Runtime Support for Multi-scale Applications on High-end Systems

U. Tennessee/Oak Ridge National Laboratory
Robert Harrison

Pacific Northwest National Laboratory
Jarek Nieplocha

Ohio State University
Atanas Rountev,
P. Sadayappan

Louisiana State University
Gerald Baumgartner,
J. Ramanujam

Supported by NSF and DOE

Motivation

- Programmer effort to develop complex high-performance applications is very high
- The majority of existing scalable parallel applications are written in MPI
  - Application programmers would prefer to use a parallel global address space framework IF they can get high performance
  - MPI will likely not be sufficient to achieve high performance on systems with significant multi-level parallelism (e.g. cluster of SMPs of 80-core CMPs)
- Can we provide runtime support that eases parallel programming and deliver high performance?
  - Focus on block-sparse computations
  - Motivating applications:
    - Tensor Contraction Engine – Domain-specific compiler for a class of ab-initio quantum chemistry models
    - MADNESS (Multiresolution ADaptive NumErical Scientific Simulation) – new quantum chemistry code from Robert Harrison

Tensor Contraction Engine

- Automatic transformation from high-level specification
  - Chemist specifies computation in high-level mathematical form
  - Synthesis system transforms it to efficient parallel program
  - Code is tailored to target machine
  - Code can be optimized for specific molecules being modeled
- Multi-institutional collaboration (OSU, LSU, Waterloo, ORNL, PNNL, U. Florida)
- Two prototypes of TCE are operational
  - a) Full exploitation of symmetry, but fewer optimizations, b) Dense tensors, but more sophisticated optimizations; Integrated version with all optimizations and exploitation of sparsity/symmetry is nearing completion
  - Used to implement over 20 models, included in latest release of NWChem
  - First parallel implementation for many of the methods
  - TCE Workshop at Sanibel 2007 meeting of quantum chemists generated much interest

$$A3A = \frac{1}{2}(X_{ce,af} Y_{ae,cf} + X_{\bar{c}\bar{e},\bar{a}\bar{f}} Y_{\bar{a}\bar{e},\bar{c}\bar{f}} + X_{c\bar{e},a\bar{f}} Y_{a\bar{e},c\bar{f}} + X_{\bar{c}e,\bar{a}f} Y_{\bar{a}e,\bar{c}f} + X_{\bar{c}\bar{e},\bar{a}\bar{f}} Y_{\bar{a}\bar{e},\bar{c}\bar{f}} + X_{\bar{c}e,\bar{a}\bar{f}} Y_{\bar{a}e,\bar{c}\bar{f}} + X_{\bar{c}\bar{e},\bar{a}f} Y_{\bar{a}\bar{e},\bar{c}f} + X_{\bar{c}\bar{e},\bar{a}\bar{f}} Y_{\bar{a}\bar{e},\bar{c}\bar{f}})$$

$$X_{ce,af} = t_{ij}^{ce} t_{ij}^{af} \qquad Y_{ae,cf} = \langle ab \| ek \rangle \langle cb \| fk \rangle$$

```
range V = 3000;
range O = 100;

index a,b,c,d,e,f : V;
index i,j,k : O;

mlimit = 10000000;

function F1(V,V,V,O);
function F2(V,V,V,O);

procedure P(in T1[O,O,V,V], in T2[O,O,V,V], out X)=

begin
   A3A == sum[ sum[F1(a,b,e,k) * F2(c,f,b,k), {b,k}]
                  * sum[T1[i,j,c,e] * T2[i,j,a,f], {i,j}],
                  {a,e,c,f}]*0.5 + ...;
end
```

CCSD Doubles Equation

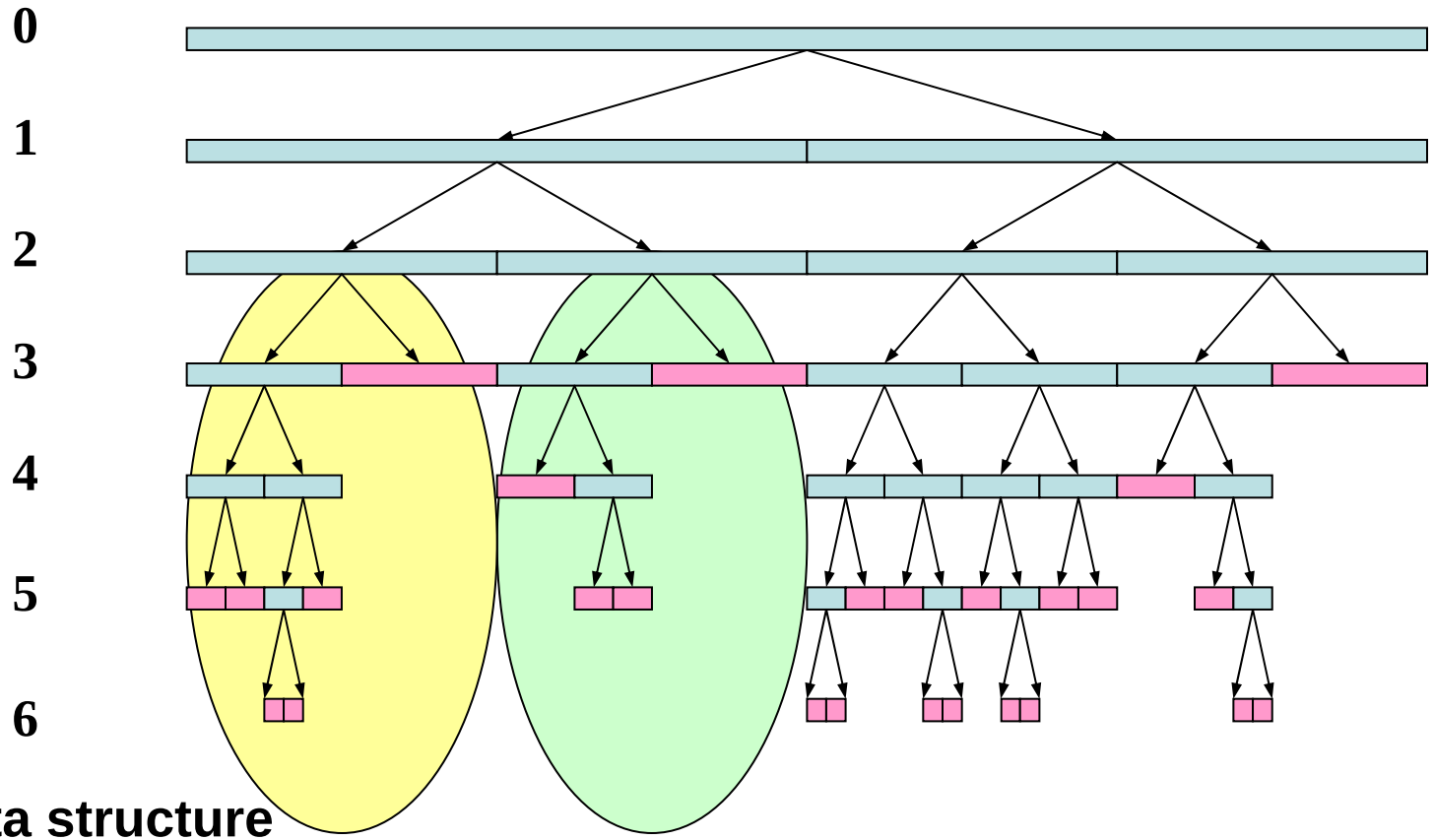
hbar[a,b,i,j] == sum[f[b,c]*t[i,j,a,c],{c}] -sum[f[k,c]*t[k,b]*t[i,j,a,c],{k,c}] +sum[f[a,c]*t[i,j,c,b],{c}] -sum[f[k,c]*t[k,a]*t[i,j,c,b],{k,c}] -sum[f[k,j]*t[i,k,a,b],{k}] -sum[f[k,c]*t[j,c]*t[i,k,a,b],{k,c}] -sum[f[k,i]*t[j,k,b,a],{k}] -sum[f[k,c]*t[i,c]*t[j,k,b,a],{k,c}] +sum[t[i,c]*t[j,d]*v[a,b,c,d],{c,d}] +sum[t[i,j,c,d]*v[a,b,c,d],{c,d}] +sum[t[j,c]*v[a,b,i,c],{c}] -sum[t[k,b]*v[a,k,i,j],{k}] +sum[t[i,c]*v[b,a,j,c],{c}] -sum[t[k,a]*v[b,k,j,i],{k}] -sum[t[k,d]*t[i,j,c,b]*v[k,a,c,d],{k,c,d}] -sum[t[i,c]*t[j,k,b,d]*v[k,a,c,d],{k,c,d}] -sum[t[j,c]*t[k,b]*v[k,a,c,i],{k,c}] +2*sum[t[j,k,b,c]*v[k,a,c,i],{k,c}] -sum[t[j,k,c,b]*v[k,a,c,i],{k,c}] -sum[t[i,c]*t[j,d]*t[k,b]*v[k,a,d,c],{k,c,d}] +2*sum[t[k,d]*t[i,j,c,b]*v[k,a,d,c],{k,c,d}] -sum[t[k,b]*t[i,j,c,d]*v[k,a,d,c],{k,c,d}] -sum[t[j,d]*t[i,k,c,b]*v[k,a,d,c],{k,c,d}] +2*sum[t[i,c]*t[j,k,b,d]*v[k,a,d,c],{k,c,d}] -sum[t[i,c]*t[j,k,d,b]*v[k,a,d,c],{k,c,d}] -sum[t[j,k,b,c]*v[k,a,i,c],{k,c}] -sum[t[i,c]*t[k,b]*v[k,a,j,c],{k,c}] -sum[t[i,k,c,b]*v[k,a,j,c],{k,c}] -sum[t[i,c]*t[j,d]*t[k,a]*v[k,b,c,d],{k,c,d}] -sum[t[k,d]*t[i,j,a,c]*v[k,b,c,d],{k,c,d}] -sum[t[k,a]*t[i,j,c,d]*v[k,b,c,d],{k,c,d}] +2*sum[t[j,d]*t[i,k,a,c]*v[k,b,c,d],{k,c,d}] -sum[t[j,d]*t[i,k,c,a]*v[k,b,c,d],{k,c,d}] -sum[t[i,c]*t[j,k,d,a]*v[k,b,c,d],{k,c,d}] -sum[t[i,c]*t[k,a]*v[k,b,c,j],{k,c}] +2*sum[t[i,k,a,c]*v[k,b,c,j],{k,c}] -sum[t[i,k,c,a]*v[k,b,c,j],{k,c}] +2*sum[t[k,d]*t[i,j,a,c]*v[k,b,d,c],{k,c,d}] -sum[t[j,d]*t[i,k,a,c]*v[k,b,d,c],{k,c,d}] -sum[t[j,c]*t[k,a]*v[k,b,i,c],{k,c}] -sum[t[j,k,c,a]*v[k,b,i,c],{k,c}] -sum[t[i,k,a,c]*v[k,b,j,c],{k,c}] +sum[t[i,c]*t[j,d]*t[k,a]*t[l,b]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[k,b]*t[l,d]*t[i,j,a,c]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[k,a]*t[l,d]*t[i,j,c,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[k,a]*t[l,b]*t[i,j,c,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[j,c]*t[l,d]*t[i,k,a,b]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[j,d]*t[l,b]*t[i,k,a,c]*v[k,l,c,d],{k,l,c,d}] +sum[t[j,d]*t[l,b]*t[i,k,c,a]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,c]*t[l,d]*t[j,k,b,a]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,a]*t[j,k,b,d]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,b]*t[j,k,d,a]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,k,c,d]*t[j,l,b,a]*v[k,l,c,d],{k,l,c,d}] +4*sum[t[i,k,a,c]*t[j,l,b,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,k,c,a]*t[j,l,b,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,k,a,b]*t[j,l,c,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,k,a,c]*t[j,l,d,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,k,c,a]*t[j,l,d,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,c]*t[j,d]*t[k,l,a,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,j,c,d]*t[k,l,a,b]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,j,c,b]*t[k,l,a,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,j,a,c]*t[k,l,b,d]*v[k,l,c,d],{k,l,c,d}] +sum[t[j,c]*t[k,b]*t[l,a]*v[k,l,c,i],{k,l,c}] +sum[t[l,c]*t[j,k,b,a]*v[k,l,c,i],{k,l,c}] -2*sum[t[l,a]*t[j,k,b,c]*v[k,l,c,i],{k,l,c}] +sum[t[l,a]*t[j,k,c,b]*v[k,l,c,i],{k,l,c}] -2*sum[t[k,c]*t[j,l,b,a]*v[k,l,c,i],{k,l,c}] +sum[t[k,a]*t[j,l,b,c]*v[k,l,c,i],{k,l,c}] +sum[t[k,b]*t[j,l,c,a]*v[k,l,c,i],{k,l,c}] +sum[t[j,c]*t[l,k,a,b]*v[k,l,c,i],{k,l,c}] +sum[t[i,c]*t[k,a]*t[l,b]*v[k,l,c,j],{k,l,c}] +sum[t[l,c]*t[i,k,a,b]*v[k,l,c,j],{k,l,c}] -2*sum[t[l,b]*t[i,k,a,c]*v[k,l,c,j],{k,l,c}] +sum[t[l,b]*t[i,k,c,a]*v[k,l,c,j],{k,l,c}] +sum[t[i,c]*t[k,l,a,b]*v[k,l,c,j],{k,l,c}] +sum[t[j,c]*t[l,d]*t[i,k,a,b]*v[k,l,d,c],{k,l,c,d}] +sum[t[j,d]*t[l,b]*t[i,k,a,c]*v[k,l,d,c],{k,l,c,d}] +sum[t[j,d]*t[l,a]*t[i,k,c,b]*v[k,l,d,c],{k,l,c,d}] -2*sum[t[i,k,c,d]*t[j,l,b,a]*v[k,l,d,c],{k,l,c,d}] -2*sum[t[i,k,a,c]*t[j,l,b,d]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,c,a]*t[j,l,b,d]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,a,b]*t[j,l,c,d]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,c,b]*t[j,l,d,a]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,a,c]*t[j,l,d,b]*v[k,l,d,c],{k,l,c,d}] +sum[t[k,a]*t[l,b]*v[k,l,i,j],{k,l}] +sum[t[k,l,a,b]*v[k,l,i,j],{k,l}] +sum[t[k,b]*t[l,d]*t[i,j,a,c]*v[l,k,c,d],{k,l,c,d}] +sum[t[k,a]*t[l,d]*t[i,j,c,b]*v[l,k,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,d]*t[j,k,b,a]*v[l,k,c,d],{k,l,c,d}] -2*sum[t[i,c]*t[l,a]*t[j,k,b,d]*v[l,k,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,a]*t[j,k,d,b]*v[l,k,c,d],{k,l,c,d}] +sum[t[i,j,c,b]*t[k,l,a,d]*v[l,k,c,d],{k,l,c,d}] +sum[t[i,j,a,c]*t[k,l,b,d]*v[l,k,c,d],{k,l,c,d}] -2*sum[t[l,c]*t[i,k,a,b]*v[l,k,c,j],{k,l,c}] +sum[t[l,b]*t[i,k,a,c]*v[l,k,c,j],{k,l,c}] +sum[t[l,a]*t[i,k,c,b]*v[l,k,c,j],{k,l,c}] +v[a,b,i,j]

MADNESS

Multiresolution
Adaptive
Numerical
Scientific
Simulation

New quantum chemistry code from Robert Harrison

Sub-Tree Parallelism in MADNESS



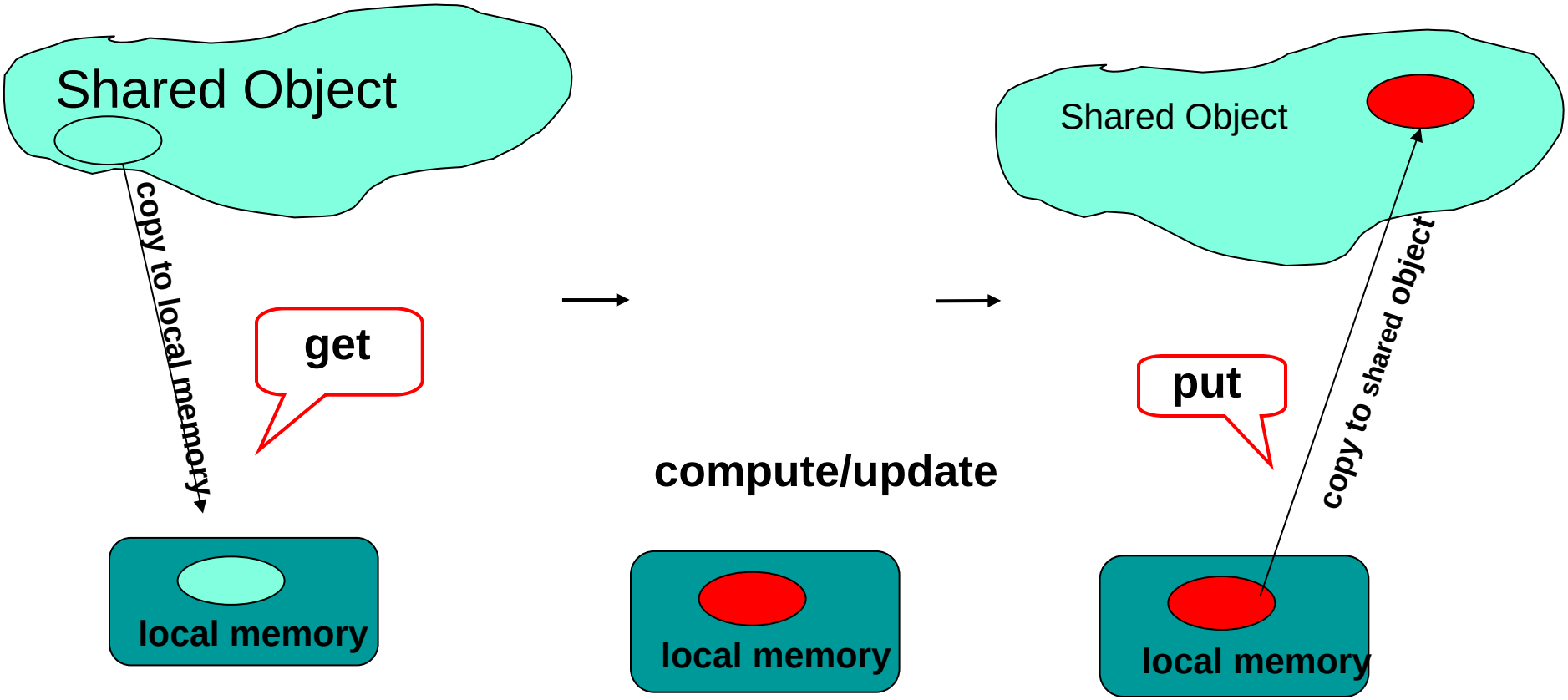
Data structure

- Tree representation of spatial functions, adaptively refined for necessary precision
- Distributed in a global address space

Operations

- Algebraic operations, integration, differentiation, etc.
- Tree traversals requiring parent-child and neighbor relationships
- Dynamically created and destroyed

Global Arrays Model of Computations



- Shared memory view for distributed dense arrays
- MPI-Compatible; Currently usable with Fortran, C, C++, Python
- Data locality and granularity control similar to message passing model
- Used in large scale efforts, e.g. NWChem (million+ lines/code)

Extensions to the GA Model

DATA: Beyond dense multi-dimensional arrays

- More complex data-structures: block sparse arrays; tree structures
- Globally addressable, but locality aware

COMPUTATION: Beyond static process-centric parallelism

- Some applications are easier to program with a task based message-driven model
- Support for load-balancing

ARCHITECTURE: Simplify programming explicitly managed memory hierarchies.

- Automatically schedule data movement
- Locality-aware load-balanced computation scheduling
- Transparent Memory Hierarchy Management
  - Non-collective I/O on local disks
  - Shared memory-style programming across distinct adress spaces, but locality-managed for performance

Non-collective I/O on local disks

- Data distributed on local disks of nodes
  - Scalable with number of nodes
- Control over data distribution
  - Better exploitation of data locality
- Any processor can non-collectively access any data on disk
  - Global address space for data on disk
  - Simplifies out-of-core programming
- Global Procedure Calls (GPC)
  - Mechanism to invoke a procedure in a remote processor
  - Provides portable active messages support in GA suite
- Non-collective I/O implemented by leveraging GPC

Transparent Memory Hierarchy Management for Independent Tasks

- Problem: Schedule computation and disk I/O operations
- Objective: Minimize disk I/O
- Constraint: Available physical memory
- Solution: Hypergraph partitioning formulation
  - Efficient solutions to the hypergraph problem exist
  - Typically used in the context of parallelization
    - Number of parts known
  - No constraints such as memory limit
    - Only balancing constraint

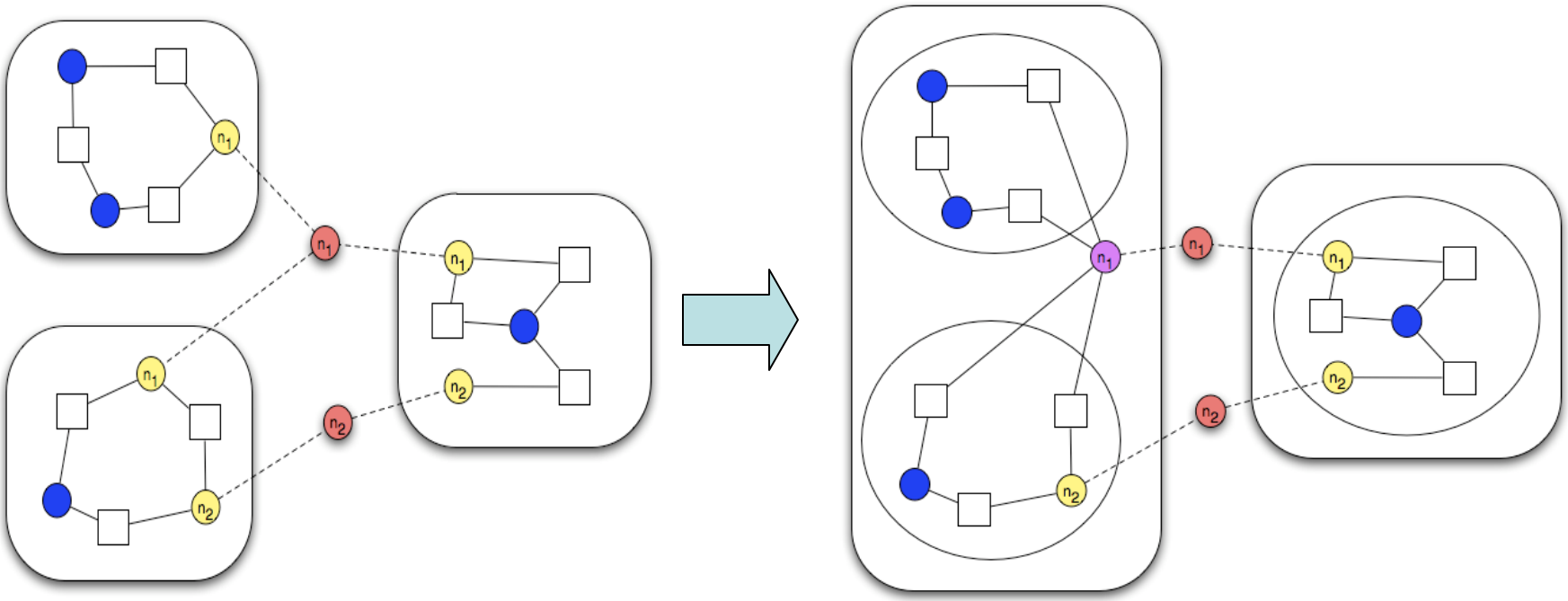
Hypergraph Formulation for Memory Management

- Formulation
  - Tasks -> vertices
  - Data -> Nets
  - No pre-assignment of nets to certain parts
  - Balance: Memory usage in the parts
    - Guarantees solution for some #parts, if it exists
  - Determine dynamically #parts
    - Modify the inherent recursive procedure of hypergraph partitioning.

Read-Once Partitioning

- Direct solution to above problem
  - Similar to approaches to parallelization
  - No refined reuse relationships
    - All tasks within a part have reuse, and none outside
- Read-Once Partitioning
  - Group tasks into steps
  - Identify data common across steps and load into memory
  - For each step, read non-common (step-exclusive) data, process tasks, and write/discard step-exclusive data
  - Better utilization of memory available -> reduced disk I/O

Read-Once Partitioning: Example



Disk I/O: 9 data elements

Disk I/O: 8 data elements

Summary

- Application-motivated approach to developing runtime support for scalable parallelism with global-address-space programming models
- Developing data and task abstractions that ease programming, but also achieve high performance
- Targeting systems with explicitly managed on-chip memory: Cell, GPGPU