Runtime Parallelization


Outline

• Overview
• Compiler-driven: Multiblock Party
• Library-driven: KeLP
• Conclusion

Overview

• Writing good parallel programs for distributed memory systems is hard.
• Idea: abstraction on top of message passing to get results
• We can do this where communication is regular: block-structured applications
• Trade off: reduced performance for reduced effort

Notes

• Midterm exam next Thursday, Nov. 17
• Group project interim reports due Monday, Nov. 21
Multiblock Parti

- Provide HPF-like language enhancements to support block-structured applications
- Treat things statically, where we can
  - Like Fortran D, etc.
- Use run-time support where we can’t establish compile-time bounds

Runtime Support

- Regular_Section_Move_Sched
  - Schedule a regular section move
- Accommodates block, cyclic, and block-cyclic distributions when the bounds & strides are known at run-time
- Overlap_Cell_Fill_Sched: schedule moves for overlap and ghost cells

Compiler Support

- Additional HPF-like directives
- Static analysis for data distribution
- Insert calls for runtime workload partitioning based on data distribution

Static Analysis

- Done on for_all loop parameters
- Categorize one of three ways
  - No communication necessary
  - Copy overlap regions
  - Copy regular sections
**Experiment: Overhead**

- Extra time from library calls and schedule building isn't too bad

**Experiment: Multiblock Code**

- Within 20% of hand-parallelized F77
- Difference between compiler-parallelized & hand-parallelized F90 is mostly in computing loop bounds and searching for previously-used schedules

**Experiment: Multigrid Code**

- Within 10% of hand-parallelized code

**Experiment: Compiler Optimizations**

- Performance stinks if schedules are not saved (Version I)
- Hand-implemented reuse improves over runtime reuse (II vs. III)
- Un-implemented optimization for loop-bounds in subroutines also improves (Version IV)
KeLP

- Library for parallelization abstraction
- Works for block-structured program with the following overall structure:

```
for i = 1 to num_iters
    data motion;
    for_all ...
        parallel computation;
    end for_all
end for
```

---

Geometric Structure Abstractions

- Points (PointD), Regions (RegionD)
- Mapping regions to processors (FloorPlanD)
- Grid (GridD), indexed by a region
- Array of grids (XArrayD), structure represented by a FloorPlanD
- Region Calculus

---

Data motion abstractions

- Motion plan (MotionPlanD), list of block moves
- MoverD, actor that executes the moves specified in a motion plan
- Plan block moves
- Can extend for move + operation

**FIG. 4.** The MotionPlan encodes a set of block copy operations between grids.

---

Implementation

- All processors store a locally relevant part of the motion plan
- Mover performs non-blocking communication in the data motion step of the outer loop
- Avoiding unnecessary buffer-packing when possible
Implementation

- Mover could be extended to move things a different way
- Utilize underlying transport
- Exploit MPI differently (all-to-all, for instance)
- Move + operation

Experiment: Conventional Applications

- Multigrid solver, FFT, matrix multiply
- KeLP did no more than 10% worse than existing code
- Sometimes did better

Experiment: Jacobi

- Three KeLP versions vs. Hand-parallelized version by manipulating the motion plan
  I. Just use fillpatch as necessary
  II. Eliminate unnecessary corner overlap cells
  III. Use contiguous faces where possible

Experiment: Jacobi

- Improvements do show benefit
  - Great benefit for using contiguous faces
  - Hand-coded uses inter-loop optimization out of the scope of KeLP
More Recent Developments

• Global Arrays
  • Library for explicit shared memory
  • Programmer dictates locality
• A++/P++ (part of Overture from LLNL)
  • Fortran-like arrays
  • P++ provides a HPF-like interface through library

Overall Conclusion

• We can get close to hand-coded performance with these systems
• Are they easier to use?