Runtime Parallelization

Gary Jackson (with mods by A. Sussman)

G. Agrawal, A. Sussman, and J. Saltz, "An Integrated Runtime and Compile-time Approach for Parallelizing Structured and Block Structured Applications", *IEEE Transactions on Parallel and Distributed Computing*, 6(7), 1995.

S.J. Fink, S.R. Kohn, and S.B. Baden, "Efficient Run-time Support for Irregular Block-Structured Applications", *Journal of Parallel and Distributed Computing*, 50(1), 1998.

Notes

- Group project interim reports due Friday, April 21
- Midterm exam on April 27

Outline

- Overview
- Compiler-driven: Multiblock Parti
- 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

Multiblock Parti

- Provide High Performance Fortran-like language enhancements to support block-structured applications
- Treat things statically, where we can
  - Like Fortran D, High Performance Fortran, 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 / ghost cells

Compiler Support

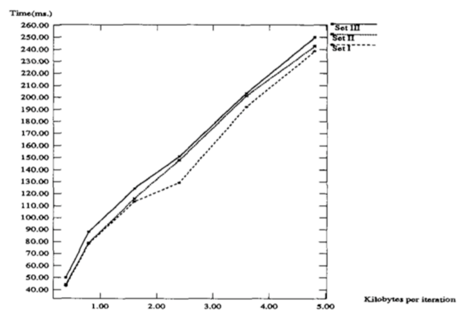
- Additional HPF-like directives
- Static analysis for data distribution
- Insert calls for runtime workload partitioning based on data distribution

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



Set I : Communication (Max. message aggregation)
Set II : Communication and copying
Set III : Communication, copying and schedule building

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

One Block: 49 × 9 × 9 Mesh (50 Iterations)

Number of Processors	Compiler Parallelized	Hand Parallelized F90	Hand Parallelized F77
4	6.99	6.88	6.20
8	4.17	4.06	4.00
16	2.47	2.35	2.28
32	1.55	1.45	1.41

Fig. 5. Performance comparison for small mesh, one block (sec).

Two Blocks: 49 × 17 × 9 Mesh (50 Iterations)

Number of Processors	Compiler Parallelized	Hand Parallelized F90	Hand Parallelized F77
8	7.49	6.69	6.17
16	4.64	4.07	4.03
32	2.88	2.32	2.30

Fig. 6. Performance comparison for larger mesh, two blocks (sec).

Experiment: Multigrid Code

- Within 10% of hand-parallelized code

No. of Proc.	Compiler: First Iteration	Compiler: Per-subsequent Iteration	By Hand: First Iteration	By Hand: Per-subsequent Iteration
8	4.80	2.29	4.60	2.14
16	3.84	1.38	3.41	1.35
32	3.03	.95	2.48	.88

Fig. 7. Semicoarsening multigrid performance (sec).

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)

Two Blocks: 49 × 9 × 9 Mesh (50 iterations)

No. of Proc.	Compiler Version I	Compiler Version II	Compiler Version III	Compiler Version IV	Hand F90
4	13.45	7.63	7.41	7.33	6.79
8	15.51	4.78	4.58	4.54	4.19
16	11.72	2.85	2.71	2.62	2.39
32	8.01	1.85	1.79	1.66	1.47

Version I: Runtime Library does not save schedules
Version II: Runtime Library saves schedules
Version III: Schedule reuse implemented by hand
Version IV: Loop bounds reused within a procedure

Fig. 8. Effects of various optimizations (sec).

KeLP

- Library for parallelization abstraction
- Works for block-structured programs 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

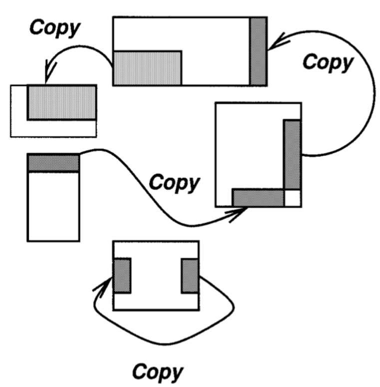


FIG. 4. The MotionPlan encodes a set of block copy operations between grids.

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 with explicit shared memory programming model
  - 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?