

An Integrated Runtime and Compile-Time Approach for Parallelizing Structured and Block Structured Applications

Gagan Agrawal, Alan Sussman, Joel Saltz

Presented by Sam Angiuoli

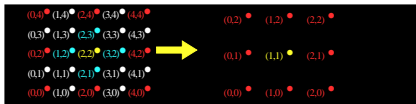
Overview

- Using a combination of compile-time and run-time analysis to optimize memory access and load balancing
- Targeted for HPF applications on distributed memory machines where data distribution and access parameters are not known at compile time
- Compiler extensions insert calls to a library of routines that can optimize data access at runtime

Multigrid

■ Multigrid applications

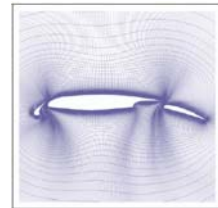
- Use a combination of course and fine meshes to accelerate computation via approximations
- Used for solving PDEs
- Mesh resolution and nesting can vary at runtime
- Data distribution may be necessary within a meshes and when propagating values between meshes



Multiblock

■ Multiblock applications

- Sets of meshes (blocks) with non-uniform structure
- Adjacent blocks may have different mesh sizes
- Outer loop performs time step
- Data distribution both within and between blocks
- Data distribution dependent on the object's mesh



Runtime library optimizations

- Multiblock Parti library
- Dynamic data distribution
 - Regular section moves
- Partitioning loops via symbolic loop bounds and strides

Runtime primitives

- Communication schedule
 - Schedule describes data motion between processors
 - Goal is to look at the data distribution at runtime and build a schedule that optimizes data communication
 - Schedules can be reused for similar data distributions
- Data movement

Data distribution

- Regular section move
 - Forall loops with array assignments where loop bounds and strides not known at compile time
 - Schedule determines data elements sent and received by each processor
 - Moving array elements between processors ie. From one distributed array to another
 - Regular_section_copy_sched(...)
- Often only ghost/overlap cells between adjacent blocks/meshes need to be copied
 - Overlap_cell_fill_sched(...)

Data distribution cont'

- Loop partitioning → handling symbolic loop bounds and strides
 - Owner computes rule
 - Loop iteration performed by process owning LHS array element
 - Loop bound transformations
 - Local_lower_bound()/Local_upper_bound() used to transform loop bounds
 - Mapping for local indices
 - Local_to_global/global_to_local

Compiler support

- Extensions to accommodate operations in the runtime library
- Extend processor abstraction to support subspaces
 - PROCESSORS P(N)
 - PSUBSPACE P1 IS P(UPPER:LOWER)
- Extend Align to specify border/ghost cells
 - ALIGN A(i,j) WITH T(i:2:3;j:2:3)

Communication patterns

- Methodology for analyzing forall loops and performing data moves when necessary
- Case I
 - Array A,B aligned to different template
 - No information about relationship
- Case II
 - Array A,B aligned to same template
 - A,B same size and shape
- Case III
 - Array A,B aligned to same template
 - Different loop bounds, strides

Communication patterns

- Look at loop bounds/strides and classify loops as follows
 - Not requiring any communication
 - Can be handled by filling overlap/ghost cells
 - Requiring regular section moves

Align A(i,j) with T(i,j)
Align B(i,j) with T(2i-1+3:2i)

L.H.S. Expression	R.H.S. Expression	Regular Section Move Required	Overlap Cell Fill Required
A(i,j)	B(i,j)	Yes	No
A(i,j)	B(j,i)	Yes	No
A(2 <i>i</i> ,j)	B(j,i)	No	Yes
A(2 <i>i</i> ,j)	B(j,1+3)	No	Yes
A(2 <i>i</i> ,j)	B(j-1,i)	No	No

```

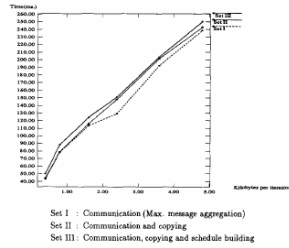
C ORIGINAL HPF CODE
C Arrays A, B are distributed identically
forall (i = 1:100:2,j = 1:50) A(i,j) = B(2*i,j)

C TRANSFORMED CODE
NumSrcDim = 2      NumDestDim = 2
SrcDim(1) = 2      DestDim(1) = 1
SrcDim(2) = 1      DestDim(2) = 2
SrcLoc(1) = 2      DestLoc(1) = 1
SrcLoc(2) = 1      DestLoc(2) = 1
SrcHis(1) = 100    DestHis(1) = 100
SrcHis(2) = 100    DestHis(2) = 50
SrcStr(1) = 2      DestStr(1) = 2
SrcStr(2) = 2      DestStr(2) = 1
Sched = Regular_Section_Move_Sched(DAD,DAD,NumSrcDim,NumDestDim,
SrcDim,SrcLoc,SrcHis,SrcStr,
DestDim, DestLoc, DestHis, DestStr)
Call Data_Move(B,Sched,A)
    
```

Fig. 3. Regular section move example.

Overhead

- Cost of copying (I vs. II) < 5%
- Scheduling time is small
 - Especially true on larger problems with regular access patterns



Case studies

- Comparing compiler optimized with hand optimized codes
- Both using same runtime library routines
- Compiler optimized within 10-20% of hand optimized

Number of Processors	Compiler Parallelized	Hand Parallelized P90	Hand Parallelized P77
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).

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. Semicoursing multigrid performance (sec).

More results

- Version I rebuilds schedule during each loop iteration
- Increased communication from distributing blocks over entire process space

No. of Proc.	Compiler Version I	Compiler Version II	Compiler Version III	Compiler Version IV	Hand P90
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).

No. of Processors	Blocks Mapped of Entire Processor Space	Blocks Mapped Against Diagonal Processor Space
4	8.99	7.59
8	5.14	4.54
16	3.24	2.83
32	2.41	1.87

Fig. 9. Effect of data distribution.