

# Efficient Run-time Support for Irregular Block-Structured Applications

By Stephen J. Fink and Scott B. Baden and Scott  
R. Kohn



Presented for your delectation by Asad B. Sayeed

# Background

- Main type of application considered: scientific numerical methods.
- These applications often use structured irregular representations to improve accuracy.
  - Difficult to implement.
  - Cause unpredictable/irregular communication patterns, impeding performance optimization.
- Goal: assist programmer in arranging parallelism so the data layout and distribution best exploit memory arrangements.

# Kernel Lattice Parallelism

- KeLP: Kernel Lattice Parallelism.
  - Library for higher level abstractions for managing data layout and data motion.
  - Applications with dynamic block structures: uniform rectangular data arrays with irregular data motion.
  - Geometric programming abstractions represent data layout and motion patterns
  - Data orchestration model: separate description of motion patterns from interpretation/implementation.
  - Structural abstraction: separate structure of data from storage.

# Kernel Lattice Parallelism

- System implemented on top of MPI in C++.
- Data orchestration implemented via MotionPlans and Movers.
  - Programmers define MotionPlans to schedule communication via geometric operations on memory structure.
  - Movers interpret the plans so as to conform to the hardware architecture and other application-specific issues.

# Programming Model

- Programs begin with a single (logical) control thread.
- for\_all loop iterations: each one executes independently on one SPMD process.
- Storage model: distribute each block of data to its own logical address space, one space per processor.
- Little compiler automation, even for consistency.
  - Programmer explicitly describes data decomposition and also data motion (via block copy operations).

# Data Layout Abstractions

- Four core data decompositions abstractions: Point, Region, Grid, XArray, inherited from KeLP's predecessor LPARX. KeLP innovation: FloorPlans.
  - Point: represents point in n-dim space.
  - Region: rectangular subset of Points..
  - Grid: array of data indexed by Region.
  - XArray : array of Grids of different (irregular) shape.
  - FloorPlans: array of Regions representing processor assignments for XArray.

# Data Layout Abstractions

- Regions are constructed by Region calculus.

Table 2  
Region Calculus Operations Used in the Examples

Operation	Interpretation
$extents(\text{Region } R, \text{int } i)$	Length of Region $R$ along the $i$ th axis
$shift(\text{Region } R, \text{Point } P)$	Translation of Region $R$ by the vector $P$
Region $R \cap$ Region $S$	Geometric intersection of Regions $R$ and $S$
$grow(\text{Region } R, \text{Point } P)$	Region $R$ padded with $P(i)$ cells on $i$ th axis

- XArrays and FloorPlans:

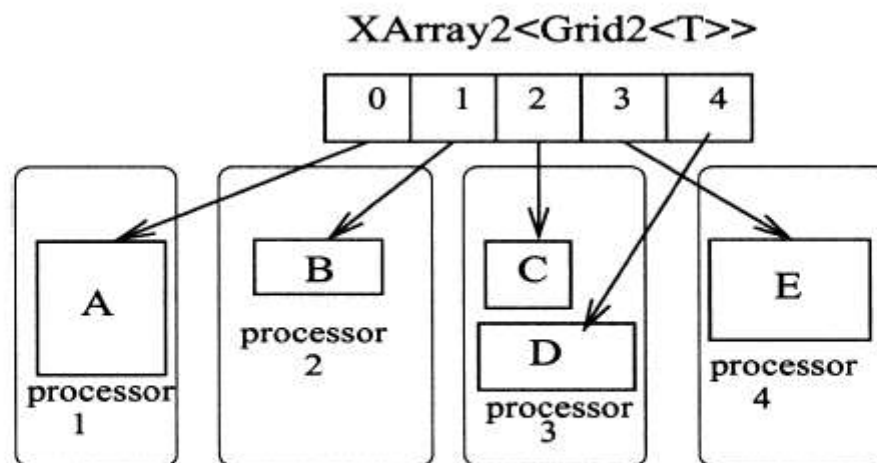


FIG. 3. The XArray is a coarse-grained distributed array of blocks of data, whose structure is described by a FloorPlan. The blocks may have different sizes and each is assigned to a single address space.

# Data Motion Abstractions

- MotionPlan
  - Data motion pattern defined over Xarrays.
  - Specified by programmer as set of array copy operations built via Region calculus.
  - Let  $G, H$  be Grids; let  $R, S$  be Regions.  
 $\{G \text{ on } R \rightarrow H \text{ on } S\}$  means copy index region  $R$  from  $G$  to region  $S$  from  $H$ .
- Mover: analyzes MotionPlan and performs movement.
  - Programmer can extend the Mover class to represent various communication operations.

# Data Motion Abstractions

- MotionPlans illustrated:

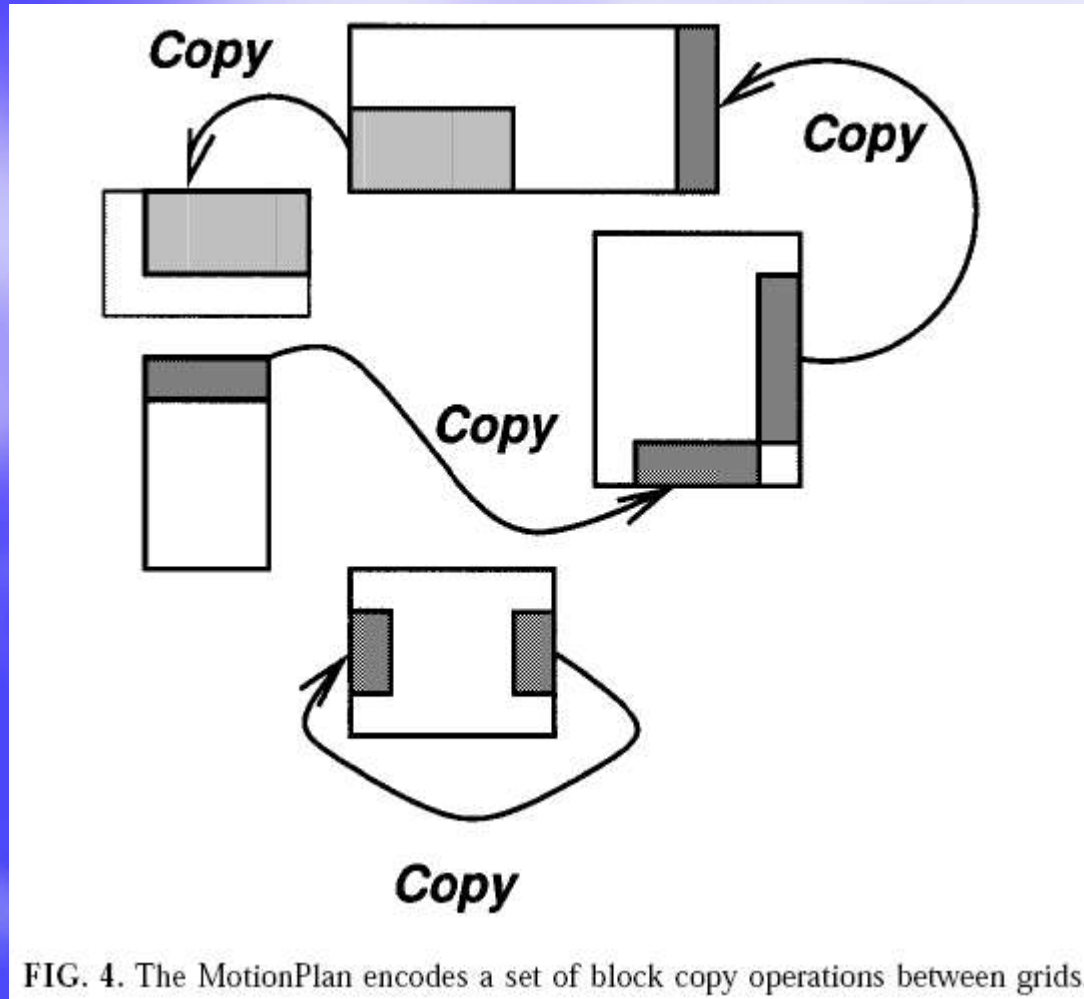


FIG. 4. The MotionPlan encodes a set of block copy operations between grids.

# Data Layout and Data Motion

- Summary of classes:

**Table 1**  
A Brief Synopsis of the KeLP Data Types

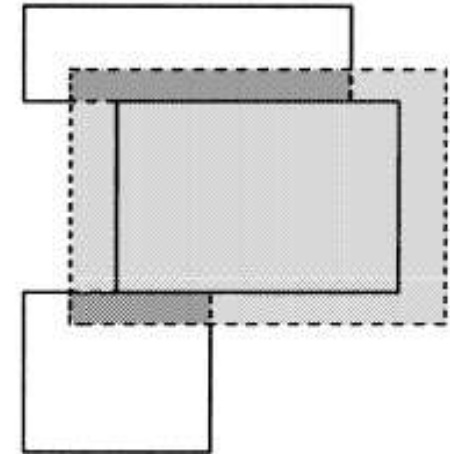
Geometric structural abstractions		
Name	Definition	Interpretation
PointD	$\langle \text{int } l_0, \text{int } l_1, \dots, \text{int } l_{D-1} \rangle$	A point in $Z^D$
RegionD	$\langle \text{PointD } l, \text{PointD } h \rangle$	A rectangular subset of $Z^D$
FloorPlanD	Array of $\langle \text{RegionD } R, \text{int } p \rangle$	A set of regions, each assigned to a processor $p$
MotionPlanD	List of $\langle \langle \text{int } f, \text{RegionD } R_f \rangle, \langle \text{int } t, \text{RegionD } R_t \rangle \rangle$	Block-structured communication pattern between two FloorPlans
Data types that interpret abstractions		
Name	Description	
GridD	A multidimensional array whose index space is a RegionD	
XArrayD	An array of GridDs; structure represented by a FloorPlanD	
MoverD	Object that atomically performs the data motion pattern described by a MotionPlan	

# Simple Data Motion Example

- fillpatch: fills in ghost cells from logically overlapping grids.
- Code and example of irregular XArray.

```
BuildFillpatch(XArray X, MotionPlan M)
begin
  for each  $i \in X$ 
     $I = grow(X(i), -1)$ 
    for each  $j \in X$ 
      if ( $i \neq j$ ) {
         $R = I \cap X(j)$ 
         $M.Copy(X, i, R, X, j, R)$ 
      }
    end for
  end for
end
```

a



b

FIG. 5. (a) Pseudocode to generate a fillpatch MotionPlan M to fill in ghost cells for XArray x. (b) The dark shaded regions represent ghost regions that are copied into the central Grid.

# Bigger Ghost Cells Example

- Elliptic PDE solver:

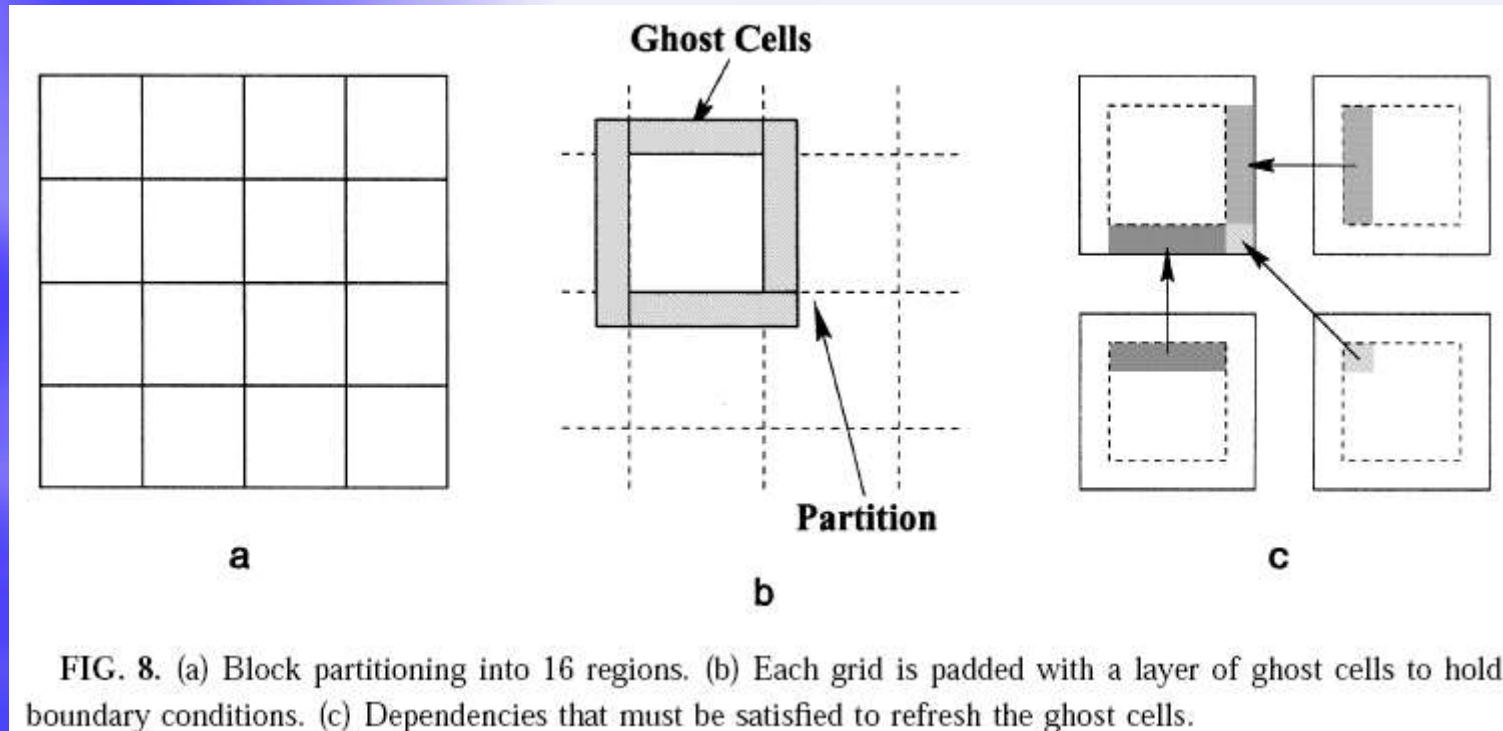
```
(1) Region2 domain(1,1,N,N);
(2) Processors2 P;
(3) Decomposition2 T(domain);
(4) T.distribute(BLOCK,BLOCK,P);
(5) for_1(i,T)
(6)     T.setregion(i,grow(T(i),1));
(7) end_for
(8) XArray2< Grid2<double> > U(T);
(9) InitGrid(U);
(10) int RedBlack = 0
(11) for (int k=0 ; k<NITERS*2; k++) {
(12)     fillGhost(U);
(13)     for_all(i,U)
(14)         sweep(U(i),RedBlack);
(15)     end_for_all
(16)     RedBlack = 1 - RedBlack;
(17) }
```

FIG. 7. Main procedure for red-black Gauss-Seidel example.

- Region2, etc are 2D arrays.

# Bigger Ghost Cells Example

- Elliptic PDE FloorPlan:



- for<sub>1</sub> vs for<sub>all</sub> -> current thread vs distributed
- The for<sub>1</sub> loop does initial ghost cell padding.
- Sweep: performs iteration, probably in Fortran.

# Bigger Ghost Cells Example

- fillGhost function: central to parallelism.

```
void fillGhost(XArray2<Grid2<double> >& X)
{
(1)  MotionPlan2 M;
(2)  for_1(i,X)
(3)    Region2 inside = grow(X(i).region(), -1);
(4)    for_1(j,X)
(5)      if (i != j) {
(6)        M.CopyOnIntersection(X,i,X,j,inside);
(7)      }
(8)    end_for
(9)  end_for
(10) Mover2<Grid2<double>, double> DM(X,X,M);
(11) DM.execute();
}
```

FIG. 9. FillGhost( ) function for code of Fig. 6.

- Mostly just implements fillPatch from before.
  - Last two lines perform the movement.
  - We're recomputing ghost cells each time: not normal.

# Implementation Issues

- KeLP predecessor: LPARX.
  - LPARX allowed asynchronous one-sided communication: creates barriers for process state global synchronization.
  - KeLP: bans copy operations from for\_all loops, eliminating this problem; ie, only for\_1 loops perform copies. Each process stores relevant portion of movement plan.
- Mover: implemented via nonblocking MPI send
  - In and out buffers allocated to each process.
  - Receives data while it waits for sends to finish.

# Performance

- Comparison to MPI.
  - Three benchmarks involving heavy matrix computation: NAS-FT, NAS-MG, SUMMA.

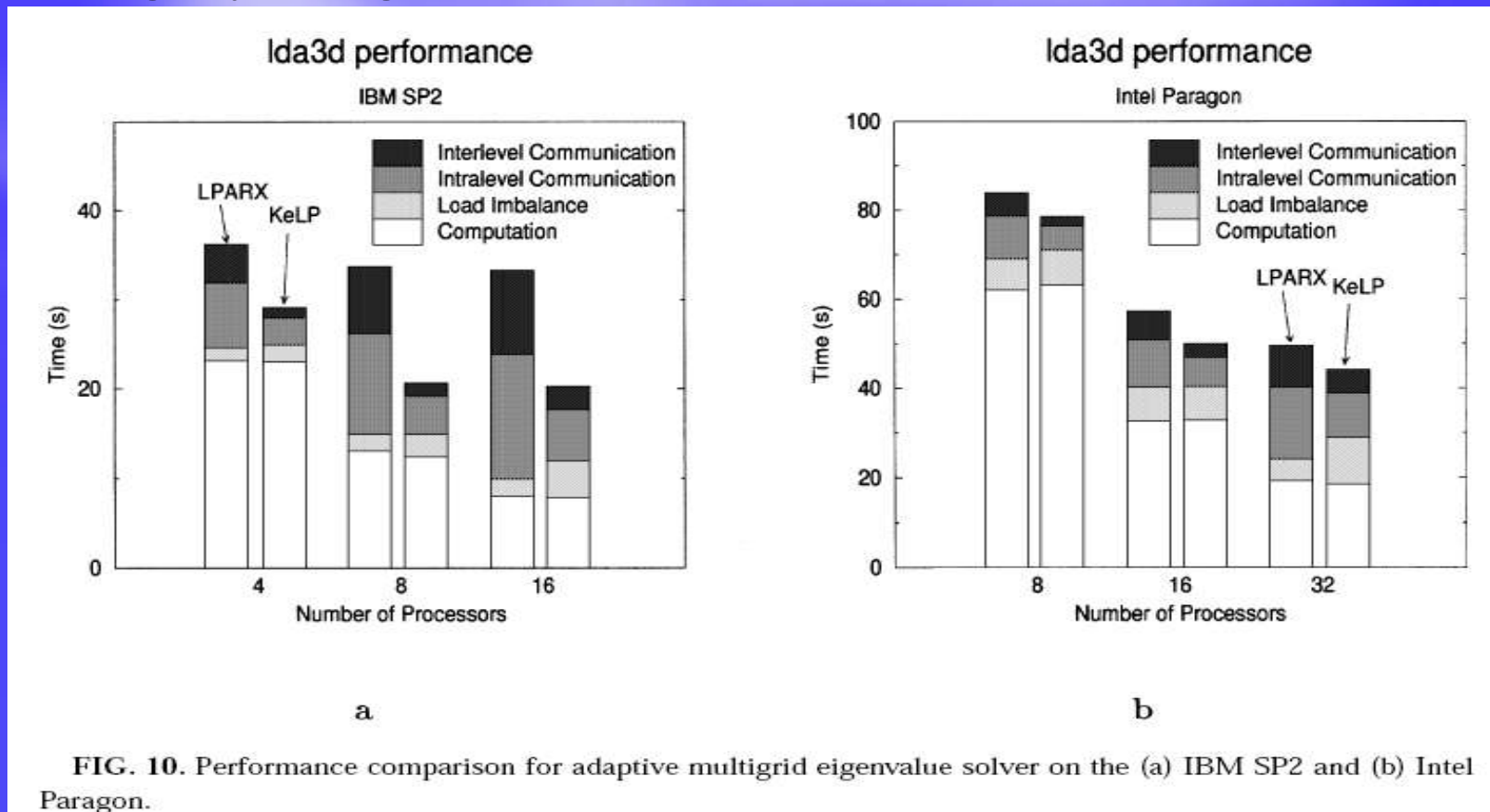
**Table 3**  
**Comparison of KeLP Performance with Three MPI Reference Codes on an IBM SP2**

Code	KeLP performance (MFLOPS)				MPI code performance (MFLOPS)			
	8 nodes	16 nodes	32 nodes	64 nodes	8 nodes	16 nodes	32 nodes	64 nodes
NAS-MG Class B	311	558	851	1266	283	480	706	1342
NAS-FT Class A	111	203	385	713	101	223	420	749
SUMMA	1231	2452	4699	9308	1255	2465	4518	8380
	Normalized KeLP running time (normalized so MPI running time = 1.0)				Percentage of time spent communicating in KeLP version			
NAS-MG Class B	0.91	0.86	0.83	1.06	22	32	46	60
NAS-FT Class A	0.91	1.10	1.09	1.05	15	21	21	23
SUMMA	1.02	1.00	0.96	0.90	23	26	27	30

- Very conservatively translated to KeLP from MPI.

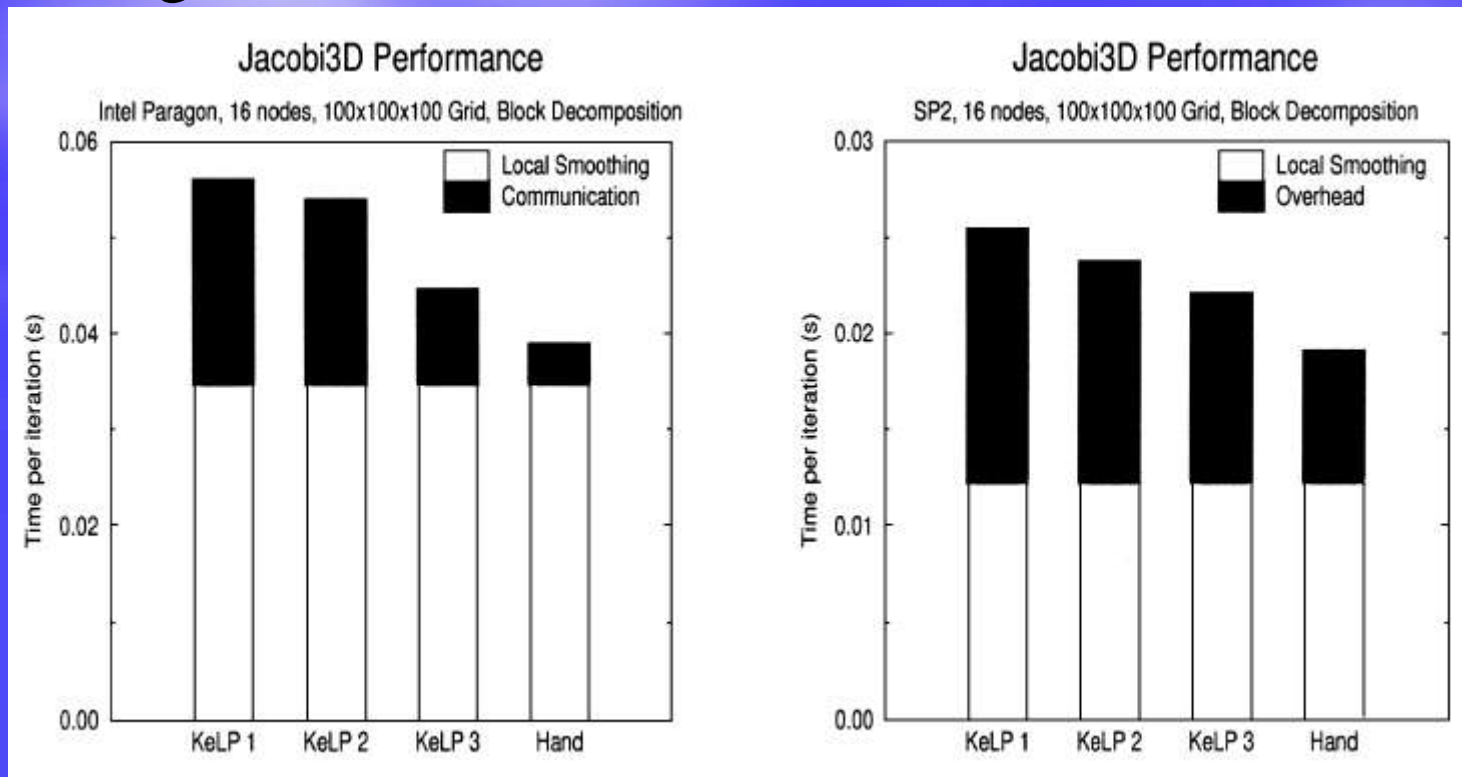
# Performance

- Adaptive multigrid: lda3d
  - Eigenvalue solver from “ab initio” materials science.
  - Highly irregular communication.



# Performance

- jacobi3d: tuning communication performance.
  - Three KeLP versions and one hand-coded version.
  - KeLP versions vary by ghost cell communication arrangements.



# Related Work

- KeLP: structural abstraction from LPARX combined with inspector/executor communication analysis.
- Other structural abstraction implementation: BOXLIB, DAGH. More specialized.
- Inspector/executor appears in Multiblock PARTI, which does not allow irregular block decompositions—doesn't have same level of structural abstraction.
- Number of other related applications.

# Conclusion

- Structural abstractions hide some of the dirty work required for efficient communication within irregular block decompositions.
- Despite KeLP being a high-level abstraction over MPI, performs very favorably compared to MPI.
- Inspector/executor paradigm (MotionPlans vs Movers) allows retargetting to various situations.



*Kernel Lettuce Decomposition,  
more widely eaten than KeLP,*