CMSC 714
Lecture 18
Runtime Parallelization

Gary Jackson and Alan Sussman

Notes

- Midterm exam in 2 weeks, on Tuesday, Nov. 13
  - on readings through previous week
- Group Project interim report due Nov. 9

Outline

- Overview
- Compiler-driven: Multiblock Parti
- Library-driven: Global Arrays
- 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: (somewhat) 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

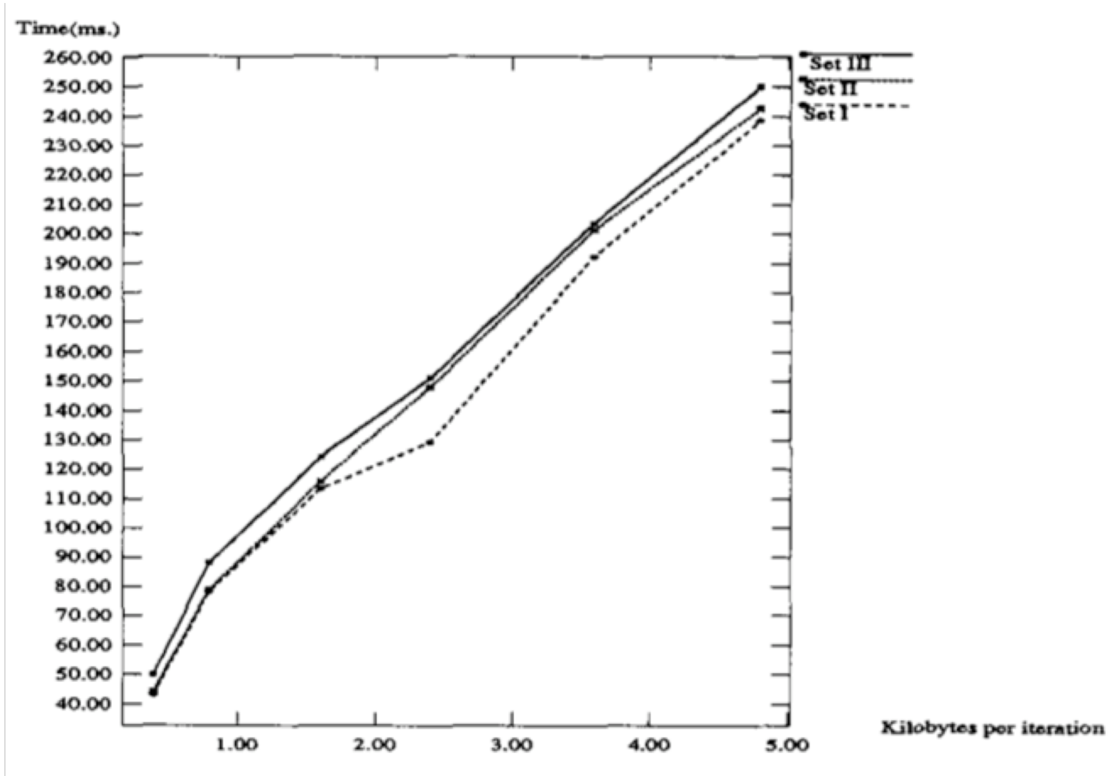
- Additional HPF-like directives
- Static analysis for data distribution
- Insert calls for runtime workload partitioning based on data distribution

Static Analysis

- Done on for_all loop parameters
- Categorize one of three ways
  - No communication necessary
  - Copy overlap (ghost) 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: $49 \times 9 \times 9$ 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).

Global Arrays

- Library for parallelization abstraction
  - On distributed memory systems (clusters)
  - SPMD model
- Idea is to program as if shared memory, but move data between distributed memory and local memory as needed
  - Only operate on local data within each process
- Compatible with MPI, so can mix GA calls with MPI calls as needed
  - Built on top of ARMCI library for one-sided communication (put/get) – portable and efficient
  - One-sided can be more efficient than send/receive, as shown for some applications, since less synchronization

Global Arrays

- Programmer can map both ways between global and local views of data objects (arrays)
  - But only compute on local view
- GA is also aware of SMP (multi-core) nodes
  - To support “mirrored view” – caching distributed memory data in shared memory for multiple processes to use
- Also has direct support for ghost cells
  - To avoid distributed to local copies for structured grid applications
  - And for periodic boundary conditions
- Paper also talks about sparse data management
  - But not clear how efficient GA is for computing with sparse matrices/vectors

Global Arrays

- Data parallel interfaces to operate on global arrays
  - To interface with other libraries like BLAS, SCALAPACK to perform data parallel collective operations
- Disk Resident Arrays allow extending global arrays to out-of-core
  - Basically distributed memory stored on (local) disks
  - With operations to move data between disks (instead of distributed memory) and local memory in each process
- Support for mapping global arrays onto subsets of processors
- Many similarities to Multiblock Parti, but also supports copies from global to local view
- Performance results show good scaling on several applications for parallel systems available at that time
  - All the applications employ large, dense, multi-dim data grids
  - And can take advantage of both low-latency and high-bandwidth networks (through ARMCI)

Overall Conclusion

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