Unstructured Mesh Computations: Effects of Ordering Strategies, Programming Paradigms, and Architectural Platforms

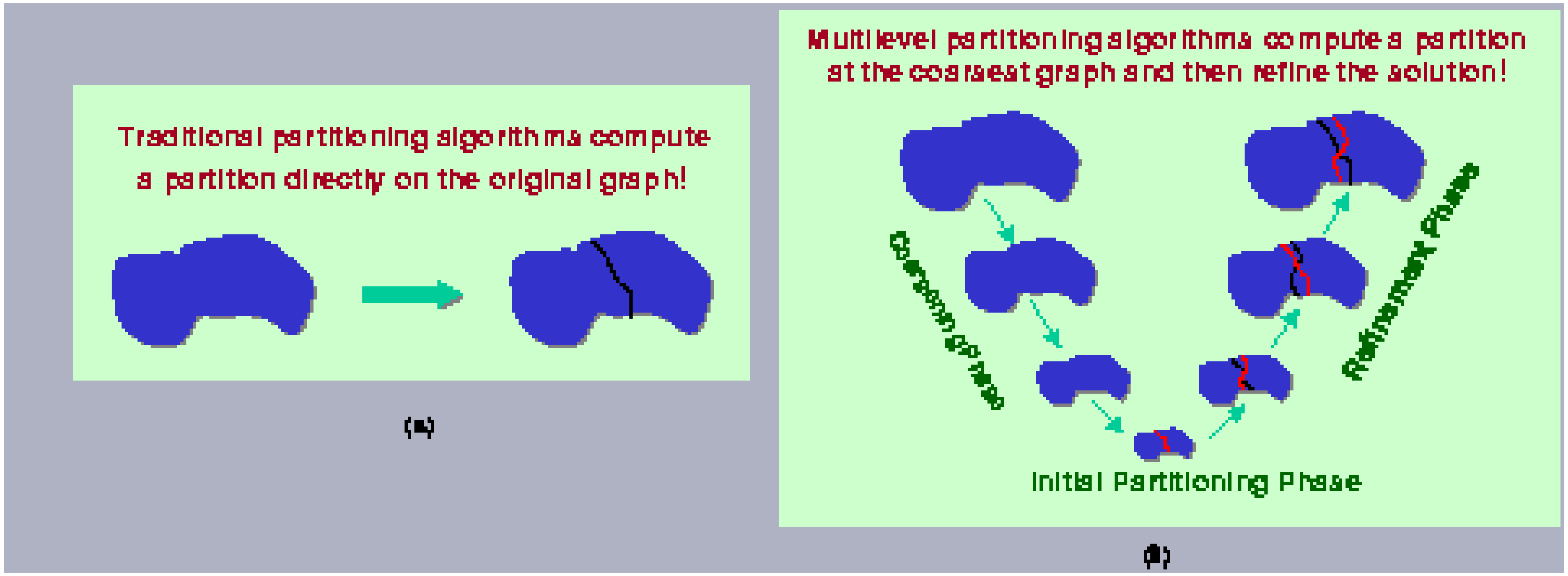
Leonid Oliker
loliker@lbl.gov
www.nersc.gov/~ oliker
NERSC, Lawrence Berkeley National Laboratory

Overview

- Generally believed that unstructured grid techniques will constitute a significant fraction of future high performance computing
- Unstructured meshes used to represent complex (evolving) geometries
- Underlying matrix is assembled and used to solve set of PDE's
- Parallel computing required to solve large sparse systems
- Leading parallel systems and programming paradigms
  - T3E/MPI, Origin2000/OpenMP, Tera MTA/MT
- Partitioning strategy is required to decompose unstructured domain onto parallel system
  - Multilevel (MeTiS), Linearization (RCM, SAW), Graph Coloring
- Examples: Conjugate Gradient (CG) performance based on SPMV
- Unstructured mesh adaptation: dynamically adaptive algorithm

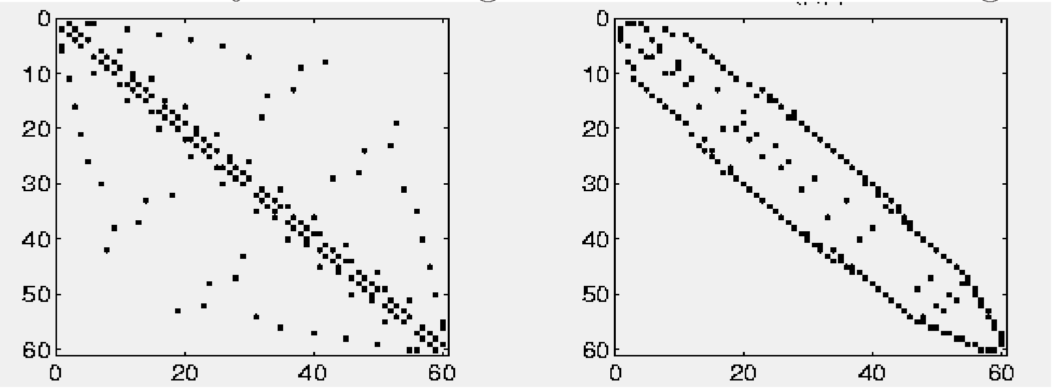
Graph Partitioning Strategies: MeTiS

- Most popular multilevel partitioners, goal:
  - Balance computational workload
  - Minimize edge cuts (interprocessor communication)
- Collapses vertices and edges, using heavy edge matching scheme, applies greedy algorithm to coarsest graph, uncoarsens it back using greedy graph growing + Kenighan-Lin.



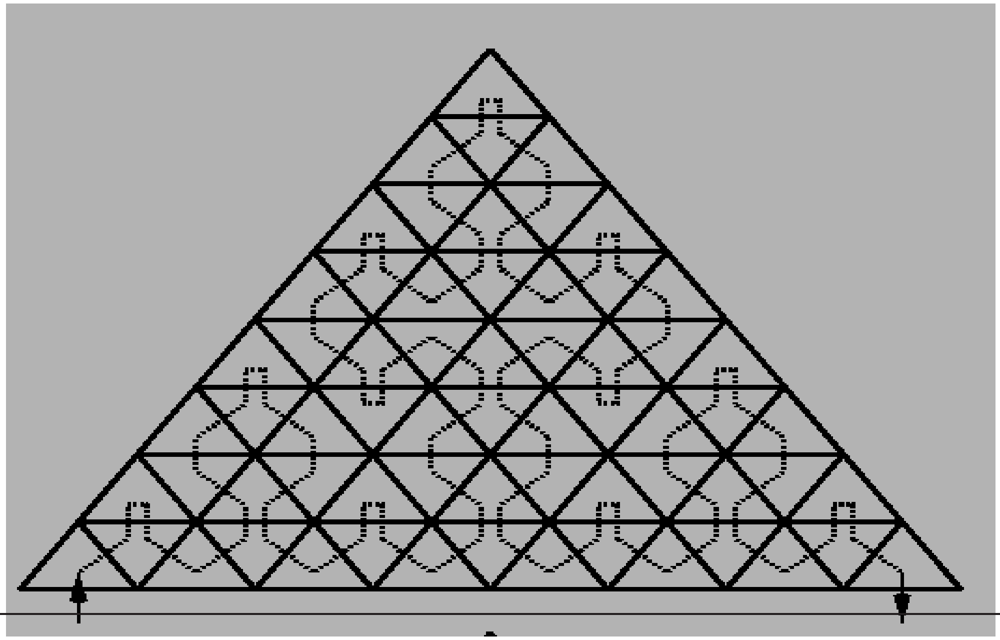
Linearization Strategies: Reverse Cuthill-McKee (RCM)

- The bandwidth, or profile, of the matrix has a significant impact of the efficiency of linear systems and eigensolvers
- RCM is a *geometry based* algorithm which generates a permutation such that the nonzero entries are close to the diagonal.
- Good preordering for LU or Cholesky factorization (reduces fill).
- Also improves cache performance, but does not explicitly reduce edge-cuts.
- Can be used as partitioning strategy by assigning each of P segments of the RCM enumeration to a processor.
- From vertex of min degree, generates a level structure by breadth-first search and orders the vertices by decreasing distance from the original vertex



Linearization Strategies: Self-Avoiding Walks (SAW)

- SAW is *mesh based* technique similar to space-filling curves
- Two consecutive triangles in the SAW share an edge or vertex (no jumps)
- SAW visits each triangle exactly once, entering/exiting over an edge or vertex
- SAW used to improve parallel efficiency related to locality (cache reuse) and load balancing, but does not explicitly address edge cuts (interproc. comm.)
- SAW amenable to hierarchical coarsening and refinement



Sparse Matrix Vector Multiply and Conjugate Gradient

- SPMV one of most heavily used kernels in large-scale numerical simulations
- To perform a SPMV, $y \leftarrow Ax$, assume that nonzeros of matrix A are stored in the Compressed Row Storage (CRS) format
- Dense vector x is stored sequentially in memory with unit stride. Various numbering of the mesh elements/vertices result in different nonzero patterns of A, which in turn cause different access patterns for the entries of x.
- The Conjugate Gradient (CG) algorithm is the oldest and best-known Krylov subspace method used to solve the linear system $Ax = b$. The method starts from an initial guess of the vector x. It then successively generates approximate solutions in the Krylov subspace, and search directions used in updating the approximate solution and residual.
- SPMV usually accounts for most of the flops within a CG iteration.

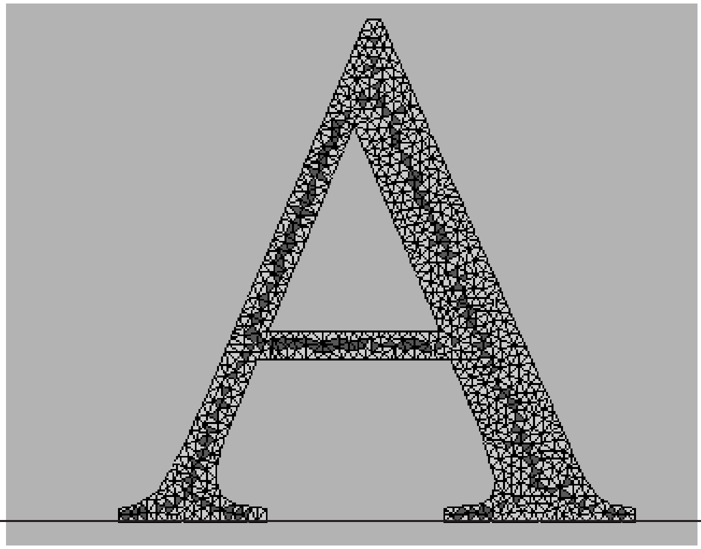
Sparse Matrix Vector Multiply and Conjugate Gradient

Compute $r_0 = p_0 = b - Ax_0$ for some initial guess x_0
for $j = 0, 1, \ldots,$ until convergence
  $\alpha_j = (r_j, r_j)/(Ap_j, p_j)$
  $x_{j+1} = x_j + \alpha_j p_j$
  $r_{j+1} = r_j - \alpha_j A p_j$
  $\beta_j = (r_{j+1}, r_{j+1})/(r_j, r_j)$
  $p_{j+1} = r_{j+1} + \beta_j p_j$
endfor

- Each iteration of CG involves one SPMV for Ap_j, three vector updates (AXPY) for x_{j+1}, r_{j+1}, and p_{j+1}, and three inner products (DOT) for the update scalars α_j and β_j
- For a symmetric and positive definite linear system, these conditions imply that the distance between the approximate solution and the true solution is minimized
- Suppose the matrix A is of order n and has nnz nonzeros. Then, one SPMV involves $O(nnz)$ floating-point operations, while AXPY and DOT involve only $O(n)$ floating-point operations. Thus, for many practical matrices, SPMV dominates the other two operations.

Test Problem

- 2D Delaunay triangulation of letter "A" generated by Triangle package.
- Contains 661,054 vertices and 1,313,099 triangles
- Underlying matrix assembled by assigning random value in $(0, 1)$ to each (i, j) entry corresponding to the vertex pair (v_i, v_j), where $1 \leq distance(v_i, v_j) \leq 3$. All other entries set to 0
- Diagonal set to 40, making it diagonally dominant (positive definite)
- Final matrix has approx 39 entries per row and 25,753,034 nonzeros.
- CG converges in 13 iterations, SPMV accounts for 87% of flops



Distributed Memory Implementation

- Each processor has its own local memory that only it can directly access.
- To access the memory of another proc. a copy of the desired data must be sent across the network using message passing (MPI or PVM)
- To run program on these systems, user must decide how the data should be distributed (or redistributed) and organize communication structure
- Allows user to design efficient code at cost of high code complexity
- Parallel CG used calls special sparse linear library (Aztec)
- Matrix A is partitioned into blocks of rows and each block assigned to proc.
- Associated component vectors x and b are distributed accordingly
- Communication may be needed to transfer some components of x
- AXPY: only local computation, DOT: local sum followed by global reduction
- T3E: 450 MHz Dec Alpha processor (900 Mflop peak theoretical), 96KB secondary cache, interconnected through 3D torus.

Distributed Memory: Locality and Communication Statistics

P	Avg. Cache Misses (10^6)				Avg. Communication (10^6 bytes)			
	ORIG	METIS	RCM	SAW	ORIG	METIS	RCM	SAW
8	3.6842	3.0340	3.7490	2.0042	3.2275	0.0107	0.0308	0.0488
16	2.0072	1.3305	1.9049	0.9706	2.3643	0.0108	0.0315	0.0362
32	1.0597	0.6576	1.0172	0.5073	1.4918	0.0092	0.0316	0.0302
64	0.6011	0.3581	0.5150	0.2900	0.8285	0.0079	0.0316	0.0229

- Results: ORIG ordering has large edge cut (interprocessor comm), and poor locality (high number of cache misses)
- MeTiS minimizes edge cuts, while SAW minimizes cache misses

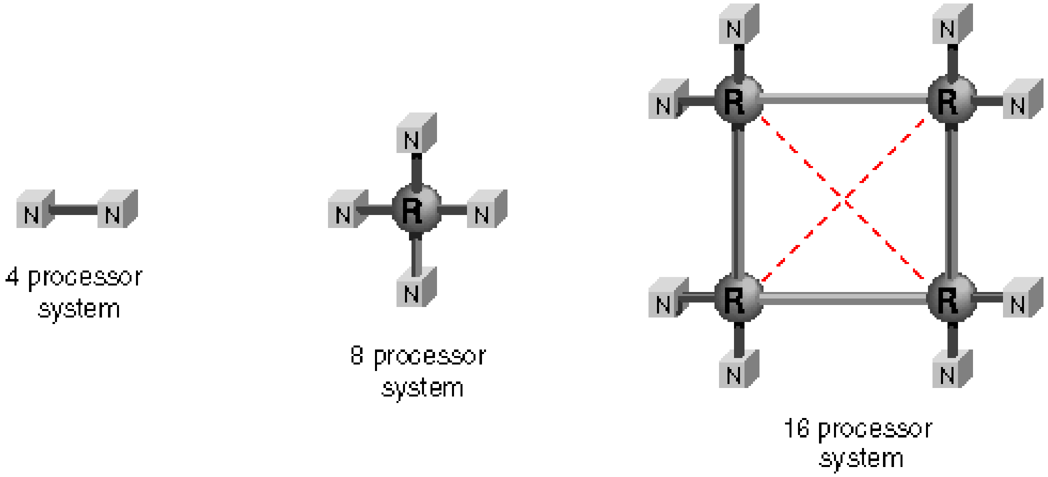
Distributed Memory: Runtime of SPMV and CG

P	ORIG		METIS		RCM		SAW	
	SPMV	CG	SPMV	CG	SPMV	CG	SPMV	CG
8	0.5622	8.6519	0.4758	7.6617	0.3812	6.1853	0.1708	2.9158
16	0.3252	5.0929	0.2682	2.9092	0.1927	3.1979	0.0861	1.4912
32	0.1990	3.1667	0.0870	1.4677	0.0951	1.6615	0.0442	0.7948
64	0.1191	1.9287	0.0559	0.9614	0.0451	0.8816	0.0283	0.4616

- Ordering/Partitioning is required to achieve performance and scalability
- For this combination of applications and architectures, improving cache reuse can be more important than reducing interprocessor communication
- Adaptivity = Repartitioning (Reordering) and Remapping

Distributed-Shared Memory System

- Origin2000: SMP of nodes each contains two 250 MHz R10000 and local mem
- Hardware makes all memory equally accessible from software standpoint, by sending memory requests through routers on nodes
- Access time to memory is nonuniform (depending on # of hops)
- Topology of the interconnection is hypercube (max $log(p)$ hops)
- Each proc has large 4MB cache, where only it can fetch and store data
- If proc access data not in cache, delay while a copy is fetched
- When proc modifies word , all other copies of that cache line are invalidated



Distributed-Shared Memory Implementation

- Use OpenMP style directives to parallelize loops - less effort than MPI
- Two implementation approaches are taken:
  - SHMEM: naively assumes Origin2000 is flat shared-mem machine. Arrays not explicitly distributed, non-local data handled by cache-coherence
  - CC-NUMA: addresses underlying distributed-memory by performing data distribution
- The computational kernels of both SHMEM and CC-NUMA are identical
- Each processor assigned equal number of rows in the matrix (block)
- No explicit synchronization since no concurrent writes
- Global reduction required for DOT

Distributed-Shared Memory Results

P	SHMEM					
	ORIG		RCM		SAW	
	SPMV	CG	SPMV	CG	SPMV	CG
1	2.224	46.911	1.489	37.183	1.460	36.791
2	1.249	28.055	0.852	21.867	0.831	21.772
4	1.425	30.637	0.935	25.350	0.915	24.751
8	0.922	16.836	0.572	14.431	0.572	14.121
16	1.047	16.348	0.635	15.516	0.645	15.548
32	1.072	16.653	0.664	15.350	0.641	15.423
64	0.747	10.809	0.323	7.782	0.324	8.450

P	CC-NUMA						MPI
	ORIG		RCM		SAW		SAW
	SPMV	CG	SPMV	CG	SPMV	CG	CG
1	2.224	46.911	1.489	37.183	1.460	36.791	
2	1.218	27.053	0.851	21.454	0.829	21.229	23.145
4	0.879	17.608	0.421	10.651	0.410	10.593	7.880
8	0.535	9.824	0.220	5.575	0.216	5.516	3.815
16	0.326	6.205	0.115	2.845	0.113	2.872	1.926
32	0.197	3.584	0.061	1.548	0.060	1.514	1.075
64	0.118	2.365	0.028	0.885	0.026	0.848	0.905

Distributed-Shared Memory Results

- CC-NUMA shows significant performance gain over SHMEM, since Origin2000 is a distributed memory machine, it should be treated as such.
- Within CC-NUMA, RCM and SAW reduce runtime compared to ORIG. Indicates that intelligent ordering is necessary to achieve performance and scalability
- Little difference between RCM and SAW, probably due to large cache (need to experiment with larger mesh)
- CC-NUMA and MPI runtimes are very similar for SAW ordering, even though programming methodologies are quite different.
- Results indicate that for this class of applications, it is possible to achieve message passing performance using shared memory constructs, through careful data ordering and distribution
- Adaptivity = Repartitioning (Reordering) and Remapping

Tera MTA Multithreaded Architecture

- 255 MHz Tera uses MTA to cover latencies and keep processor saturated
- Randomized memory mapping - data layout is impossible (3D torus)
- Near uniform data access from any processor to any memory location
- No data caches, MT is used to tolerate latency (100-150 cycles per word)
- Each proc has about 100 streams hardware (including 32 registers and PC)
- Processor makes context switch on *each* cycle, choosing the next instruction from one streams ready to execute.
- A stream can execute an instruction only once every 21 clocks, even if no instructions reference memory
- Synchronization between threads is accomplished using full/empty bits in memory, allowing for fine-grained threads
- Explicit load balancing not required since dynamic scheduling of work to threads can keep processor saturated
- For MT code: no difference between uni and multiprocessor parallelism

Tera MTA Implementation

- MT implementation is trivial - only compiler directives required
- Special assertions used to indicate no loop carried dependencies
- Compiler was then able to parallelize loop segments
- Load balancing handled by OS which dynamically assigns rows to threads
- Other than reduction for DOT, no special synchronization constructs are required since no possible race conditions in CG
- No special ordering is required (or possible) to achieve parallel performance

Tera MTA Results

	ORIG	
P	SPMV	CG
1	0.378	9.86
2	0.189	5.02
4	0.095	2.53
8	0.051	1.35

- Results on 60 streams per processor
- Both SPMV and CG show high scalability (over 90%)
- Shows enough ILP in MT CG to tolerate high overhead of memory access
- 8 proc Tera faster than 32 proc O2K and 16 Proc T3E - *with no special ordering*
- Only 8 procs available, will scaling continue as P increases?
- Adaptivity = No extra work would be required to maintain performance

Summary

- Examined 3 different parallel implementations of CG using three leading programming paradigms and architectures
- MPI: Most complicated implementation, compared graph partitioning and linearization strategies
- For this class of applications traditional graph partitioners which focus on minimizing edge cuts are not necessarily as good as locality algorithms which improve cache reuse
- Shared mem CG on O2K: ordering algorithms greatly improve performance
- It is possible to achieve message passing performance using shared mem constructs through careful data ordering and distribution
- Tera MTA was easiest to program and results show that special ordering and/or partitioning schemes are not required to obtain high efficiency and scalability.

Future work

- Create a hybrid (OpenMP/MPI) implementation on new NERSC and SDSC SP system
- Examine effects of first partitioning mesh with MeTiS followed by performing SAW on each subdomain (good match for hybrid system?)
- Evaluate parallel Jacobi-Davidson eigensolver (SPMV kernel)
- Extend SAW algorithm to 3D and modify it to efficiently handle adaptivity in parallel