Parallelization of the NIMROD Code

Authors: Alphabetical by Institution

A. Koniges, A. Tarditi, X. Xueqiao, Lawrence Livermore Nat. Lab,

D. Barnes, Los Alamos National Lab,

S. Plimpton, Sandia Albuquerque,

+ the NIMROD Team,

+ISIS-code Team, Sandia Livermore

+CRAY RESEARCH and Univ. Texas Collaborations

38th Annual Meeting, APS-DPP, 11 - 15 Nov. 1996, Denver

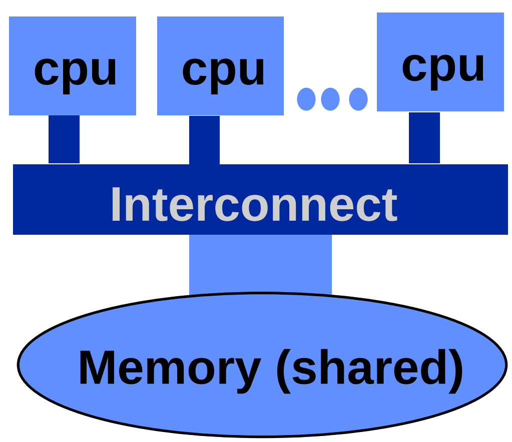
Jointly supported by DOE-OFE and DOE-MICS

OUTLINE

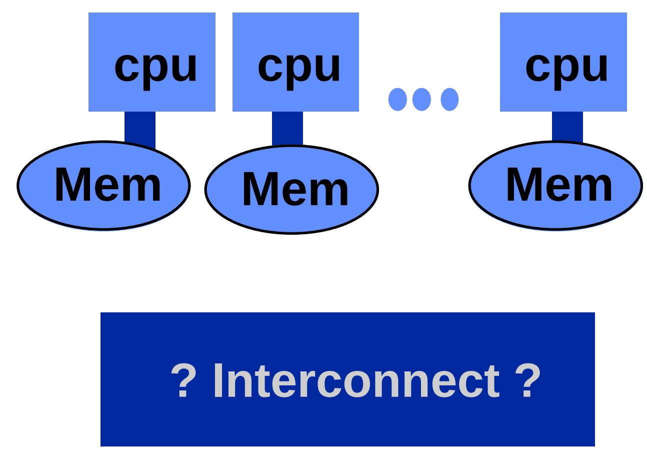
- Summary of Parallel Concepts and Architectures
- Highest Level Nimrod Parallelization
- Solver Level Parallelization
- First results
  - CRAY Research T3E at U. Texas
  - CRAY Research T3D at LLNL

Review: What is MPP?

SMP Parallel Processor



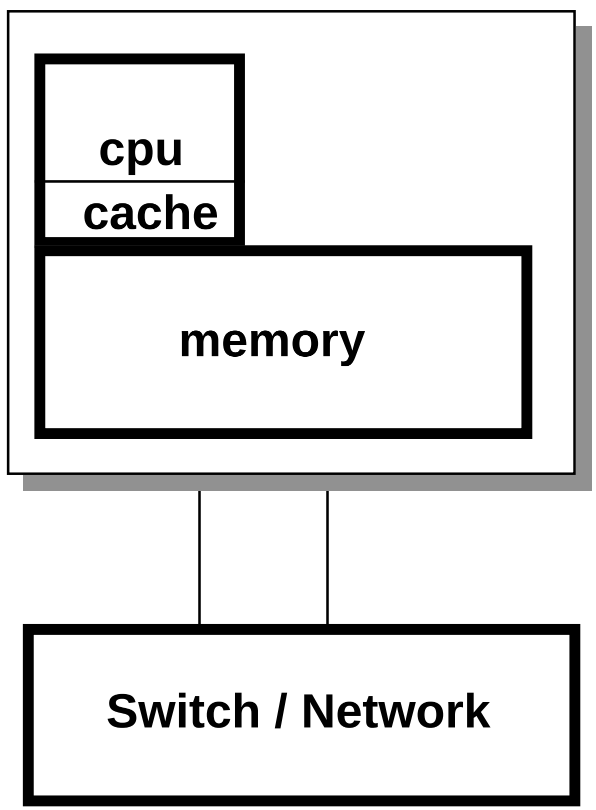
MPP Massively Parallel Processor



2-32+ processors today
Memory shared
High powered Processors (C90, SMP's)

128 - up today
Memory physically distributed
High Powered Micros (e.g. Alpha)

MPP Today



Processor

RS6000 - IBM
Dec Alpha - Cray
Sparc - TMC, Meiko

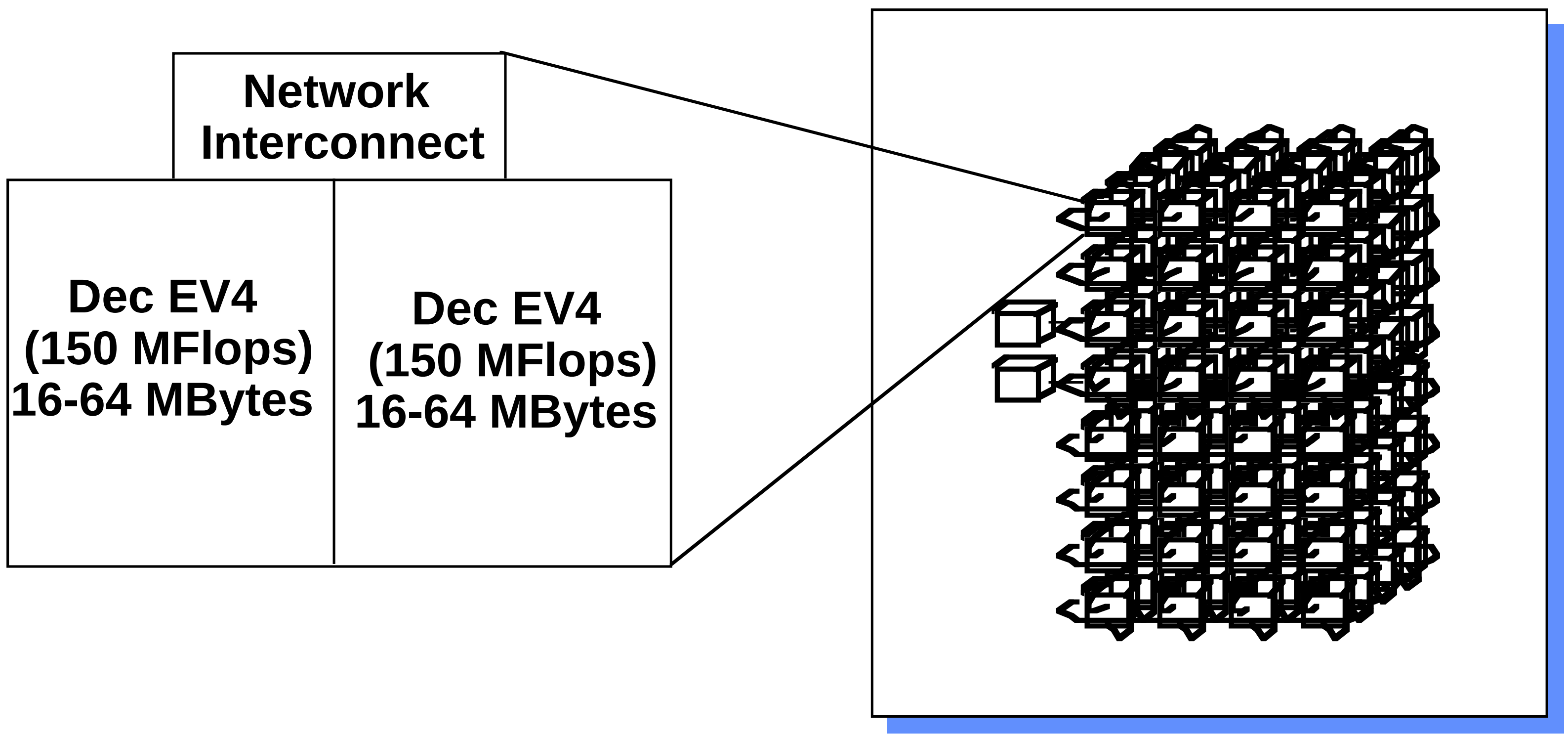
Memory

16 MBytes - 256 MBytes
Ram per processor

Network

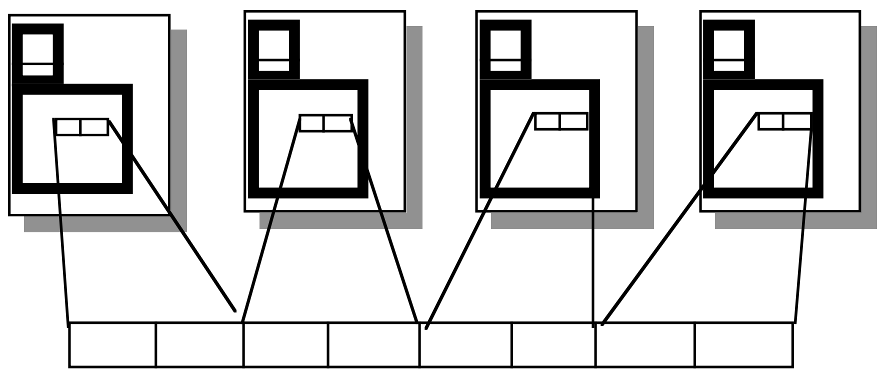
omega	IBM SP2, Meiko
3D Torus	Cray T3D,E

Cray T3D

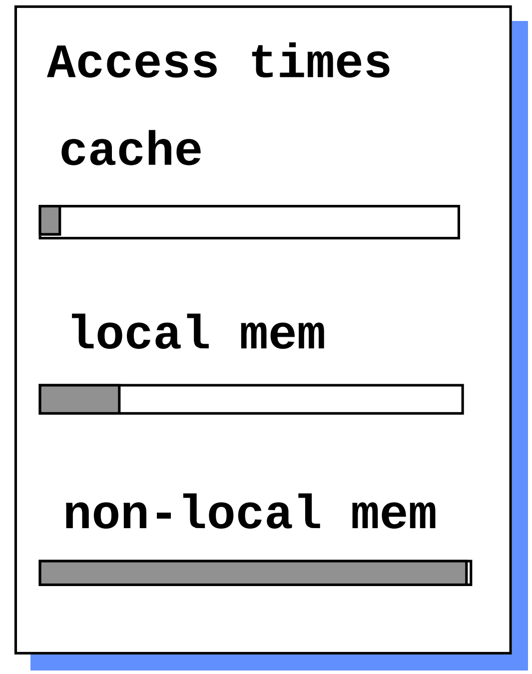


http://www.cray.com/PUBLIC/product-info/mpp/CRAY_T3D.html

Message Passing



Proc 1	Proc 1
...	...
send A[25]	receive A[25]
...	...



Each Processing Element (PE) owns part of the data.
Other PE's must generate messages to access memory.

Steps in Parallel Code Development

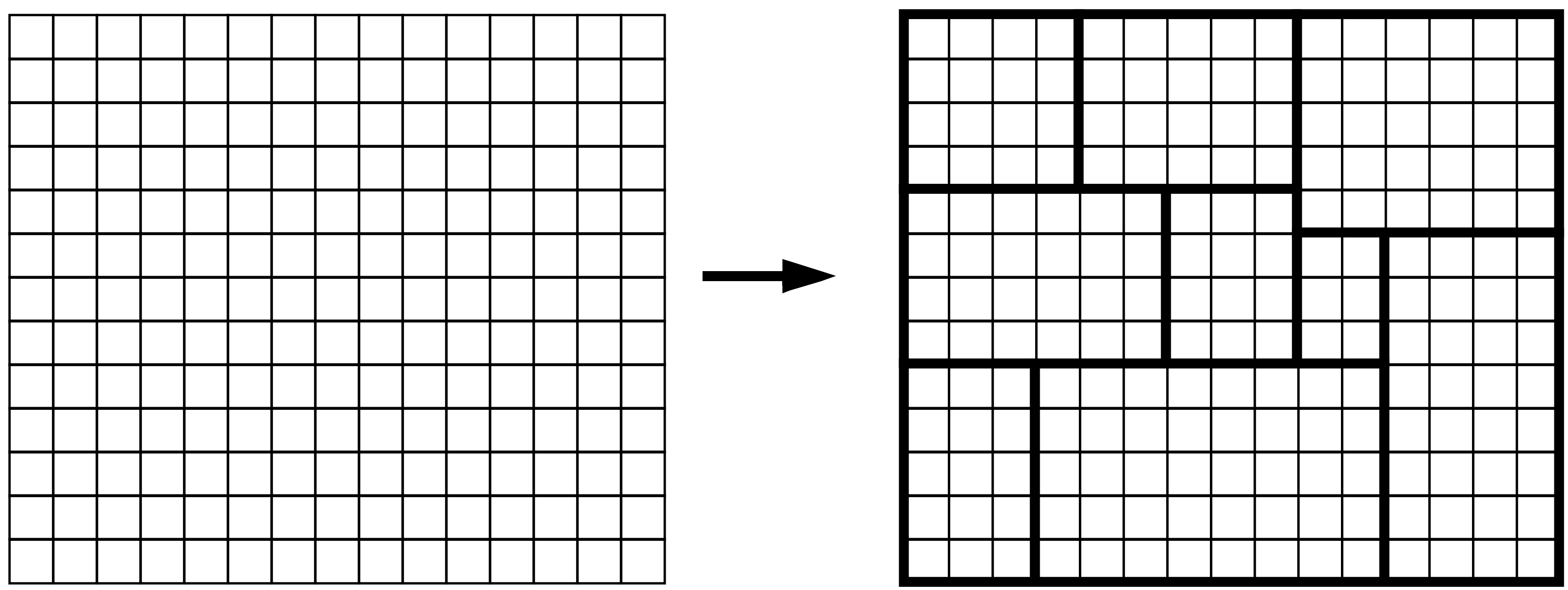
- Code design to avoid bottle necks
  - Block domain decomposition of 2D toroidal mesh
  - Blocks seam together
  - Blocks or Multiple blocks assigned to processors
  - FFT’s in third dimension restricted to block
  - Communication between blocks via Message Passing Interface (MPI)
  - Single Processor Optimization
  - Use optimized libraries for compute intensive pieces
  - Optimize use of cache
- Multiple Processor Optimization
  - overlap communication and computation
- Iterative Solver Design Issues

NIMROD Coding Choices

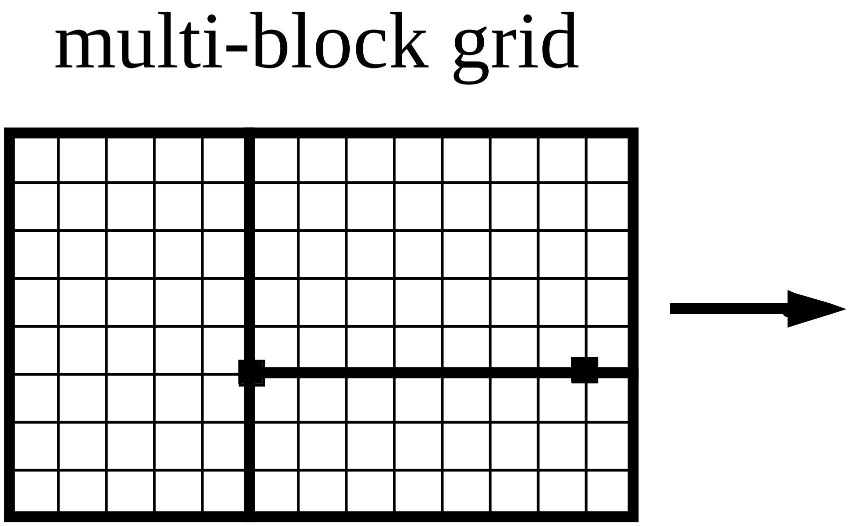
- Message-passing parallelism with F90/MP
- F90 provides dynamic memory, rich data structures
- MPI provides portability to any machine with a single-processor F90 compiler
- MPI allows irregular, asynchronous communication
- Same code will run on workstation, Cray C90, or parallel platforms:
  - Cray T3D/E
  - IBM SP2
  - Workstation Clusters

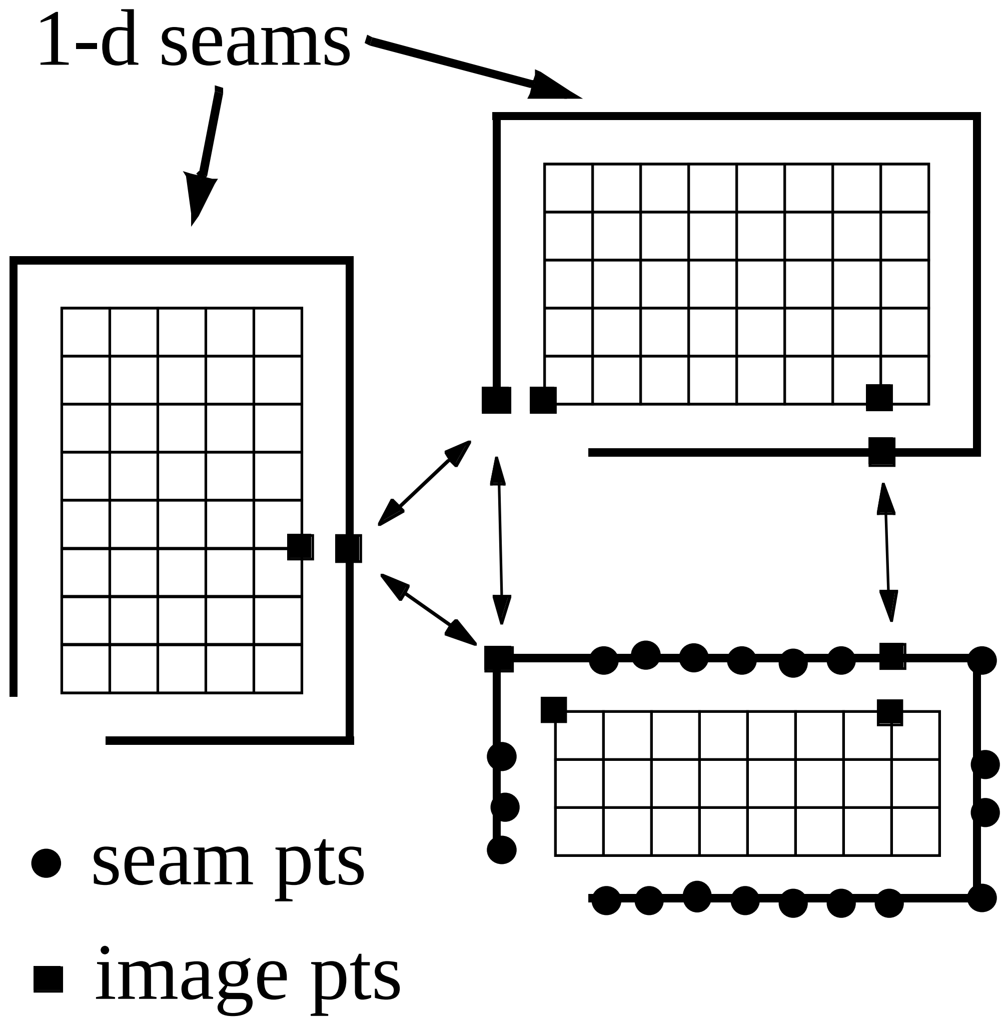
Grid Structure of NIMROD

- NIMROD grid is a general collection of joined sub-blocks mapped to the poloidal plane.
- Edge points of adjacent blocks join exactly.



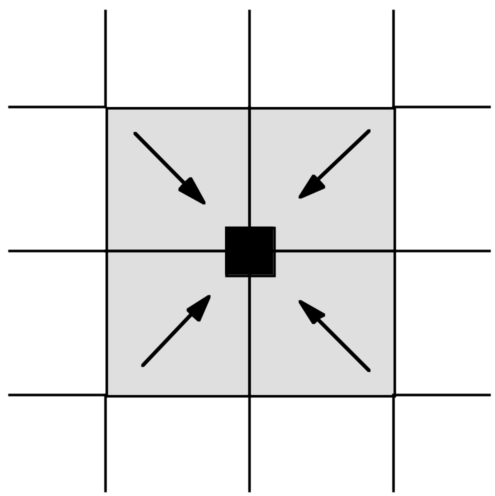
Sub-blocking with associated seams



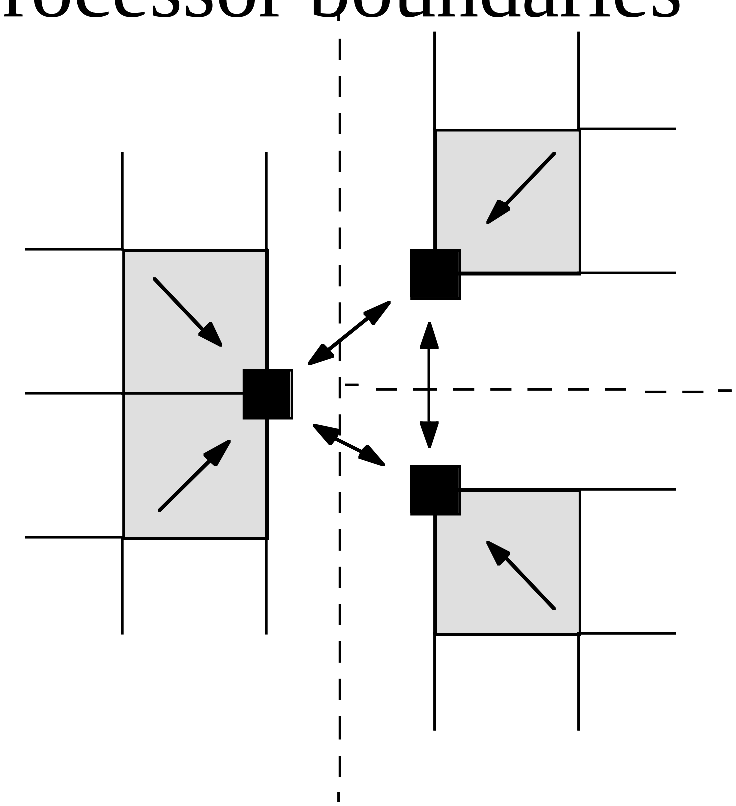


Each edge point has "image" points in other blocks/seams.

FE integration stencil for block interior and across block and/or processor boundaries



Interior



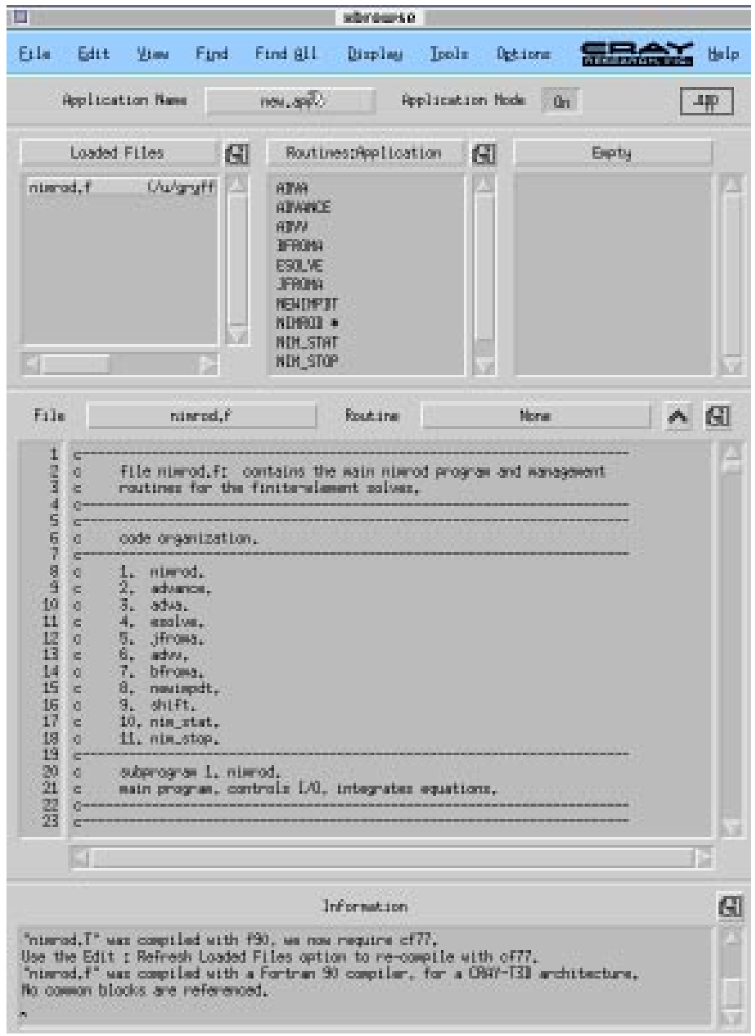
Across boundaries

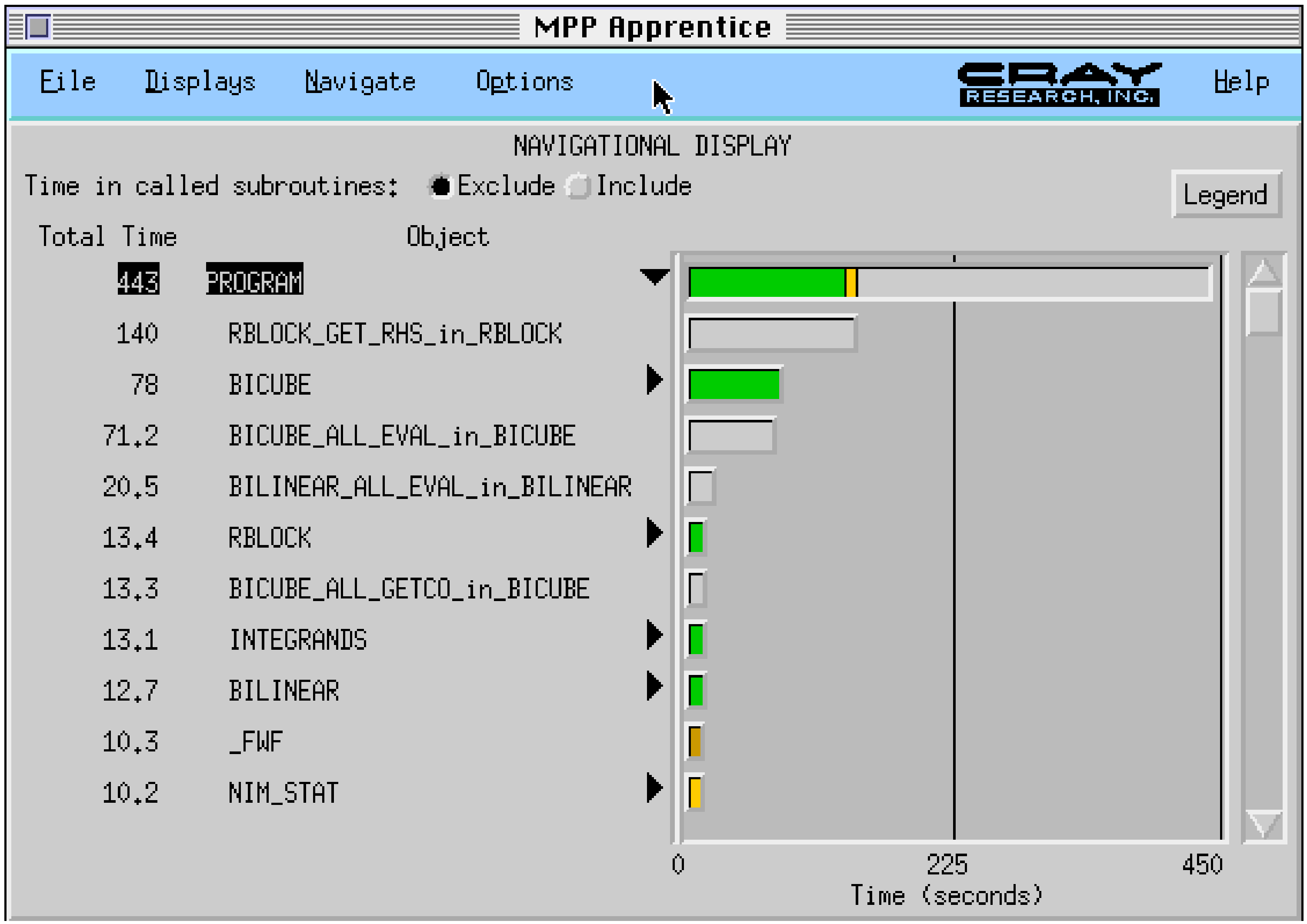
- If 2 adjacent blocks are on different processors, a data exchange is needed to complete the integration.

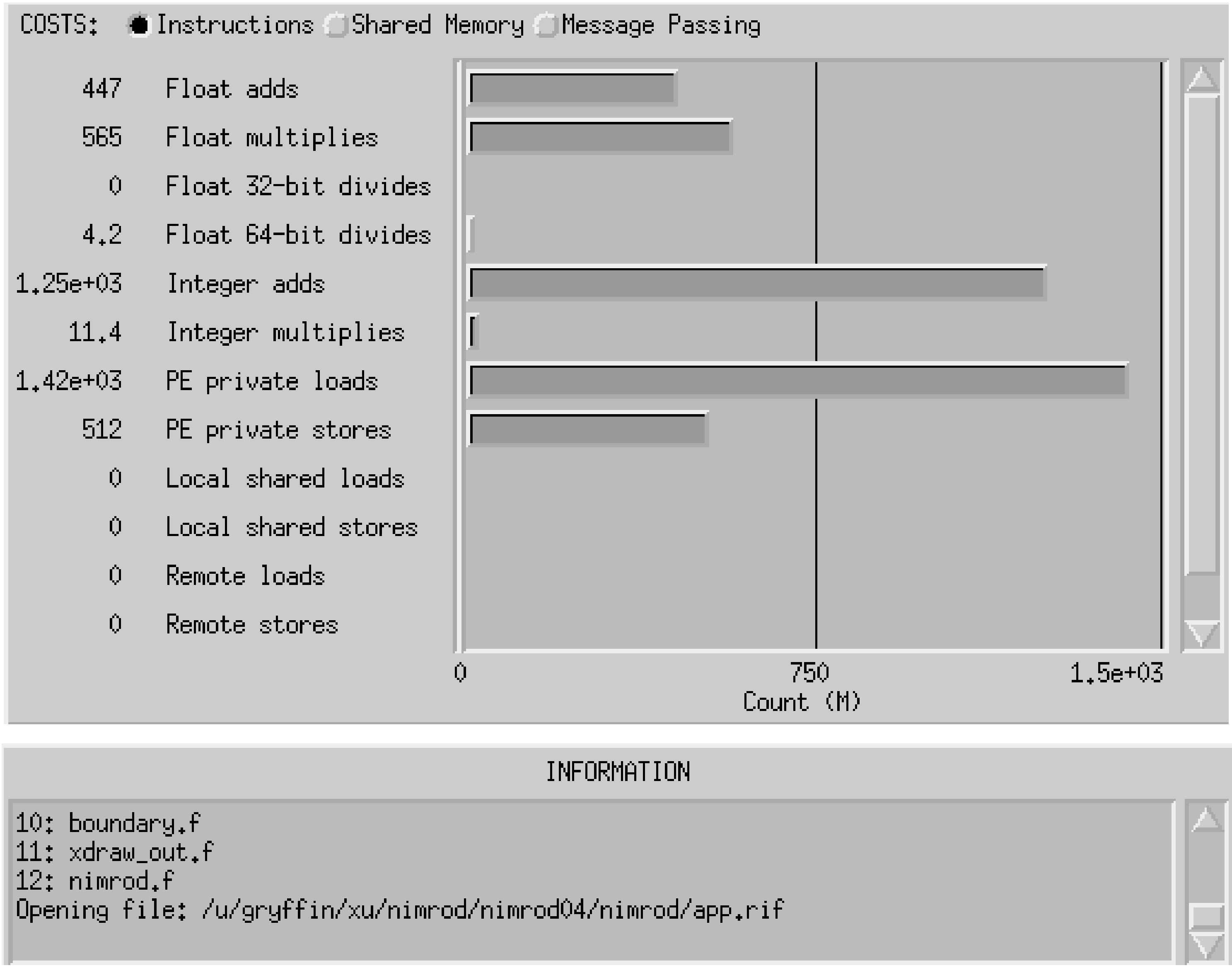
Inherent parallelism in NIMROD

- Each processor owns 1 or more “blocks” and their associated “seams”.
- Computations can be done on each block independently.
- Only connection/communication with other processors is via “seams”

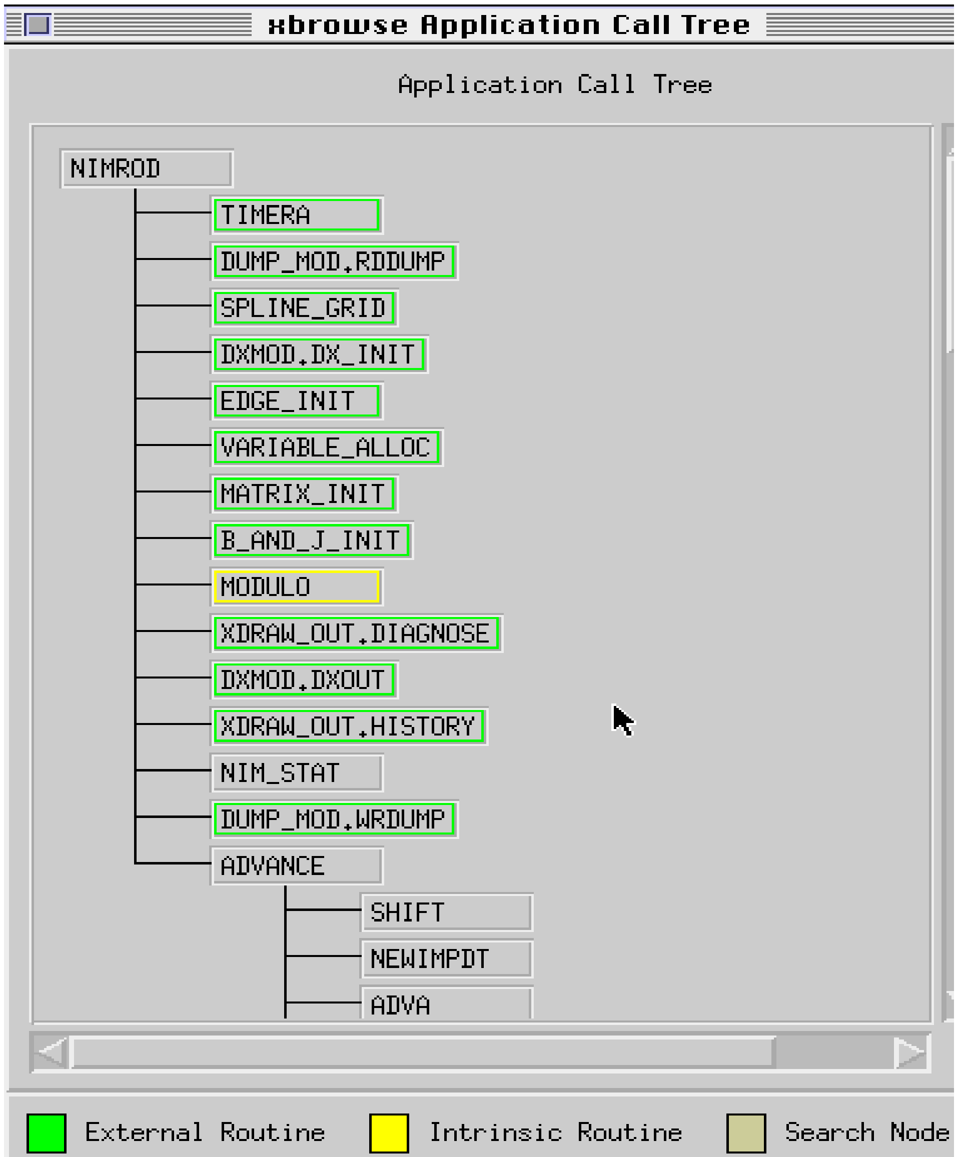
Parallel Tools are used to analyze performance

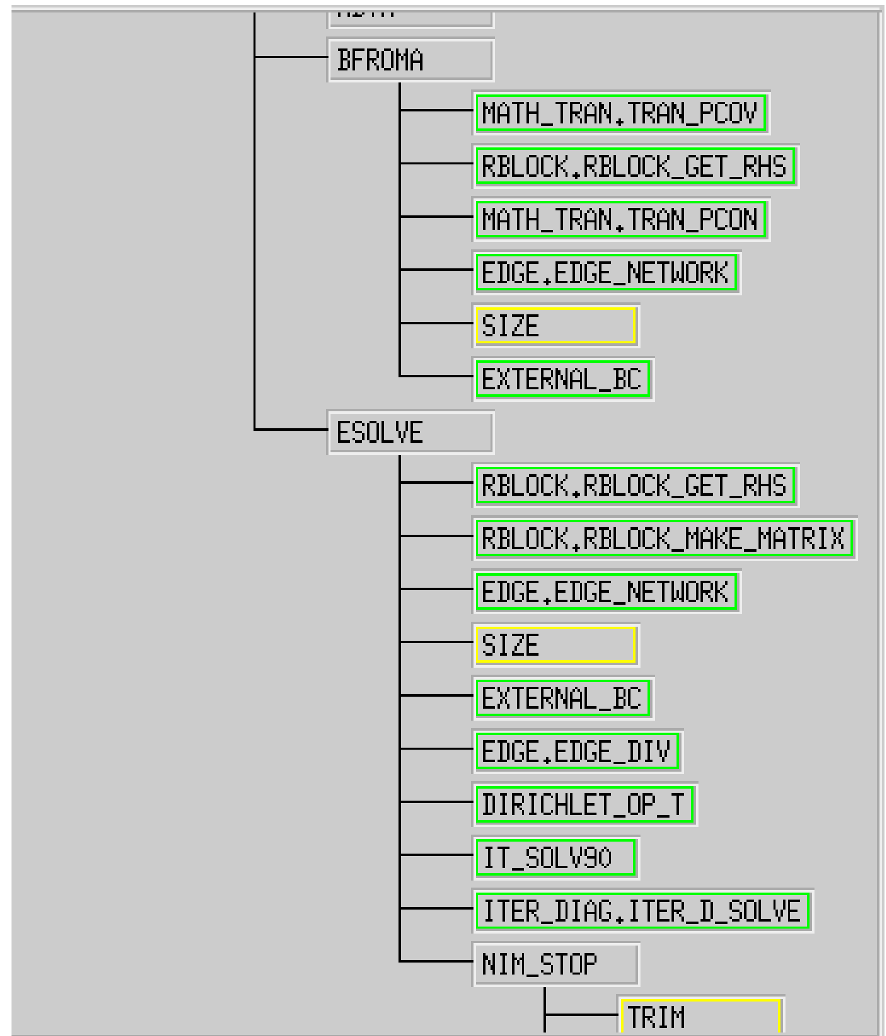


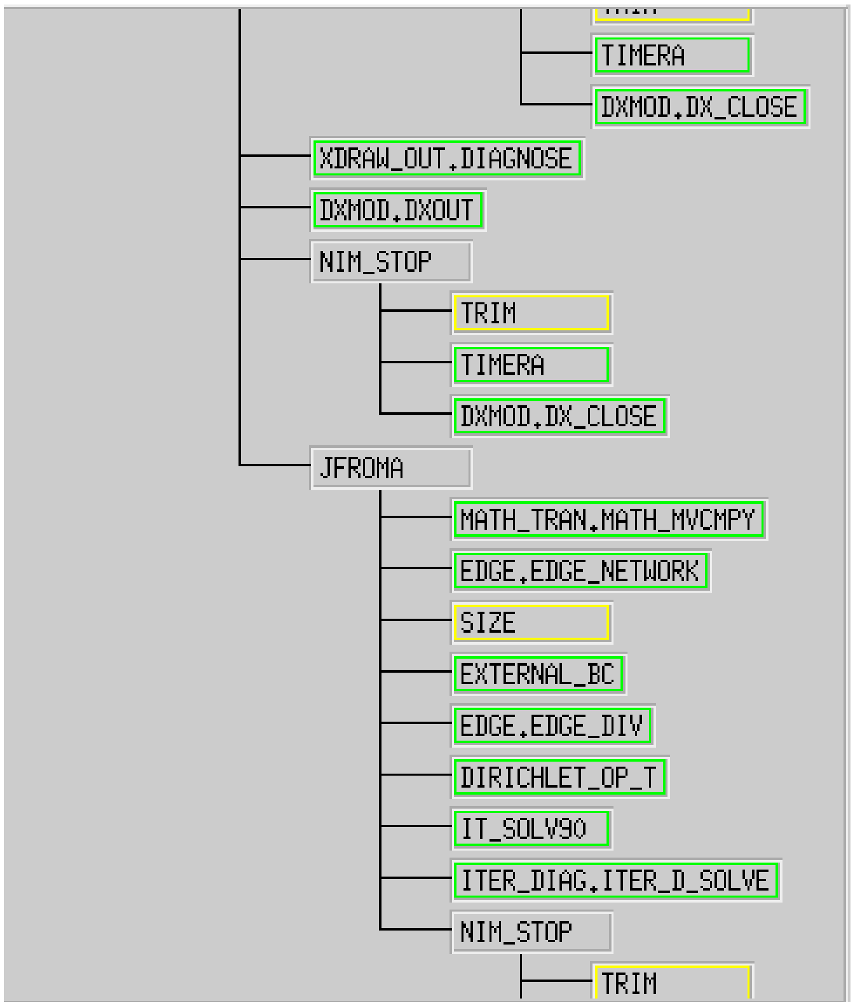

MPP Apprentice
File
Displays
Navigate
Options
CRAY RESEARCH, INC.
Help
NAVIGATIONAL DISPLAY
Time in called subroutines: Exclude Include
Legend
Total Time
Object
443 PROGRAM
140 RBLOCK_GET_RHS_in_RBLOCK
78 BICUBE
71.2 BICUBE_ALL_EVAL_in_BICUBE
20.5 BILINEAR_ALL_EVAL_in_BILINEAR
13.4 RBLOCK
13.3 BICUBE_ALL_GETCO_in_BICUBE
13.1 INTEGRANDS
12.7 BILINEAR
10.3 _FWF
10.2 NIM_STAT
0
225
450
Time (seconds)

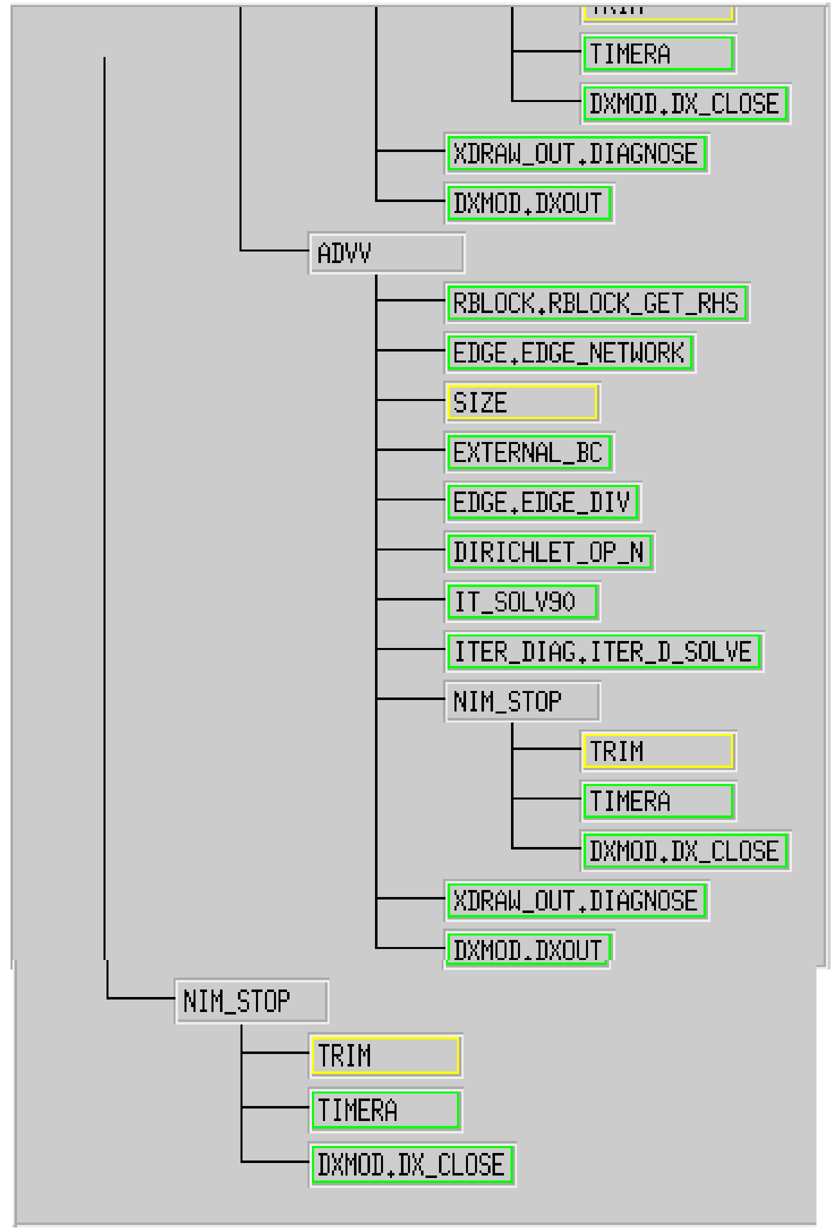

COSTS: Instructions Shared Memory Message Passing
447 Float adds
565 Float multiplies
0 Float 32-bit divides
4.2 Float 64-bit divides
1.25e+03 Integer adds
11.4 Integer multiplies
1.42e+03 PE private loads
512 PE private stores
0 Local shared loads
0 Local shared stores
0 Remote loads
0 Remote stores
0
750
1.5e+03
Count (M)
INFORMATION
10: boundary.f
11: xdraw_out.f
12: nimrod.f
Opening file: /u/gryffin/xu/nimrod/nimrod04/nimrod/app.rif

Performance Analysis with Apprentice: Browse






TIMERA
DXMOD.DX_CLOSE
XDRAW_OUT.DIAGNOSE
DXMOD.DXOUT
NIM_STOP
TRIM
TIMERA
DXMOD.DX_CLOSE
JFROMA
MATH_TRAN.MATH_MVCMPY
EDGE.EDGE_NETWORK
SIZE
EXTERNAL_BC
EDGE.EDGE_DIV
DIRICHLET_OP_T
IT_SOLV90
ITER_DIAG.ITER_D_SOLVE
NIM_STOP
TRIM


TIMERA
DXMOD.DX_CLOSE
XDRAW_OUT.DIAGNOSE
DXMOD.DXOUT
ADVV
RBLOCK.RBLOCK_GET_RHS
EDGE.EDGE_NETWORK
SIZE
EXTERNAL_BC
EDGE.EDGE_DIV
DIRICHLET_OP_N
IT_SOLV90
ITER_DIAG.ITER_D_SOLVE
NIM_STOP
TRIM
TIMERA
DXMOD.DX_CLOSE
XDRAW_OUT.DIAGNOSE
DXMOD.DXOUT
NIM_STOP
TRIM
TIMERA
DXMOD.DX_CLOSE

Performance Analysis is Beginning with Apprentice

- Compiling, loading and executing with new/old languages on MPP architectures is difficult
- Apprentice is being used to identify problem areas
- Exclude option is used to isolate subroutines
- Subroutines are sorted by seconds (Amdahl's Law)
- GFlop rate can be computed only after flops in libraries are defined

Single Processor Optimization is in progress

Code Performance:

1 Total processors (PEs) allocated to this application

2.28 x 10^6 Floating point operations per second (for 1 PEs)
2.83 x 10^6 Integer operations per second (for 1 PEs)

2.28 x 10^6 Floating point operations per second per processor
2.83 x 10^6 Integer operations per second per processor

3.20 x 10^6 Private loads per second per processor
1.15 x 10^6 Private stores per second per processor
0.00 x 10^6 Local shared loads per second per processor
0.00 x 10^6 Local shared stores per second per processor
0.00 x 10^6 Remote loads per second per processor
0.00 x 10^6 Remote stores per second per processor

3.03 x 10^6 Other instructions per second per processor
12.50 x 10^6 Instructions per second per processor

0.71 Floating point operations per load
0.89 Integer operations per load

Time spent performing different task types:

```
 56 sec (12.61%) executing "work" instructions
 77 sec (17.31%) loading instruction and data caches
  0 sec ( 0.00%) waiting on shared memory operations
  0 sec ( 0.00%) waiting on PVM communication
  0 sec ( 0.01%) executing "read" or other input operations
 10 sec ( 2.35%) executing "write" or other output operations
301 sec (67.72%) executing uninstrumented functions
        --------
        100.00%  Total
```

Detailed Description of Single PE Performance on T3D

The combined losses due to single instruction issue, instruction cache and data cache activity are estimated to be 77 sec, or 17.31% of the measured time for this program.

The combined expenditure of time for output routines is measured to be 10 sec, or 2.35% of the measured time for this program.

The combined expenditure of time for input routines is measured to be 0 sec, or 0.01% of the measured time for this program.

The sections of code, below the current selection, with the largest amount of total time including subordinate code and called routines, are NIMROD, ADVANCE, RBLOCK.

The sections of code, below the current selection, with the largest amount of total time excluding subordinate code and including called routines, are ADVANCE@128, RBLOCK, INTEGRANDS.

Parallel Additions to Serial NIMROD

- Assignment of blocks to processors (load-balancing)
- Setup of data structures for parallel seaming.
- Knit seams between blocks.
  - used in explicit timestepper
  - used in matrix-vector multiply of CG-solver
- Dot-products for CG-solver
- I/O

Serial Seam Connection

1) Copy from block-edge grid points to seams

2) Loop over images of each seam point, sum image values to block-edge grid points

3) Apply external boundary conditions.

Parallel Seam Connection

1) Send my seam data to neighboring processors.
2) For seam points where I own both image pairs, sum image values to my block-edge grid points.
3) Receive incoming image data from other processors sum it to my block-edge grid points.
4) Apply external boundary conditions.
5) Copy from my block-edge grid points to my seams.

Attributes of Parallel Seam Connection routine

- Uses asynchronous communication in irregular pattern of connectivity between processors.
- Overlaps communication and computation (steps 2-4).
- Pre-computes data structures to optimally pack/unpack messages being exchanged with other processors.
- Fast !
  - Seam communication is only small fraction of block computation time.

Results

- Parallel performance of seam-connection kernel
- Parallel performance of explicit timestepper
- Parallel performance of CG solver using diagonal pre-conditioning

Timing Results for Parallel Seam Connection on T3E

- 1.02 million grid cells, 174 blocks, 51200 seam points, 3 values/grid-cell
- CPU seconds for 1 seam-operation:

Procs	1	2	5	10	20	30
Time	0.64	0.25	0.12	0.081	0.033	0.024

- Scales roughly linearly with size of grid and number of processors

Timing Results for Explicit Nimrod T3D Calculation

- CPU seconds for 100 timesteps on the T3D shows excellent scalability as problem size increases.

Blocks/Cell	1 PE	2 PEs	4 PEs	8 PEs	16 PEs	32 PEs
4/400	42.2	19.0	10.3			
16/1600	145.3		38.5		11.0	
64/6400				75.2	53.7	21.0
256/25,600					150.5	75.9
1024/102,400						

Timing Results for Explicit Nimrod T3E Calculation

- CPU seconds for 100 timesteps on the T3E shows excellent scalability as problem size increases.

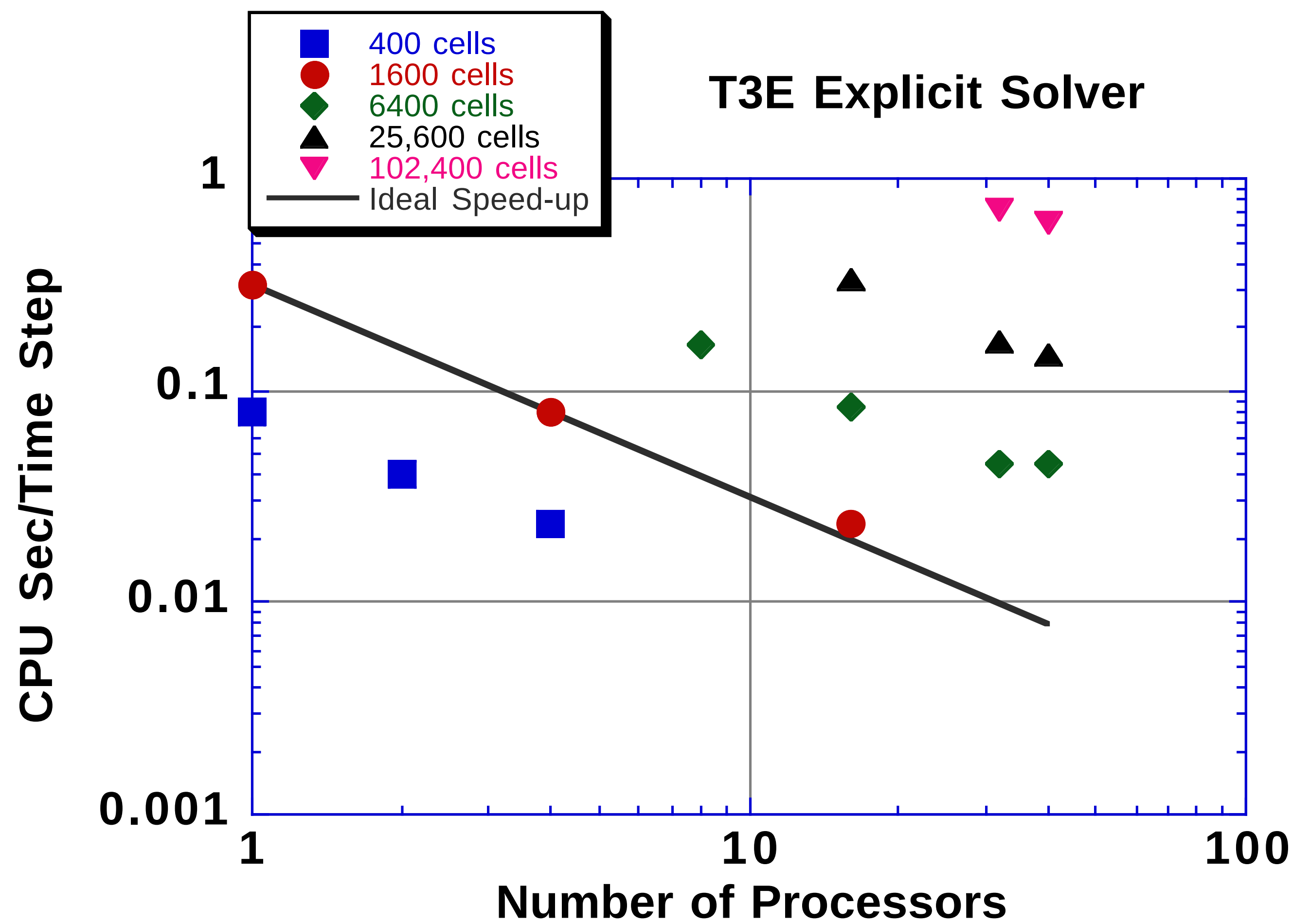
Blocks/Cell	1 PE	2 PEs	4 PEs	8 PEs	16 PEs	32 PEs	40 PEs
4/400	7.85	4.05	2.35				
16/1600	31.7		7.95		2.35		
64/6400				16.5	8.45	4.55	4.55
256/25,600					33.0	17.0	14.8
1024/102,400						71.5	62

Timing Results for Implicit Nimrod T3E Calculation

- CG solver with diagonal preconditioning
- 50 timesteps, roughly 40 CG iterations per step
- Preconditioning methods for CG solver require more study

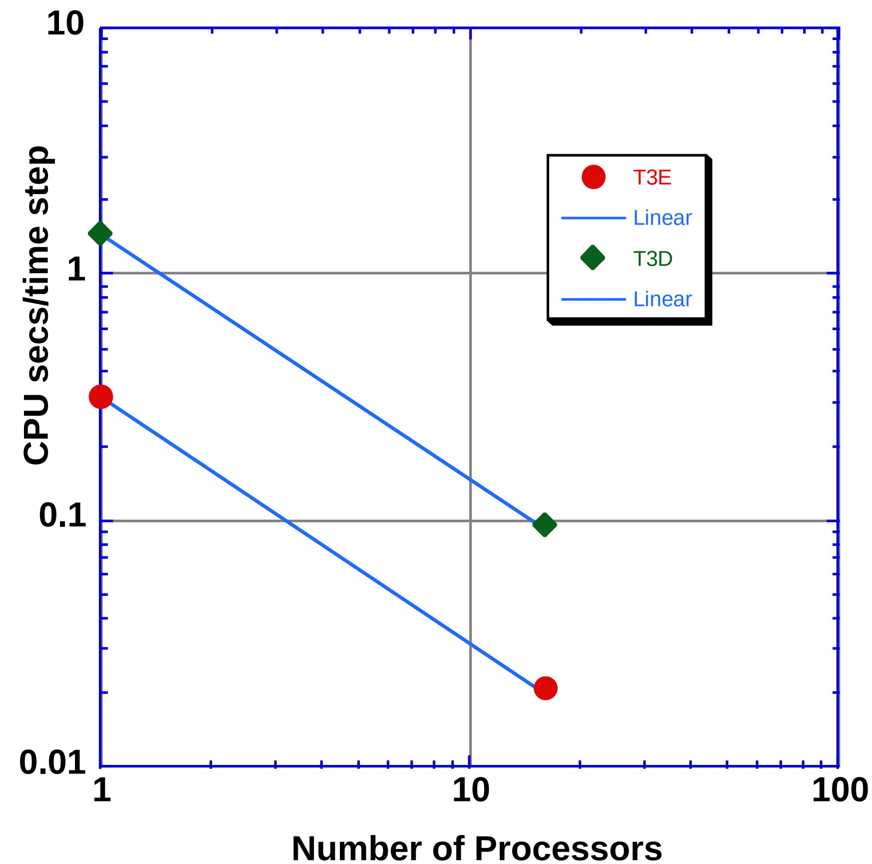
Blocks/Cell	1 PE	2 PEs	4 PEs	8 PEs	16 PEs	32 PEs	40 PEs
4/400	23.5	12.4	6.9				
16/1600	89.7		24.8		7.9		
64/6400				43.3	21.7	12.1	12.6
256/25,600					88.3	44.4	40.0

First Performance Results Show Nearly Ideal Speed-up (even for fixed problem size)



Scaled Speed-up Comparison of T3E/D

- T3E is factor of 4.5 faster
  - 2X processor speed
  - chaining
  - cache effects
- Scalability is virtually linear for both machines



Conclusions

- Blockwise-design of NIMROD enables rapid message-passing parallelization.
- Explicit and diagonal-preconditioned CG solver are running well in parallel
- T3E out-performs T3D, but both perform well
  - (Cache, Processor speed)
- F90: great language
  - terrible compilers in general
  - Good on T3E, but libraries still missing
  - Acceptable on T3D, but performance tools need improvement
  - does it produce fast code ?? (open question)

Future Work

- Implement 2nd NIMROD CG solver (block-invert preconditioner) in parallel
- Test convergence and performance of solvers as a function of number-of-blocks, number-of-processors, physics being solved
- Try new iterative solvers
- Optimize code performance

Special Acknowledgements

- DOE MICS Office supported SNL work on solvers and parallelization for the NIMROD project
- The High Performance Computing Facility at UT Austin provided computer time on their 40-processor Cray T3E
- The Institutional Computing Facility at LLNL provide computer time on their 256-processor Cray T3D