An Execution Model for Irregular Applications

Arun Chauhan (Rice University)

Kathleen Knobe (Compaq Computer Corp)

Workshop on

Compilers for Parallel Computation - 2000

January 4-7, Aussois, France

Motivation

- Parallelization is hard
  - It is even harder for irregular applications
- Most popular current solutions are inadequate
  - message passing: efficient, but hard
  - shared memory: easy, but could be inefficient
- We need an intermediate solution

What do we want to do?

Automatic Parallelization from
a High-Level description

For irregular applications

On heterogeneous environments

High-Level Approach

Develop Compiler Technology to recognize and handle irregular applications

Develop run-time support system

Increasingly, the distinction is fading. We could view the run-time system as dynamic compilation, in some cases.

Key Ideas

Decompose data domain

$\Longrightarrow$ data items

Partition computation

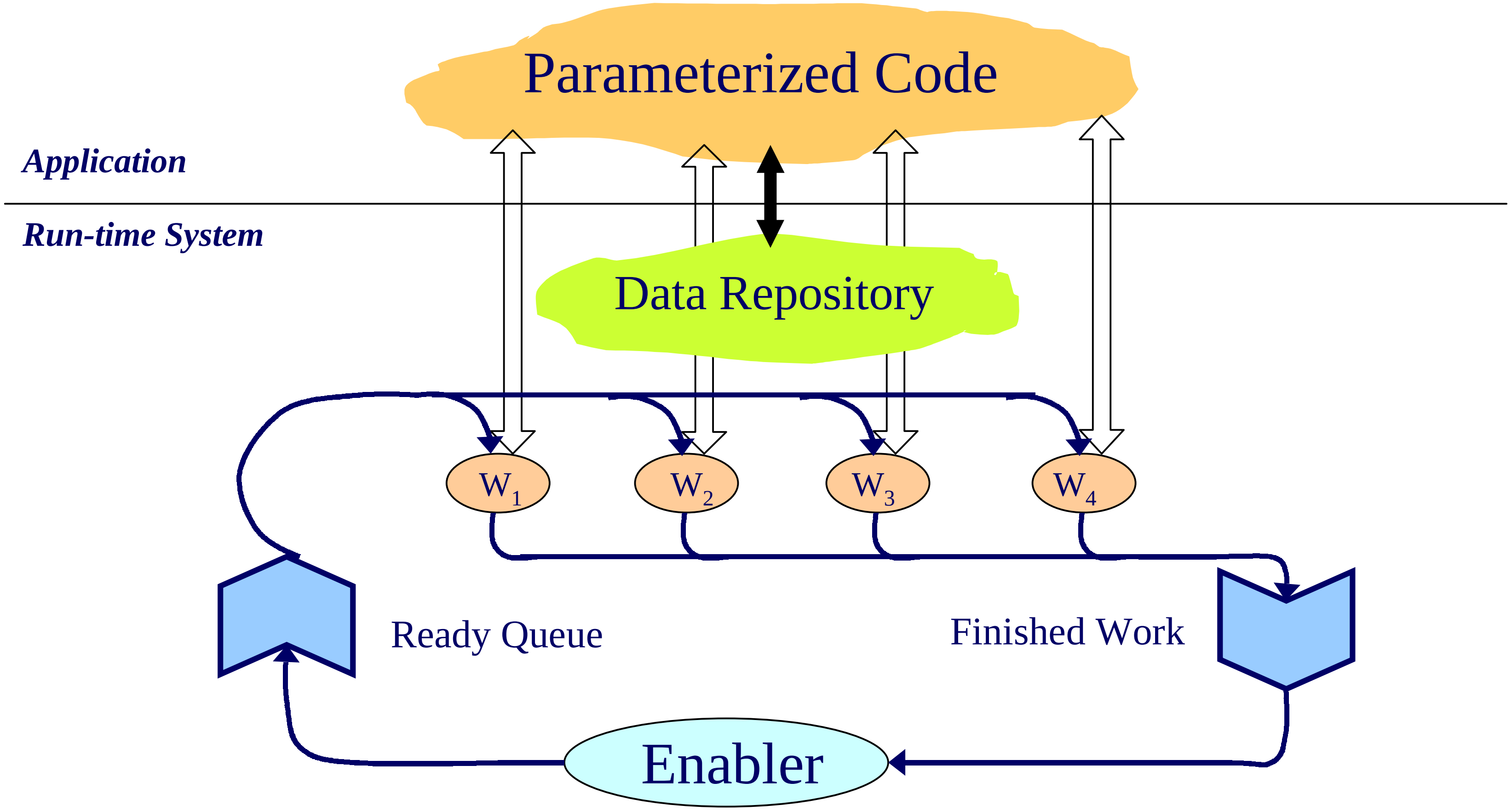
$\Longrightarrow$ work-orders

Work Queue based execution

"Data Repository" for shared data

global naming scheme for data items

Overall Architecture



Key Characteristics

- Load balancing through workers
- Tuple-based global naming for blocked data
- Read-only data
  - avoids coherency problem
  - eliminates all but true dependencies
- Reference count based garbage collection

Cholesky Factorization

The problem:

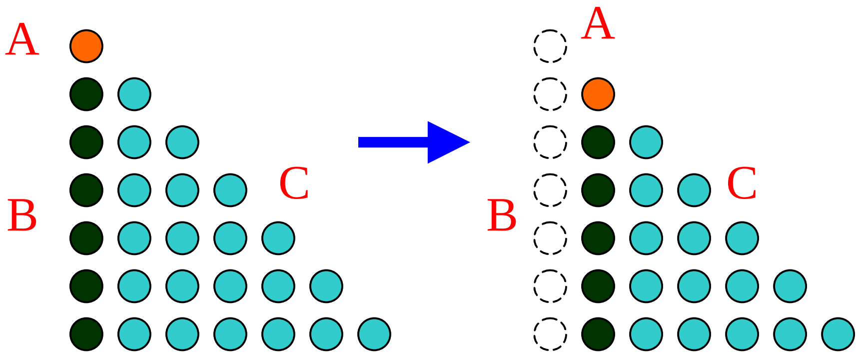
Given symmetric positive definite matrix, M, compute L such that $L.L^T = M$.

Sequential algorithm:

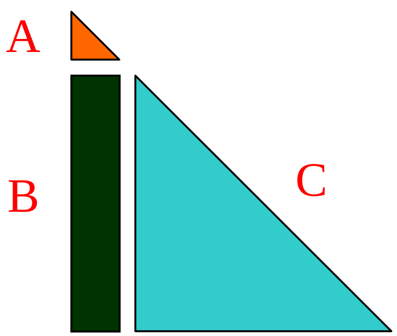
$A = \sqrt{A}$

$B = B / A$

$C = C - B.B^T$



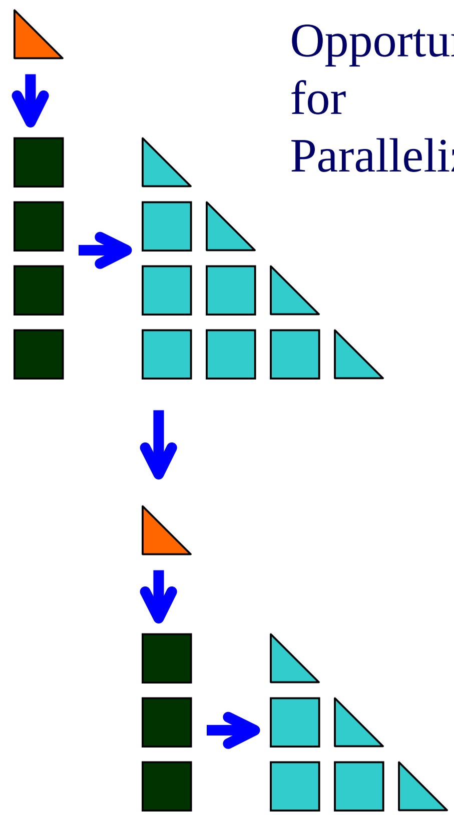
Blocked Cholesky



$A = \text{Cholesky}(A)$

$B = B \times A^{-1}$

$C = C - B \times B^{T}$



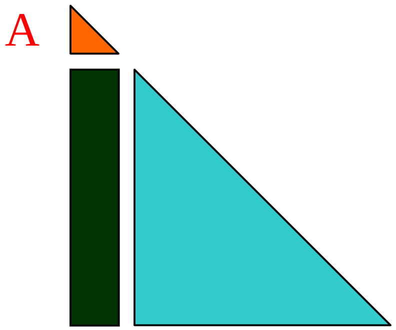
Opportunities for Parallelization

Parallelilizing Cholesky

- register application dependent information
  - tuple size, iteration vector size, etc.
- register three types of code:
  - input thread: initial data and work orders
  - output thread: gather final results and display
  - executors: various computations
- run-time system executes a virtual data-flow computation

Parallel Cholesky

```
Executor A:
input: work order WO
{
  read_inputs (WO, matrix A);
  B = Cholesky(A);
  let i = row & col number of block A;
  let d = data-item ID for B;
  for r = i+1, NUM_BLOCKS do
        let w = work-order for block i;
        insert work-order (w, d, 2);
  endfor
  write data-item (B, d, NUM_BLOCKS-i);
}
```

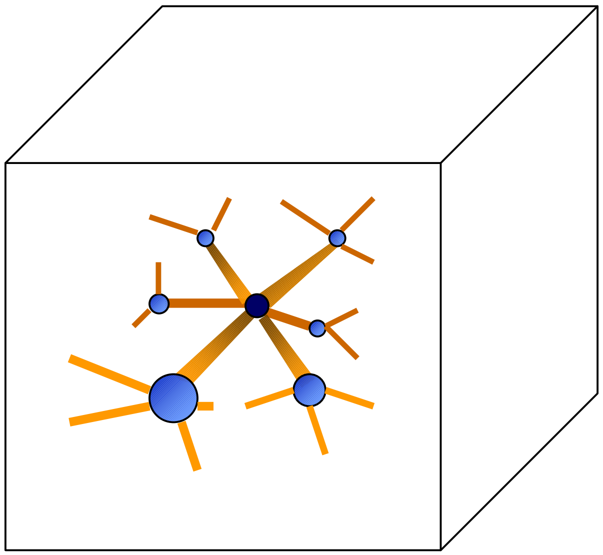


Molecular Dynamics

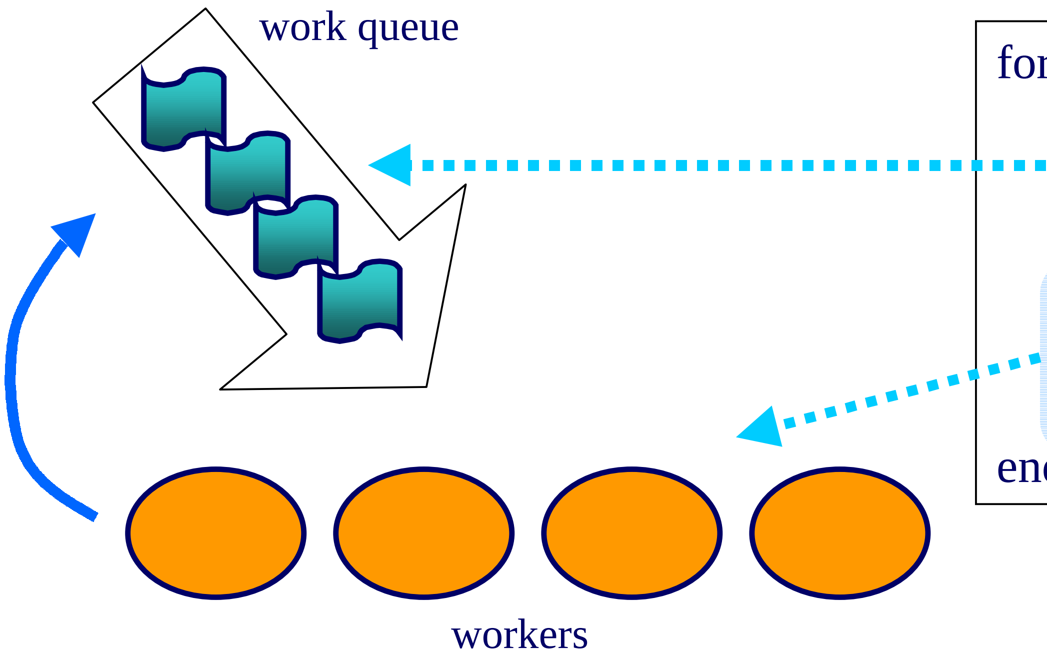
The problem:

Given N bodies in a bounded box, compute their evolution in time based on mutual interactions.

```
for t = 1, MAX_TIME_STEPS do
    if (mod(t,K) == 0)
        re_compute_neighbors;
    endif
    compute_mutual_forces;
    update_particle_attributes;
    compute_system_KE;
endfor
```



Parallel Molecular Dynamics




```
for t = 1, MAX_TIME_STEPS do
    if (mod(t,K) == 0)
        re_compute_neighbors;
    endif
    compute_mutual_forces;
    update_particle_attributes;
    compute_system_KE;
endfor
```

interactions ⇒ work orders
computations ⇒ executors

Current Status

- Two applications validate our system
- Performance tuning in progress
  - application level
  - system level
- Porting other applications to the model
  - e.g., hierarchical n-body
- Refining the model in the process

Related Work

- LINDA, from Yale
  - tuple spaces similar; but different focus
- SMARTS, from LANL
  - iteration level scheduling
  - no mechanism for remote data-naming
- CHAOS, from UMCP
  - inspector-executor model

Future Directions

- Hierarchical design
  - scalability
  - locality
- Heterogeneous environments
- Locality awareness
- Compiler technology

Conclusion

- An Execution Model for irregular apps
  - dynamic load balancing
  - scalable in space usage
  - avoids coherency & dependency problems
  - fine granularity minimizes false sharing
- Past experience shows promise
- Stay tuned:
  - **http://www.cs.rice.edu/~achauhan/**