Using Memory Performance To Understand the Mixed MPI/OpenMP Model

Dinesh K. Kaushik
MCS Division, Argonne National Laboratory &
CS Department, Old Dominion University

David E. Keyes
Math. & Stat. Department, Old Dominion University &
ISCR, Lawrence Livermore National Laboratory

William D. Gropp
Barry F. Smith
MCS Division, Argonne National Laboratory

Acknowledgments

- Research participants were supported by
  - U.S. Department of Energy
  - U.S. Department of Education GAANN Fellowships Program
  - U.S. National Science Foundation
- Computer access was provided by
  - U.S. Department of Energy, NERSC, and ASCI centers
  - U.S. National Science Foundation NCSA center

Organization of the Presentation

- Case for hybrid MPI/OpenMP model
- Memory performance as a tool
- Performance issues for MPI and OpenMP
- Conclusions and future directions

Our View of the Hybrid Model

- MPI Extreme
  - the user manages the memory updates
- OpenMP Extreme
  - the system manages the memory updates
- Hybrid MPI/OpenMP
  - Some memory updates are managed by the user and the rest by the system

Motivation for Hybrid Model

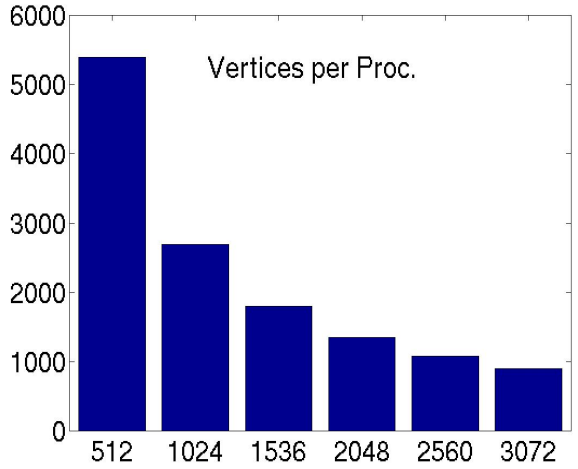
- Given
  - a scalable MPI based code
- Goal
  - use hybrid model to achieve better performance than MPI alone
- Methodology:
  - assign one subdomain to one MPI process
  - use OpenMP with in a subdomain that gets mapped to a node (with 2 or more processors)
- Advantage
  - take advantage of shared memory programming within a subdomain
  - results in bigger subdomains as more than one thread can work on a subdomain as compared to pure MPI case

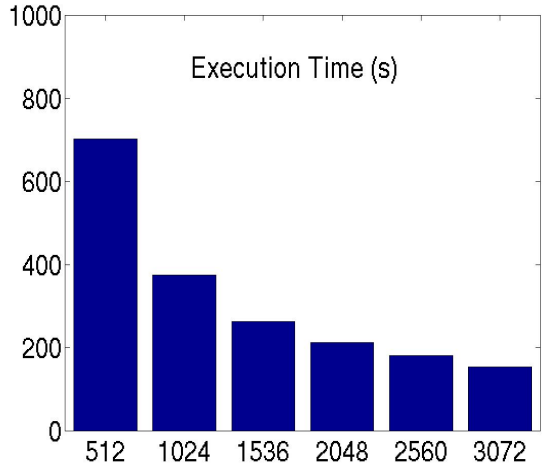
Description of PETSc-FUN3D

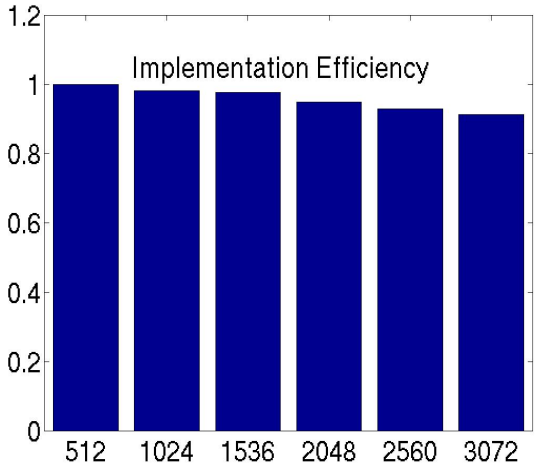
- PETSc-FUN3D is the result of porting FUN3D (developed by **W. K. Anderson, NASA Langley**) to **PETSc** toolkit
- Tetrahedral vertex-centered unstructured grid code for incompressible and compressible Euler and Navier-Stokes equations
- 1st- or 2nd-order Roe for convection and Galerkin for diffusion, and false time stepping with backward Euler for nonlinear continuation towards steady state
- Newton-Krylov-Schwarz (fully implicit, matrix free) solver; the timestep is advanced towards infinity by the switched evolution/relaxation (**SER**) of Van Leer and Mulder
- The preconditioner (incomplete LU with zero and one level fill) in each domain is derived from from 1st-order accurate jacobian

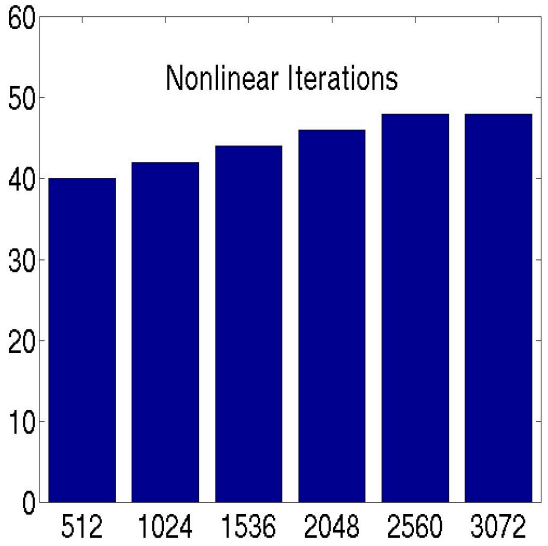
Parallel Scaling Results on ASCI Red

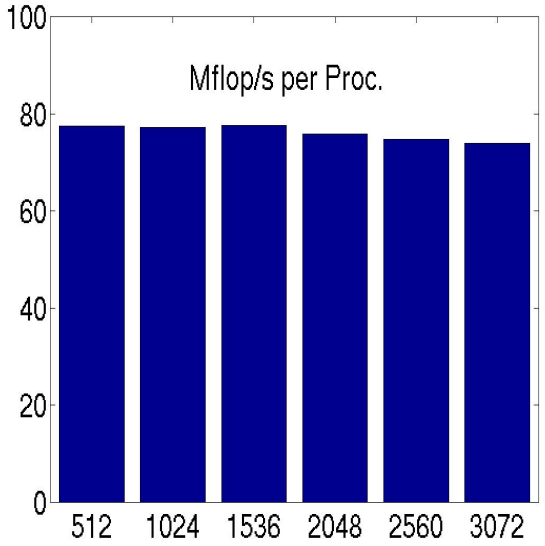
ONERA M6 Wing Test Case, Tetrahedral grid of 2.8 million vertices (about 11 million unknowns) on up to 3072 ASCI Red Nodes (each with dual Pentium Pro 333 MHz processors)

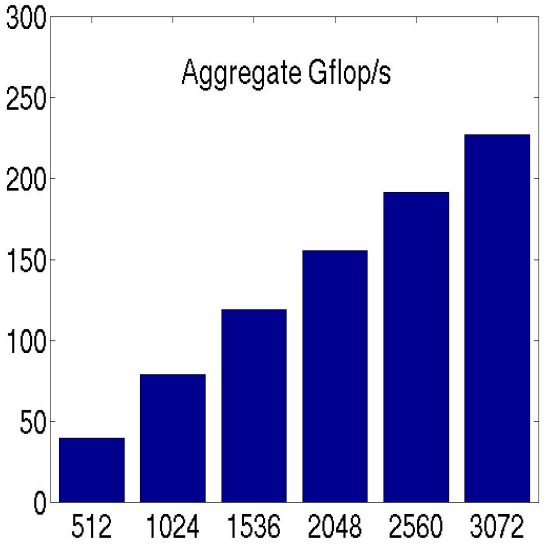












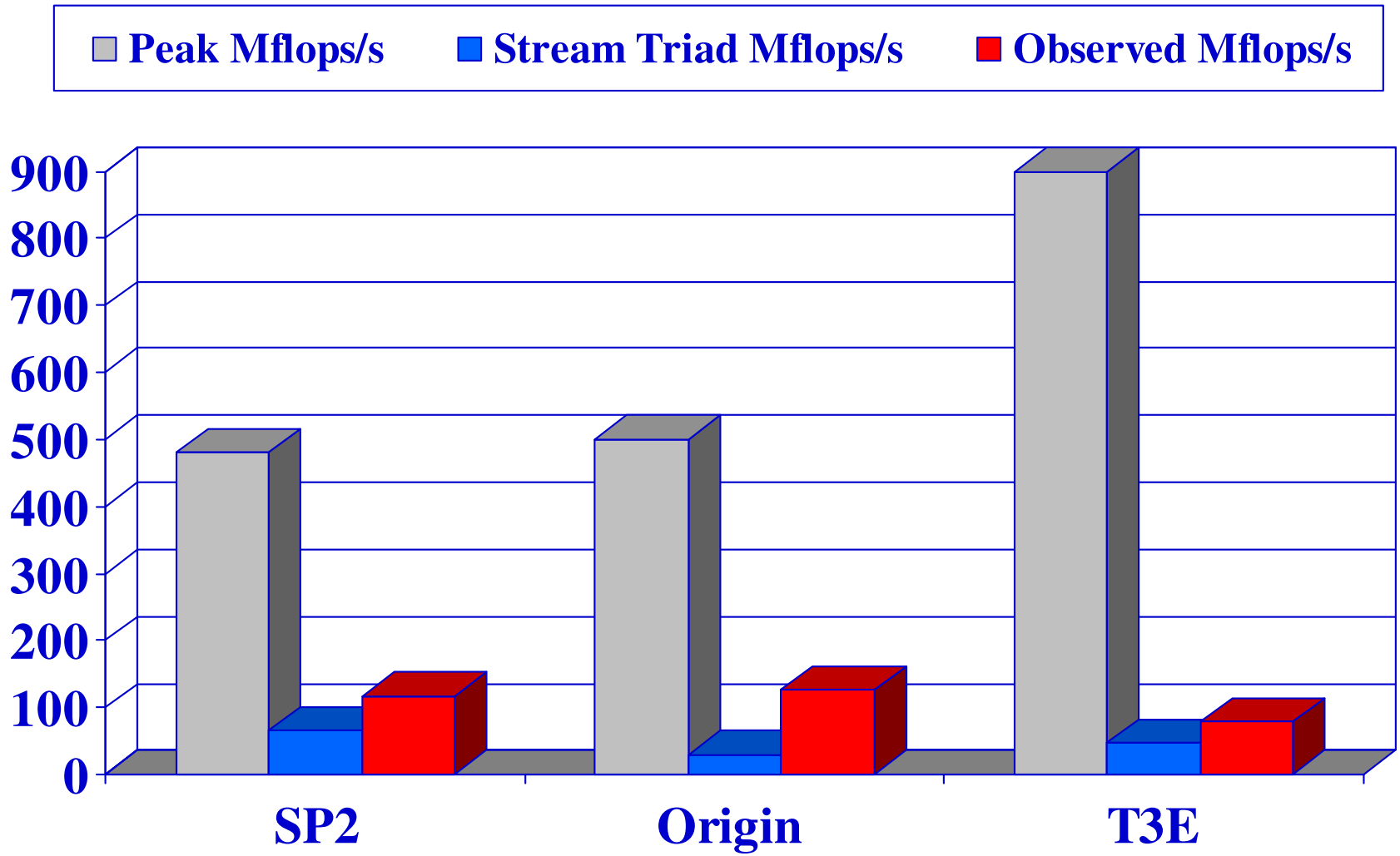
Observations on ASCI Red Scalability

- Sequential performance on many machines is a low percentage of peak
- Per-processor performance on ASCI Red stays fairly constant while going from 128 to 3024 nodes
- The programmer can do a good job in expressing the coarse-grained concurrency but getting good cache locality is a big challenge (especially for unstructured PDE solvers)
- Getting good per processor (or per node in SMP case) performance is the key to achieving good parallel performance

Memory Performance - A Limitation

- Memory performance improvement rate (7% per year) is far behind the CPU performance growth (about 55% per year)
- The performance of many scientific computing codes is limited by the available memory bandwidth
- In shared memory programming, when the processors compete for the memory bandwidth, there is added motivation to reduce the number of memory transactions (necessary or artificial).
- Memory performance models can be very helpful in understanding the observed performance of a code

Sequential Performance of PETSc-FUN3D



Sparse Matrix Vector Algorithm: A General Form

```
for every row, i {
   fetch ia(i+1)
   for j = ia(i) to ia(i + 1)  {     // loop over the non-zeros of the row
        fetch ja(j), a(j), x1(ja(j)), .....xN(ja(j))
        do N fmadd (floating multiply add)
    }
    Store y1(i) .....yN(i)
}
```

Estimating the Memory Bandwidth Limitation for Matrix-Vector Kernel

- Number of Floating-Point Multiply Add (fmadd) Ops = N*nz
- For square matrices,

$$\text{Bytes transferred/fmadd} = (16 + \frac{4}{N})*\frac{n}{nz} + \frac{12}{N}$$

(Since nz >> n, Bytes transferred / fmadd ~12/N)

- Similarly, for **Block AIJ** (**BAIJ**) format

$$\text{Bytes transferred/fmadd} = (16 + \frac{4}{N*b})*\frac{n}{nz} + (\frac{4}{N*b} + \frac{8}{N})$$

Performance Summary of MatMultVec on 250 MHz R10000

- Matrix size, n = 90,708; number of nonzero entries, nz = 5,047,120
- Number of Vectors, N = 1, and 4

Format	Number of Vectors	Bytes / fmadd	Bandwidth		MFlops	
			Required	Achieved	Ideal	Achieved
AIJ	1	12.36	3090	276	58	45
AIJ	4*					45
AIJ	4	3.31	827	221	216	120
BAIJ	1	9.31	2327		84	55
BAIJ	4*					55
BAIJ	4	2.54	635	229	305	175

(*here the vectors are multiplied with the matrix one by one i.e. matrix has been streamed 4 times. Also note that the observed MFlops for N =1 and this case are the same, as they should be).

Lower Precision Storage:

A Way to Reduce the Required Memory Bandwidth

Execution times on a 250MHz Origin 2000 for 358K-vertex case with single or double precision versions of the preconditioner matrix.

Number of Processors	Computational Phase			
	Linear Solve		Overall	
	Double	Single	Double	Single
16	223s	136s	746s	657s
32	117s	67s	373s	331s
64	60s	34s	205s	181s
120	31s	16s	122s	106s

OpenMP Programming Model

- Natural for SMPs
- Simpler to program
- Allows incremental parallelization
- Possibility of lower latency communication
- Portable code

Performance Issues for OpenMP

- Overhead of thread management
- Redundant storage and work
- Sequential reduction phase, which tend to be memory bandwidth bound
- Simplicity goes away when user takes care of memory updates (similar to MPI model)

Overhead of Managing the Threads

- Depends on
  - the number of times parallel regions are entered
  - the number and size of private data structures
  - the cost of the synchronization mechanism
- Some implementations are not able to handle the single threaded case well:
  - For example, on Origin 2000 (250 MHz R10000) sample program Jacobi (from OpenMP web site) takes
    - 81 seconds when run on single OpenMP thread
    - 62 seconds when compiled without OpenMP
    - scales w.r.t. the former

Competing for the Available Memory Bandwidth

- The processors on a node compete for the available memory bandwidth
- The computational phases that are memory bandwidth limited will not scale
  - may even run slower because of the extra synchronizations

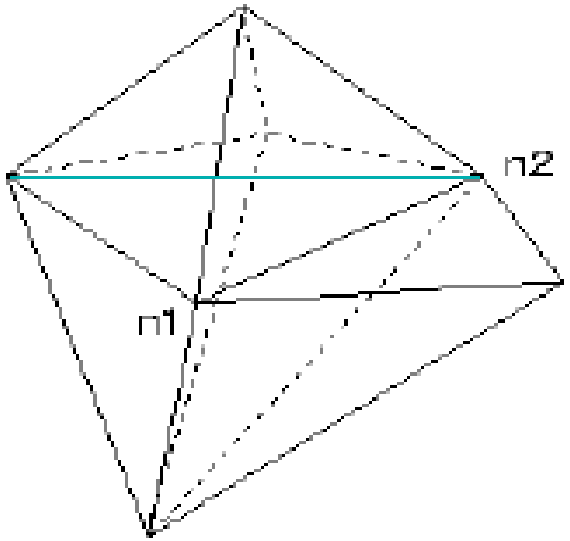
Stream Benchmark on ASCI Red MB/s for the Triad Operation

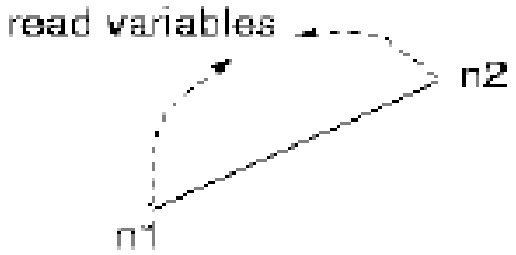
Vector Size	1 Thread	2 Threads
1E04	666	1296
5E04	137	238
1E05	140	144
1E06	145	141
1E07	157	152

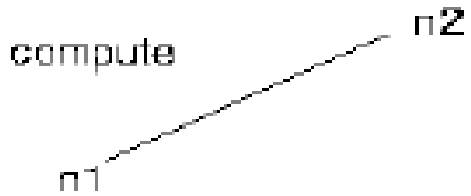
Redundant Storage and Work

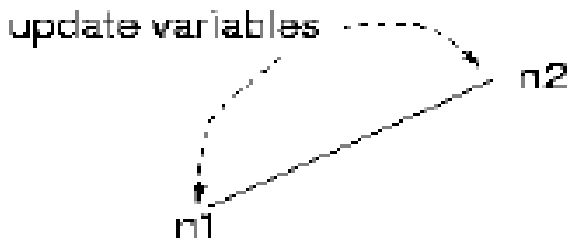
- To manage memory updates efficiently, we might need to create extra private work arrays
- These work arrays need to be copied into a shared array at the end of the parallel region
  - A memory bandwidth limited sequential phase
- A vector reduce operation is needed in the OpenMP standard

Flux Evaluation in PETSc-FUN3D









Variables at each node:
- density,
- momentum (*x,y,z*),
- energy,
- pressure

Variables at edge:
- identity of nodes,
- orientation(*x,y,z*)

Example of Redundant Work

```
!$omp parallel firstprivate(res) shared(resvec)
! Compute private residual updates
! Combine the private residual updates
!$omp critical (residual_update)
#if defined(_OPENMP)
       do n = 1,nnodes
          resvec(1,n)=resvec(1,n)+res(1,n)
          resvec(2,n)=resvec(2,n)+res(2,n)
          resvec(3,n)=resvec(3,n)+res(3,n)
          resvec(4,n)=resvec(4,n)+res(4,n)
       enddo
       flops = flops + 4*nnodes
#endif
!$omp end critical (residual_update)
!$omp end parallel
```

Apply the "Owner Compute" Rule for OpenMP

- Create the disjoint working sets to eliminate the redundant private arrays (e.g. by coloring the edges and nodes)
- Alternatively, use OpenMP over subdomains
  - each MPI process will repartition its domain
  - each thread will work on its assigned subdomain
- Brings in the complexity of programming as the user is taking care of the memory updates

MPI/OpenMP in PETSc-FUN3D

- Only in the flux evaluation phase as it is not memory bandwidth bound
- Gives the best execution time as the number of nodes increases because the subdomains are chunkier as compared to pure MPI case

Nodes	MPI/OpenMP		MPI	
	1 Thr	2 Thr	1 Proc	2 Proc
256	483s	261s	456s	258s
2560	76s	39s	72s	45s
3072	66s	33s	62s	40s

Conclusions

- Memory performance is crucial to achieve good per processor performance
- The hybrid MPI/OpenMP achieves good overall performance but should be used only in the phases that are not memory bandwidth limited
- Even for the OpenMP case, memory updates need to be managed by user (following the “owner computes” rule)
- There is a need for the vector reduce operation in the OpenMP standard

Future Directions

- Tolerate the load imbalance by using OpenMP over subdomains and assigning the work dynamically to threads
- Study the implementation level techniques like loop structure, array layout, array padding (to avoid false sharing) etc. for the hybrid model

Relevant URLs

- PETSc-FUN3D Project at Argonne

  http://www.mcs.anl.gov/petsc-fun3d
- PETSc

  http://www.mcs.anl.gov/petsc
- ODU NSF and ASCI projects

  http://www.math.odu.edu/~keyes/nsf

  http://www.math.odu.edu/~keyes/asci