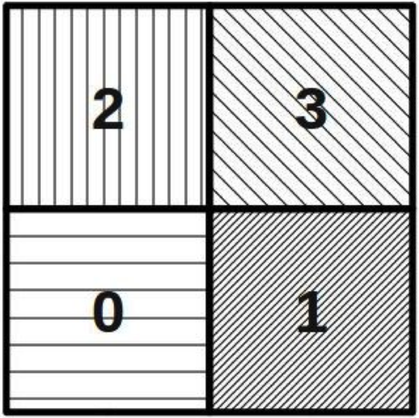
Overdecomposition Example: Weather Forecasting in BRAMS

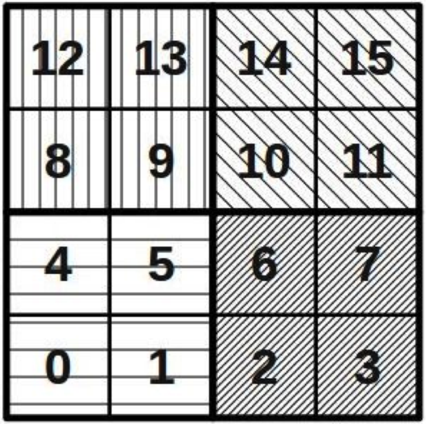
- BRAMS: Brazillian weather code (based on RAMS)
- AMPI version (Eduardo Rodrigues, with C. Mendes and J. Panetta)



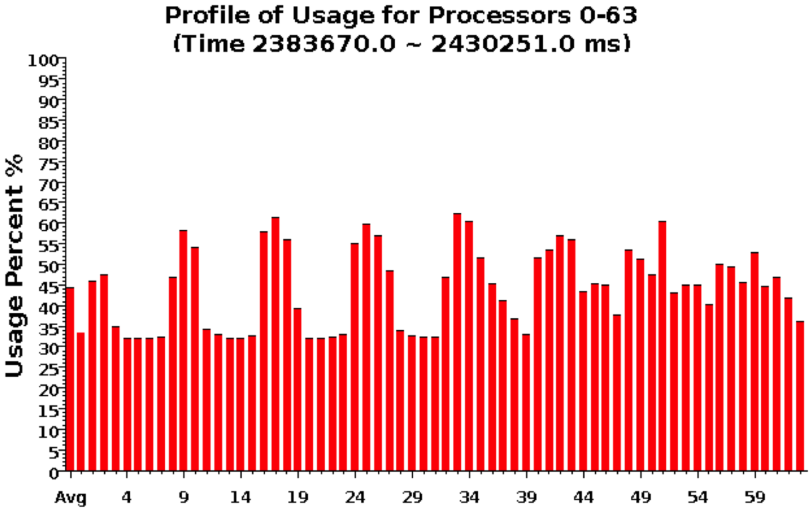


Basic Virtualization of BRAMS



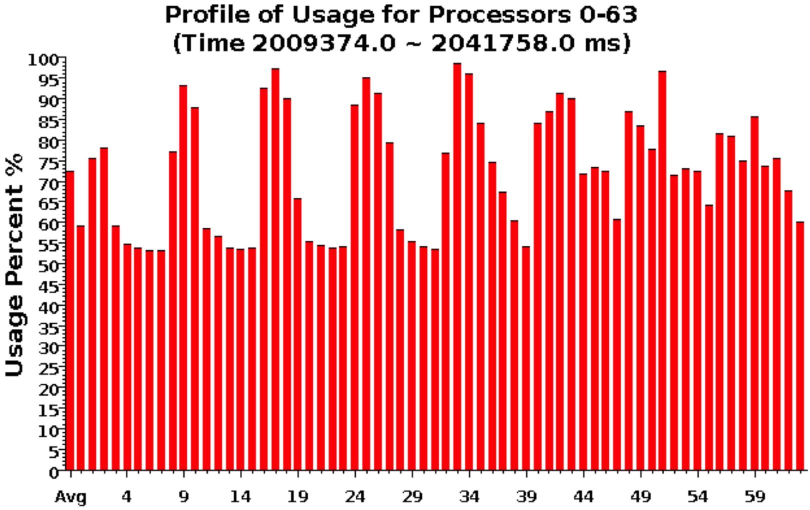


Baseline: 64 objects on 64 processors



Over-decomposition: 1024 objects on 64 processors

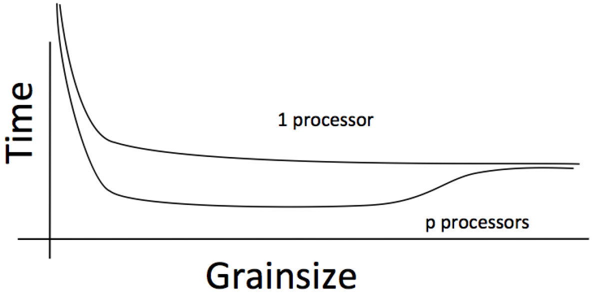
Benefits from communication/computation overlap



Grain Size

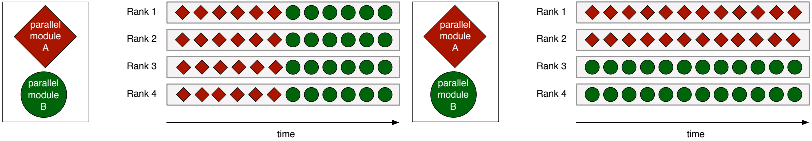
Working Definition

The amount of computation per potentially parallel event (task creation, enqueue/dequeue, messaging, locking, etc.)

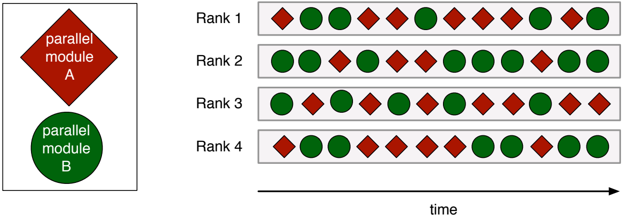


Modularity & Composability

- Easy to write code separately and then run it separately



- Possible to write code for explicit paralllel composition, interleaving multiple modules
- Want seamless resource sharing by separate pieces of code



Separation of Roles and Concerns

Different layers / components, different focus

- Application logic
- Parallel Algorithm
- Performance related application code
- Parallel runtime infrastructure

Separation of Roles and Concerns

Different layers / components, different focus

- Application logic
- Parallel Algorithm
- Performance related application code
- Parallel runtime infrastructure

Different expertise, different focus

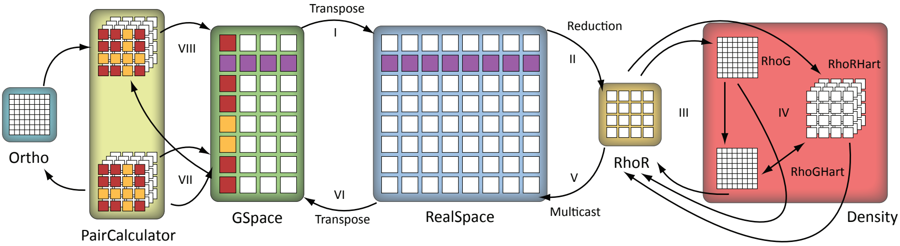
- Domain specialists write domain logic
- Performance experts specify tuning and optimizations
- HPC and CS experts develop and deploy runtime services

Different expertise, different focus: Object Mapping Code

```
/// Implement a mapping that tiles a 2D processor tile
/// in the 2D chare array
class LUMap : public CBase_LUMap {
  // . . .
  int procNum(int arrayHdl, const CkArrayIndex &idx) {
    const int *coor = idx.data();
    int tileYIndex = coor[1] / peCols;
    int XwithinPEtile = (coor[0] + tileYIndex * peRotate) % peRows;
    int YwithinPEtile = coor[1] % (peCols / peStride);
    int subtileY = (coor[1] % peCols) / (peCols / peStride);
    int peNum = XwithinPEtile * peStride +
                YwithinPEtile * peStride * peRows + subtileY;
    return peNum;
  }
};
```

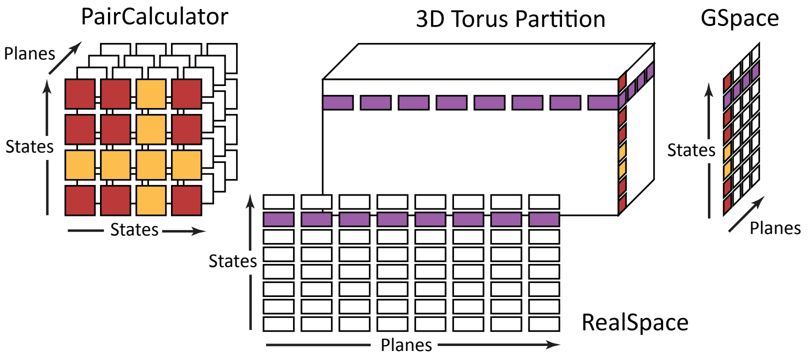
Different expertise, different focus

Mapping Example: Quantum Chemistry with OpenAtom



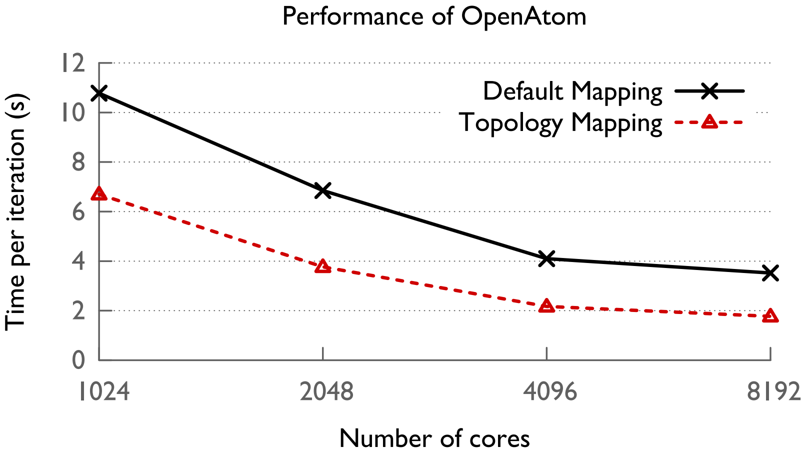
Different expertise, different focus

Mapping Example: Quantum Chemistry with OpenAtom



Different expertise, different focus

Mapping Example: Quantum Chemistry with OpenAtom



40% improvement, application only changed in initialization!

Separation of Concerns

Layered responsibility

Application worries about *what*, runtime system worries about *how*

- What data to send, vs. message allocation and packing
- Who to talk to, vs. where they live

Separation of Concerns

Example: Object location services

- Possible solutions to "Where does object X live?"
  - Name is location-specific
  - Object creator specifies location, passes along with name
  - Fixed mapping from names to locations
  - **Dynamic lookup**

Separation of Concerns

Example: Object location services

- Possible solutions to "Where does object X live?"
  - Name is location-specific
  - Object creator specifies location, passes along with name
  - Fixed mapping from names to locations
  - **Dynamic lookup**
- Charm++approach
  - *Mapping scheme* defines *home location* – default location, and responsible for knowing current location
  - Cache of last known locations on each processor
  - Messages sent to cached location, or home if none known

Separation of Concerns

Example: Object location services

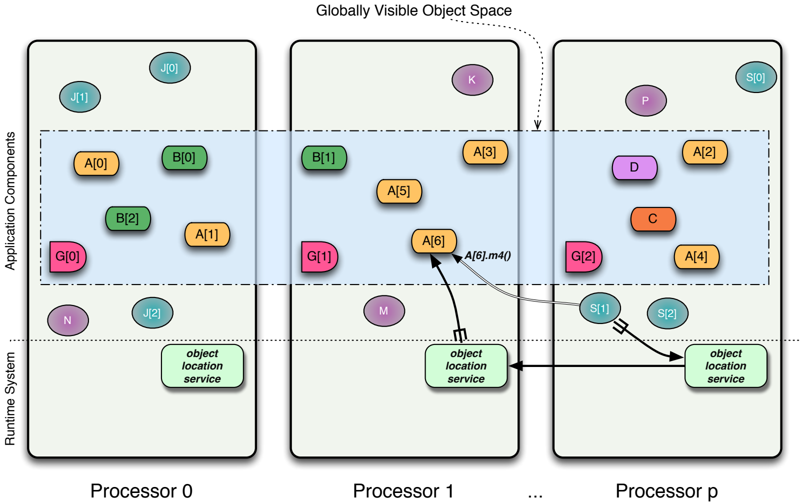
- Possible solutions to “Where does object X live?”
  - Name is location-specific
  - Object creator specifies location, passes along with name
  - Fixed mapping from names to locations
  - **Dynamic lookup**
- Charm++approach
  - *Mapping scheme* defines *home location* – default location, and responsible for knowing current location
  - Cache of last known locations on each processor
  - Messages sent to cached location, or home if none known

Application is mostly oblivioous

Fire off message, runtime delivers

Separation of Concerns

Example: Object location services



Separation of Concerns: Object Migration

Why migrate?

- Fault tolerance
- Communication locality
- Load balance
- Power, Energy, and Heat management

Separation of Concerns: Object Migration

Why migrate?

- Fault tolerance
- Communication locality
- Load balance
- Power, Energy, and Heat management

Application provides serialization routines, runtime can do the rest!

Object Serialization

```
class MyChare : public
      CBase_MyChare {
   int a; float b; char c;
   float localArray[LOCAL_SIZE];
   int heapArraySize;
   float* heapArray;
   MyClass *pointer;
   // . . .
};
```

```
void MyChare::pup(PUP::er &p) {
    CBase_MyChare::pup(p);
    p | a; p | b; p | c;
    p(localArray, LOCAL_SIZE);
    p | heapArraySize;
    if (p.isUnpacking()) {
       heapArray =
          new float[heapArraySize];
    }
    p(heapArray, heapArraySize);
    bool isNull = pointer==NULL;
    p | isNull;
    if (!isNull) {
       if (p.isUnpacking())
          pointer = new MyClass();
       p | *pointer;
    }
}
```

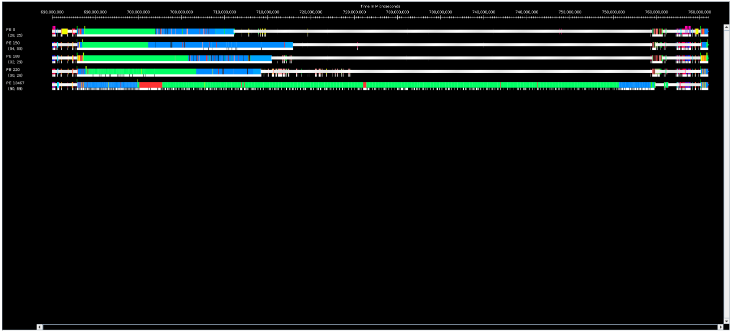
Introspective, Adaptive Runtime System

All about execution resources: processors, network, nodes, etc.

- Watch how each object and method uses resources: time running, bytes/messages sent & received, CPU frequency sensitivity, performance counters
- Record instrumented data for other components to use
- Invoke adaptation mechanisms at appropriate intervals
- Adjust system configuration accordingly

Load Imbalance

- Performance limited by difference between most-loaded processor and overall average.



Load Imbalance

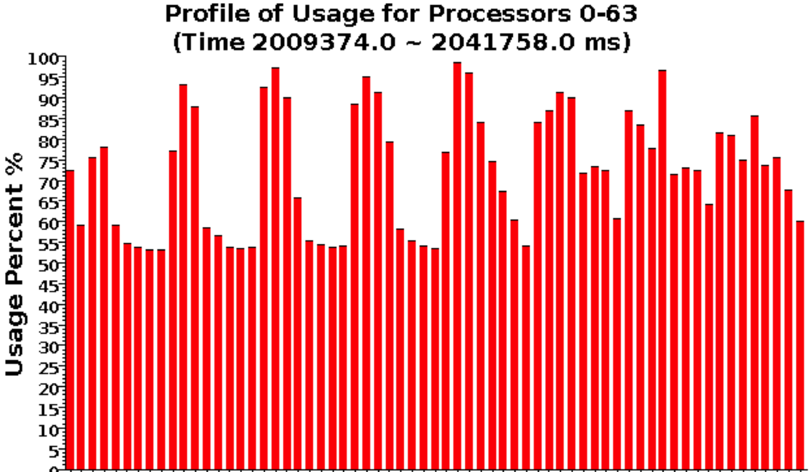
- Performance limited by difference between most-loaded processor and overall average.
- Causes vary in severity, time scale, nature

Load Imbalance

- Performance limited by difference between most-loaded processor and overall average.
- Causes vary in severity, time scale, nature
- Response must suit causes, other application concerns, system scale

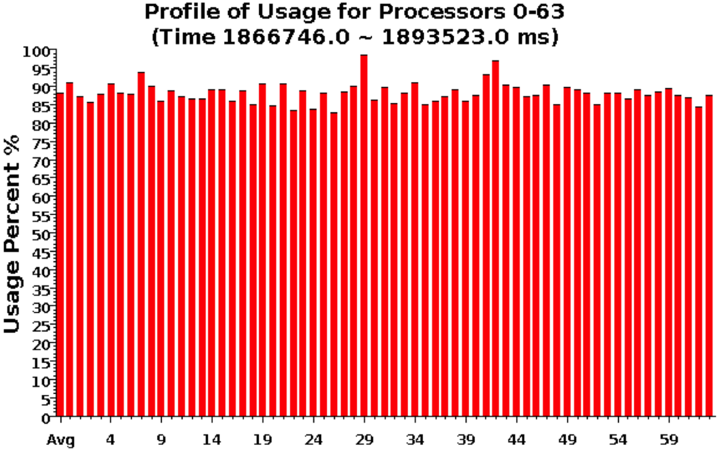
With Load Balancing: 1024 objects on 64 processors

- No overdecomp (64 threads): 4988 sec
- Overdecomp into 1024 threads: 3713 sec

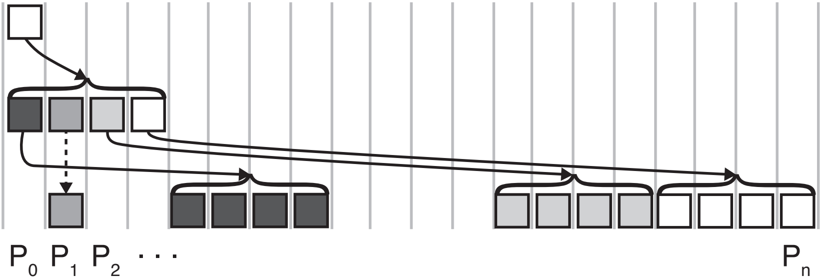


With Load Balancing: 1024 objects on 64 processors

- No overdecomp (64 threads): 4988 sec
- Overdecomp into 1024 threads: 3713 sec
- Load balancing (1024 threads): 3367 sec

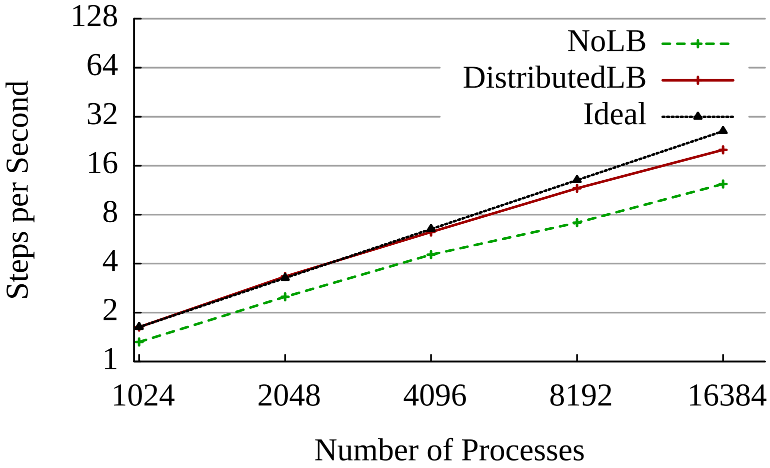


Load Balancing Adaptive Mesh Refinement for solving PDEs

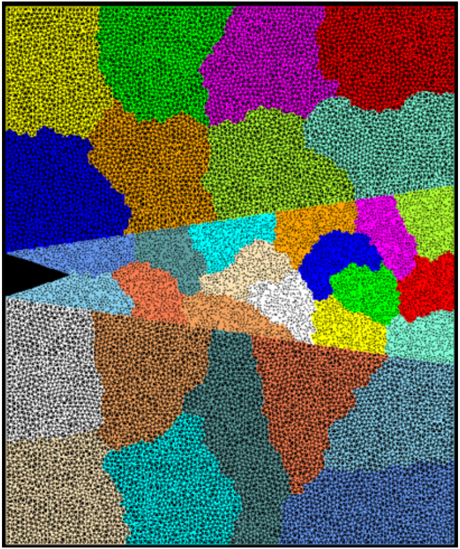


Load changes gradually and incrementally, suggesting localized strategies

Load Balancing Adaptive Mesh Refinement for solving PDEs



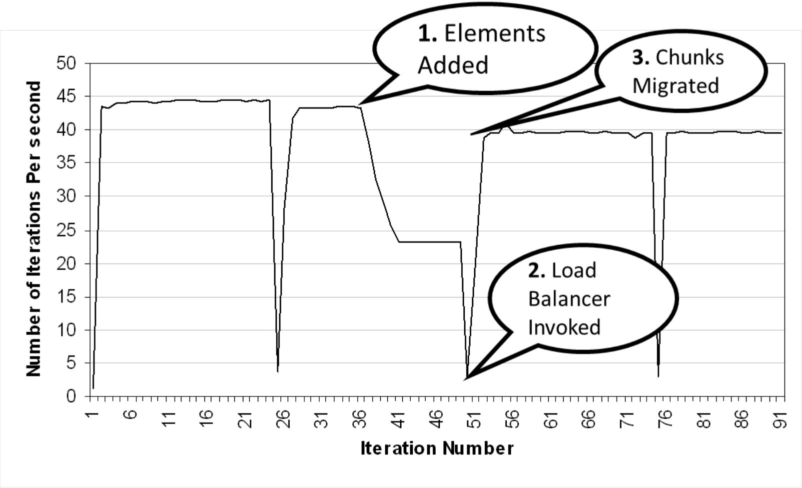
Load Imbalance: Crack Propagation



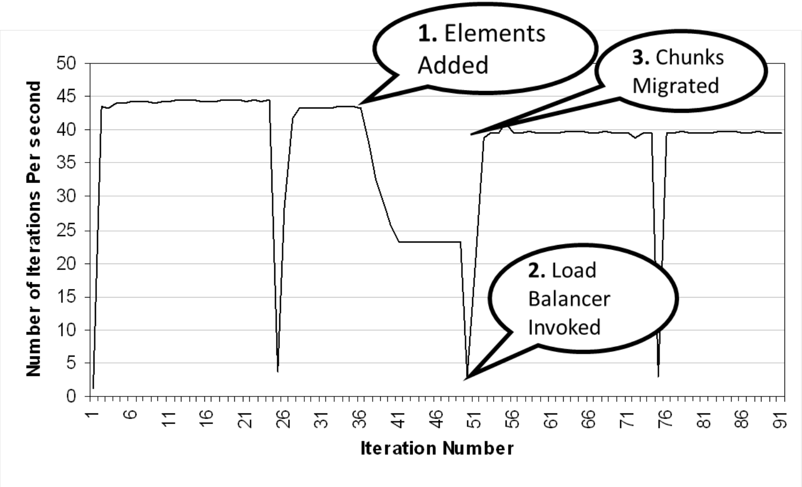
As computation progresses, crack propagates, and new elements are added, leading to more complex computations in some chunks

Picture: S. Breitenfeld and P. Geubelle

Load Imbalance: Crack Propagation



Load Imbalance: Crack Propagation



Sudden, severe shift in load suggests comprehensive rebalancing

Link-time: `-balancer GreedyLB` or `-balancer MetisLB`

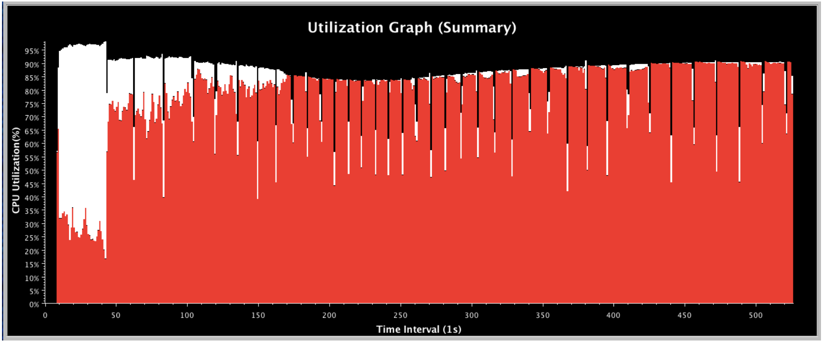
Run-time: `+balancer FooLB`

Load Imbalance: Adaptive Response

- When to run load balancer?

Load Imbalance: Adaptive Response

- When to run load balancer? When imbalance hurts (worse than the cost)!

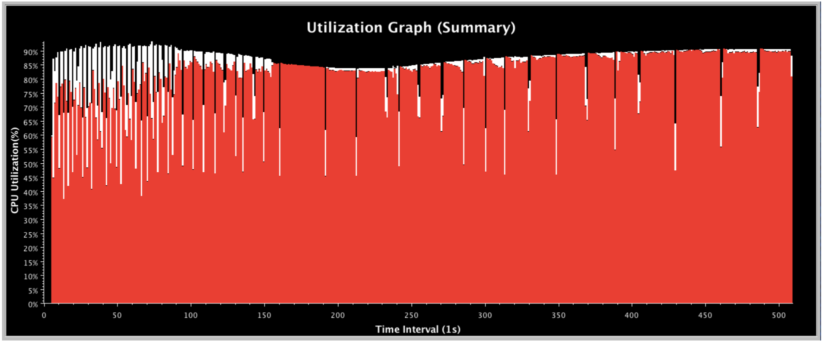


How to activate?

```
./pgm argsA argsB argsC +MetaLB
```

Load Imbalance: Adaptive Response

- When to run load balancer? When imbalance hurts (worse than the cost)!



How to activate?

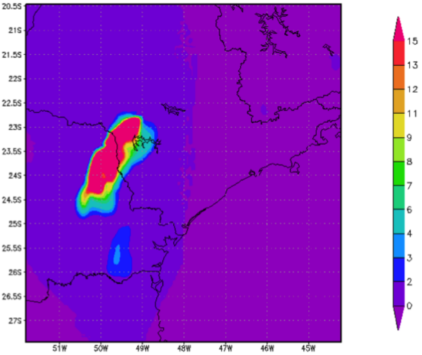
```
./pgm argsA argsB argsC +MetaLB
```

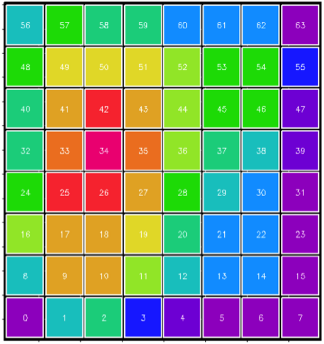
Load Imbalance: Adaptive Response

- When to run load balancer? When imbalance hurts (worse than the cost)!
- When to allow migration?

Load Imbalance: Adaptive Response

- When to run load balancer? When imbalance hurts (worse than the cost)!
- When to allow migration? When imbalance hurts (worse than the cost)!





Power, Energy, and Heat

Motivations

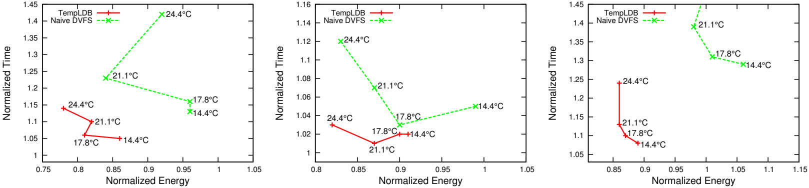
- Reduce direct costs of execution - cumulative machine energy, cooling energy from start to finish
- Reduce capital costs - transformers, chillers
- Improve reliability
- Improve user experience - fan noise, ambient heat, battery life

Power, Energy, and Heat

Established Technique

Set temperature threshold, periodic DVFS to enforce

- Slower clocks can hurt performance
- Load balance to compensate

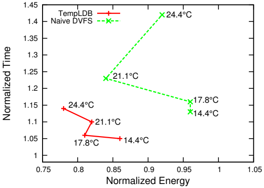


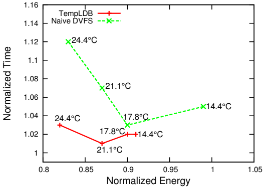
Power, Energy, and Heat

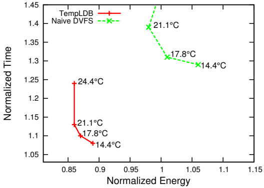
Established Technique

Set temperature threshold, periodic DVFS to enforce

- Slower clocks can hurt performance
- Load balance to compensate



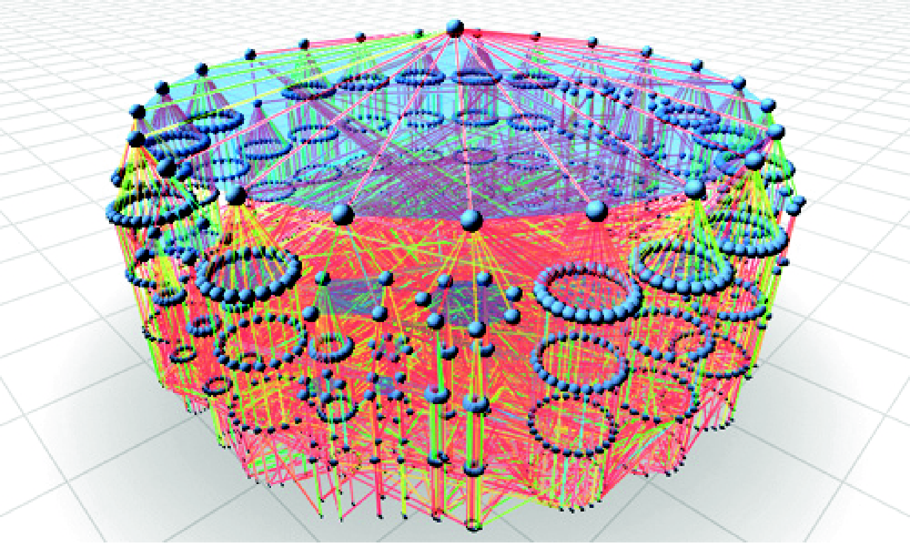




Upcoming Technique

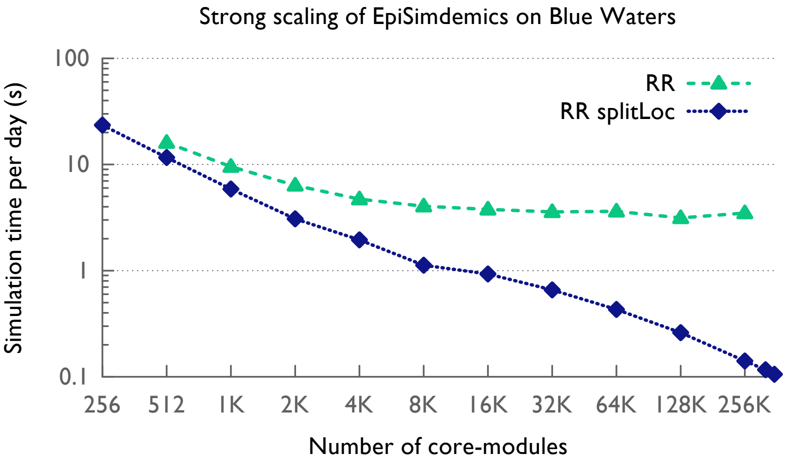
Set power threshold on newer Intel CPUs, load balance as overloads appear

Contagion and Information Spread: CharmEpiSimDemics



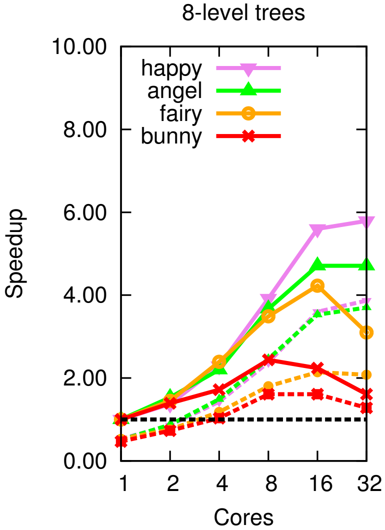
Contagion and Information Spread: CharmEpiSimDemics

Full US population simulations on Cray XE6 (Blue Waters)



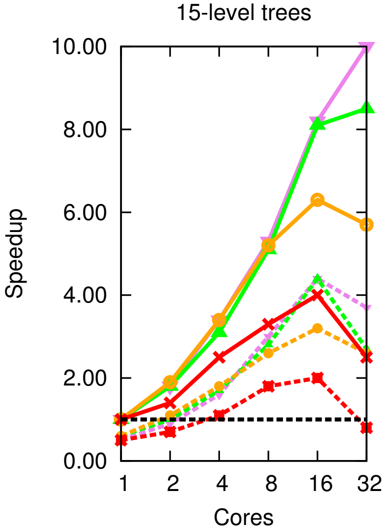
kd-tree construction on multicores

4 socket, 40 core intel xeon E7-4860 at 2.27GHz

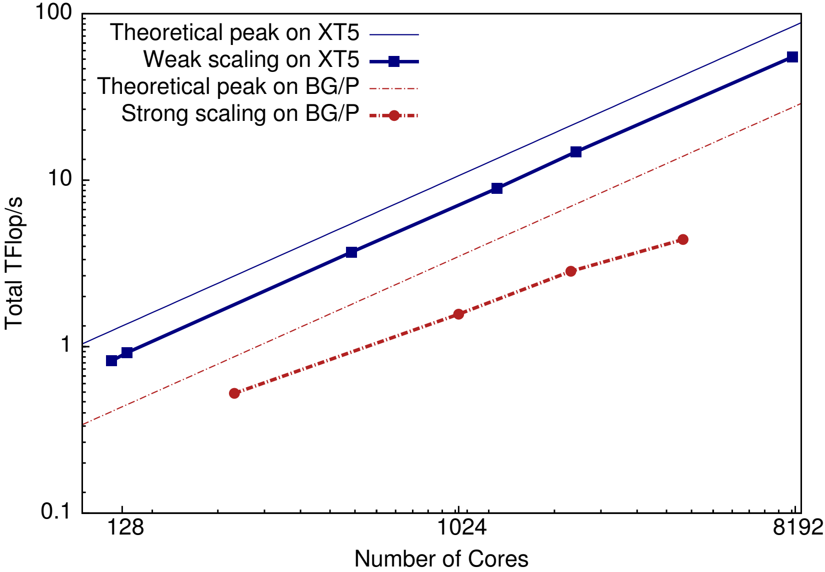


kd-tree construction on multicores

4 socket, 40 core intel xeon E7-4860 at 2.27GHz



Numerical Linear Algebra: Dense LU Factorization



Performance Analysis Using Projections

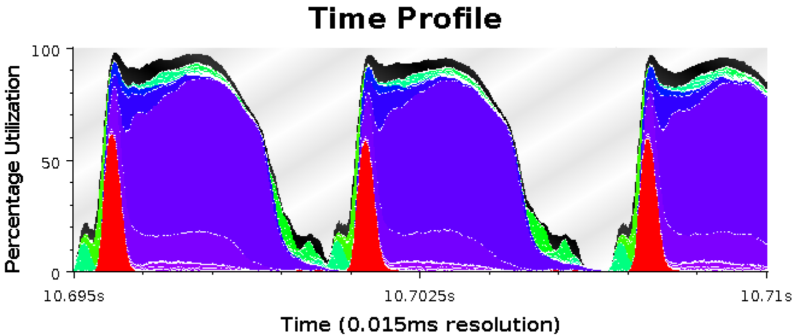
Instrumentation and measurement during program execution

- Easy setup: just modify link options
- Easy setup: data is generated automatically during run
- User events can be easily inserted as needed

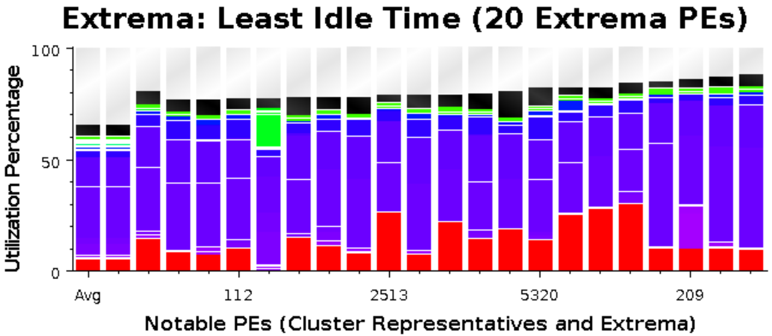
Visualization and analysis client

- Scalable: analyze execution traces for 100s of thousands of cores
- Rich feature set: time profile, time lines, usage profile, histograms, outliers etc
- Detect performance problems: load imbalance, grain size, communication bottleneck, etc

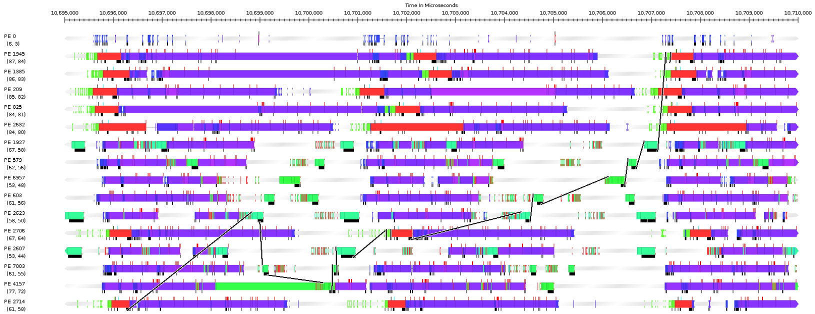
Time Profile



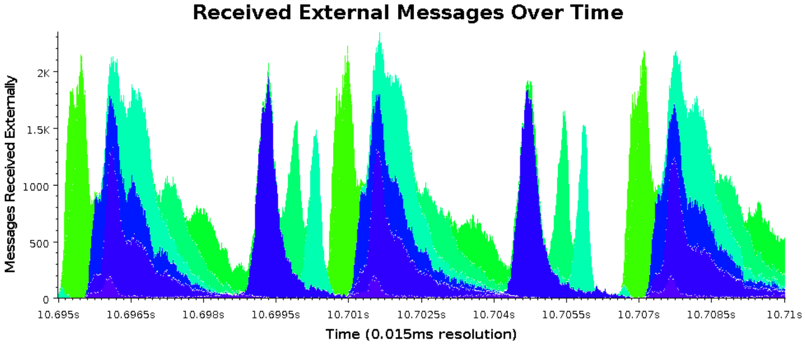
Extrema Tool for Least Idle Processors



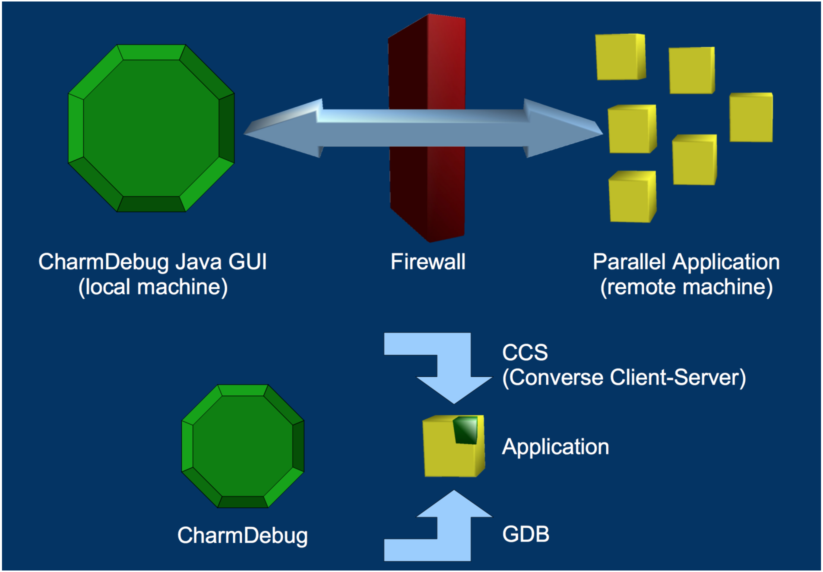
Time Lines with Message Back Tracing



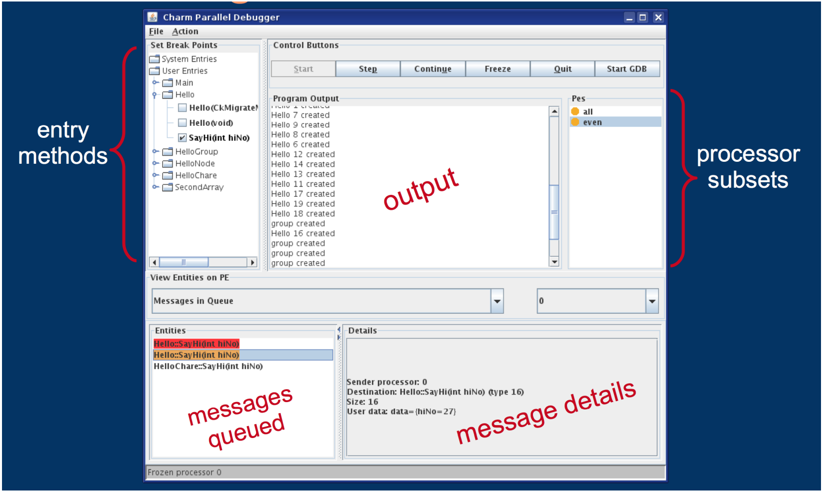
Communication over Time for all Processors



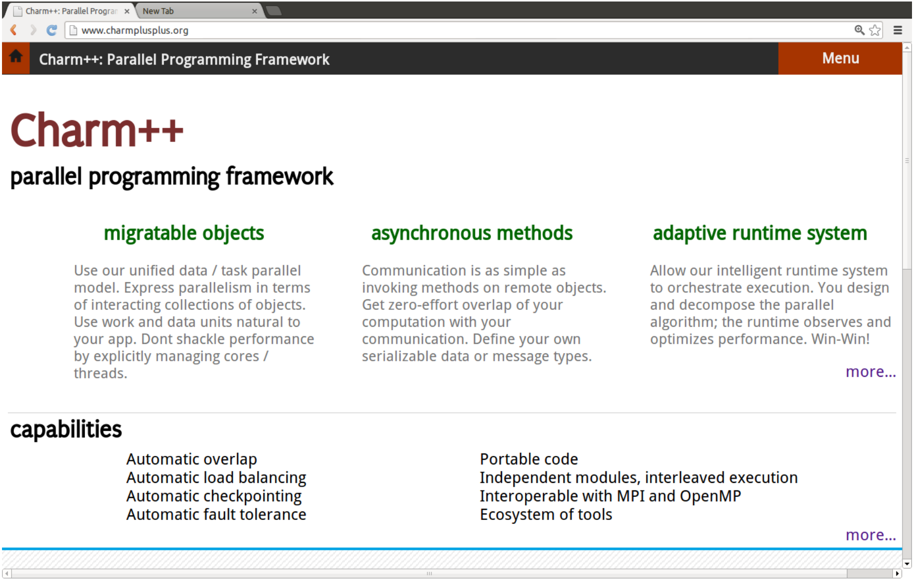
Debugging Charm++applications using CharmDebug



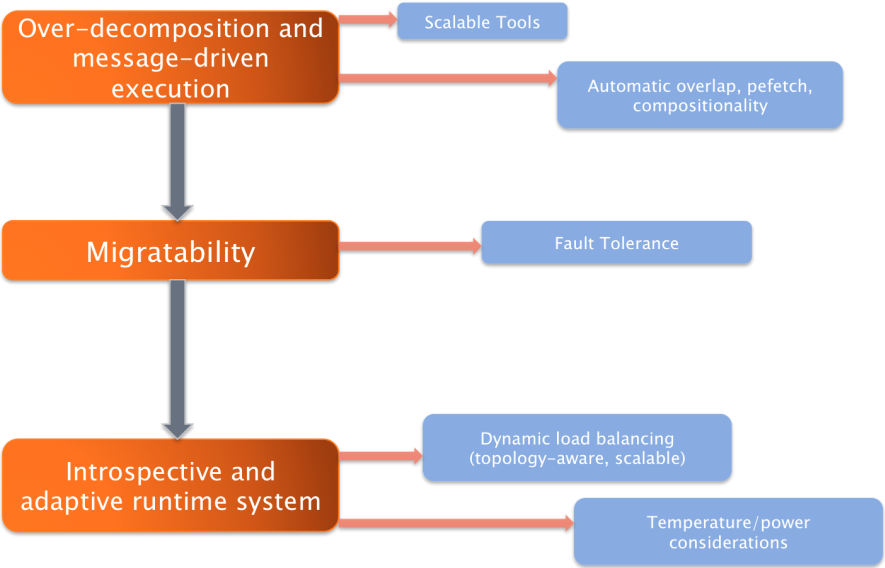
Debugging Charm++applications using CharmDebug



Recap


Charm++: Parallel Progran
New Tab
www.charmplusplus.org
Charm++: Parallel Programming Framework
Menu
Charm++
parallel programming framework
migratable objects
Use our unified data / task parallel model. Express parallelism in terms of interacting collections of objects. Use work and data units natural to your app. Dont shackle performance by explicitly managing cores / threads.
asynchronous methods
Communication is as simple as invoking methods on remote objects. Get zero-effort overlap of your computation with your communication. Define your own serializable data or message types.
adaptive runtime system
Allow our intelligent runtime system to orchestrate execution. You design and decompose the parallel algorithm; the runtime observes and optimizes performance. Win-Win!
more...
capabilities
Automatic overlap
Automatic load balancing
Automatic checkpointing
Automatic fault tolerance
Portable code
Independent modules, interleaved execution
Interoperable with MPI and OpenMP
Ecosystem of tools
more...

Questions?


Over-decomposition and message-driven execution
Scalable Tools
Automatic overlap, pefetch, compositionality
Migratability
Fault Tolerance
Introspective and adaptive runtime system
Dynamic load balancing (topology-aware, scalable)
Temperature/power considerations