Parallelization of the NIMROD Code

Jointly supported by DOE-OF and DOE-MICS

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

Authors: Alphabetical by Institution

A. Koniges, A. Tarditi, X. Xueqiao, Lawrence Livermore
S. Plimpton, Sandia Albuquerque,
D. Barnes, Los Alamos National Lab,
A. Koniess, A. Tarditi, X. Xueqiao, Lawrence Livermore Nat. Lab,

Authors: Alphabetical by Institution

Parallelization of the NIMROD Code
OUTLINE

• Summary of Parallel Concepts and Architectures
• Highest Level Nimrod Parallelization
• Solver Level Parallelization
• First Results
• Highpest Level Nimrod Parallelization Architectures
• Summary of Parallel Concepts and Architectures

- CRAY Research T3D at LLNL
- CRAY Research T3E at U. Texas
Review: What is MPP?

MPP: Massively Parallel Processor

High Powered Micros (e.g., Alpha)

Memory physically distributed

1-28 up today

Interconnect

2-32+ processors today

Memory (shared)

High Powered Processors (C90, SMP's)

SMP: Symmetric Multi-Processor

Memory shared

MPP Massively Parallel Processor

Interconnect

cpu

Mem

Mem

Mem

cpu

cpu

cpu

Review: What is MPP?
MPP Today

- Omega
- IBM SP2, Meiko
- 3D Torus
- Cray T3D

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

- Memory:
  - 16 MBytes - 256 MBytes

- Switch / Network
Cray T3D


Dec EV4

16-64 MBytes
(150 MFloops)

Dec EV4

16-64 MBytes
(150 MFloops)

Network Interconnect
Message Passing

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

Access times
- local mem
- cache
- non-local mem

Proc 1

... send A[25] ...

Proc 1

... receive A[25] ...

A[100]
Steps in Parallel Code Development

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

Message-passing parallelism with F90/MP
Grid Structure of NIMROD

- NIMROD grid is a general collection of joined 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.

1-d seams

multi-block grid

Sub-blocking with associated seams
• If 2 adjacent blocks are on different processors, a data exchange is needed to complete the integration.

Across boundaries

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

Interior
Inherent parallelism in NIMROD

• Each processor owns 1 or more "blocks" and their associated "seams".
  Only communication/communication with other independently.

• Computations can be done on each block via "seams".

• Each processor owns at least 1 or more "blocks".
Parallel tools are used to analyze performance.
Performance Analysis with Apprentice Browse
Performance Analysis is Beginning with Apprentice

- Apprentices is being used to identify problem areas.
- Subroutines are sorted by seconds (Amdahl's Law).
- Exclude option is used to isolate subroutines.
- GFlop rate can be computed only after flops in libraries are defined.

Compiling, loading and executing with new/old languages on MPP architectures is difficult.
Code Performance:

- 1 Total processors (PEs) allocated to this application
- Single Processor Optimization is in progress

- 0.71 Floating point operations per load
- 0.89 Integer operations per load
- 2.28 x 10^6 Floating point operations per second per processor
- 2.83 x 10^6 Integer operations per second per processor
- 2.28 x 10^6 Floating point operations per second
- 2.83 x 10^6 Integer operations per second
- 3.03 x 10^6 Other instructions per second per processor
- 3.03 x 10^6 Other Instructions per second
- 0.00 x 10^6 Remote loads per second per processor
- 0.00 x 10^6 Remote stores per second per processor
- 0.00 x 10^6 Remote loads per second
- 0.00 x 10^6 Remote stores per second
- 0.00 x 10^6 Remote loads
- 0.00 x 10^6 Remote stores
- 1.15 x 10^6 Local shared loads per second per processor
- 1.15 x 10^6 Local shared stores per second per processor
- 1.15 x 10^6 Local shared loads per second
- 1.15 x 10^6 Local shared stores per second
- 1.15 x 10^6 Local shared loads
- 1.15 x 10^6 Local shared stores
- 3.20 x 10^6 Private loads per second per processor
- 3.20 x 10^6 Private loads per second
- 3.28 x 10^6 Private stores per second per processor
- 3.28 x 10^6 Private stores per second
- 2.83 x 10^6 Integer operations per load (for 1 PEs)
- 2.83 x 10^6 Integer operations per load
- 2.28 x 10^6 Floating point operations per load (for 1 PEs)
- 2.28 x 10^6 Floating point operations per load
- 2.28 x 10^6 Floating point operations (for 1 PEs)

Single Processor Optimization is in progress
100.00% Total

10 sec ( 2.35%) executing "read" or other input operations
10 sec ( 2.35%) executing "write" or other output operations
301 sec (67.72%) executing uninstrumented functions
10 sec ( 2.35%) executing "work" instructions
77 sec (17.31%) loading instruction and data caches
56 sec (12.61%) executing "work" instructions

Time spent performing different task types:
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 sections of code, below the current selection, with the largest amount of time including subordinate code and called routines, are NIMROD, ADVANCE, RBLCK, INTEGRANDS.

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 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
- Dot-products for CG-solver
- Used in explicit timestepper
- Used in matrix-vector multiply of CG-solver
- I/O

Parallel Additions to Serial NIMROD
Serial Seam Connection

1) Copy from block-edge grid points to seams
2) Loop over images at 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

- Seam communication is only small fraction of block computation time.
  - Fast!

- Pre-computes data structures to optimally pack/unpack messages being exchanged with other processors.
- Overlaps communication and computation (steps 2-4).
- Uses asynchronous communication in irregular pattern of connectivity between processors.
Results

• Parallel performance of seam-connection
• Parallel performance of explicit time stepper
• Parallel performance of CG solver using diagonal pre-conditioning

Kernel

Results
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:

<table>
<thead>
<tr>
<th>Procs</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.64</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>5</td>
<td>0.12</td>
</tr>
<tr>
<td>10</td>
<td>0.081</td>
</tr>
<tr>
<td>20</td>
<td>0.033</td>
</tr>
<tr>
<td>30</td>
<td>0.024</td>
</tr>
</tbody>
</table>

- 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.
Timing Results for Explicit Nimrod T3E Calculation

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

### Blocks/Cell vs. PEs

<table>
<thead>
<tr>
<th>Blocks/Cell</th>
<th>1 PE</th>
<th>2 PEs</th>
<th>4 PEs</th>
<th>8 PEs</th>
<th>16 PEs</th>
<th>32 PEs</th>
<th>64 PEs</th>
<th>256 PEs</th>
<th>1024 PEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024/1024</td>
<td>62.0</td>
<td>71.7</td>
<td>3.95</td>
<td>2.35</td>
<td>1.75</td>
<td>0.35</td>
<td>0.25</td>
<td>0.15</td>
<td>0.12</td>
</tr>
<tr>
<td>256/256</td>
<td>15</td>
<td>4.05</td>
<td>4.05</td>
<td>4.05</td>
<td>4.05</td>
<td>4.05</td>
<td>4.05</td>
<td>4.05</td>
<td>4.05</td>
</tr>
<tr>
<td>128/128</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>64/64</td>
<td>160</td>
<td>160</td>
<td>160</td>
<td>160</td>
<td>160</td>
<td>160</td>
<td>160</td>
<td>160</td>
<td>160</td>
</tr>
<tr>
<td>32/32</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>16/16</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>8/8</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>4/4</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
</tr>
</tbody>
</table>

Explicit Nimrod T3E Calculation
### 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

<table>
<thead>
<tr>
<th>Blocks/Cell</th>
<th>1 PE</th>
<th>2 PEs</th>
<th>4 PEs</th>
<th>8 PEs</th>
<th>16 PEs</th>
<th>32 PEs</th>
<th>40 PEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>4.0</td>
<td>1.2</td>
<td>1.7</td>
<td>3.2</td>
<td>7.9</td>
<td>43.3</td>
<td>44.4</td>
</tr>
<tr>
<td>1600</td>
<td>8.9</td>
<td>2.4</td>
<td>2.8</td>
<td>6.9</td>
<td>4.8</td>
<td>8.9</td>
<td>21.7</td>
</tr>
<tr>
<td>6400</td>
<td>43.3</td>
<td>2.8</td>
<td>4.8</td>
<td>8.9</td>
<td>23.5</td>
<td>97.7</td>
<td>89.7</td>
</tr>
<tr>
<td>25,600</td>
<td>88.3</td>
<td>44.4</td>
<td>40.0</td>
<td>12.6</td>
<td>12.1</td>
<td>12.1</td>
<td>12.1</td>
</tr>
</tbody>
</table>

### Notes
- Preconditioning methods for CG solver require more study.
- 50 timesteps, roughly 40 CG iterations per step.
- CG solver with diagonal preconditioning.
First Performance Results Show Nearly Ideal Speed-up (even for fixed problem size)
• T3E is factor of 4.5 faster
  – 2X processor speed
  – chaining – cache effects

Scalability is virtually linear for both machines

T3E is factor of 4.5 faster

Scaled Speed-up Comparison of T3E/D
Conclusions

• Blockwise-design of NIMROD enables rapid message-passing.
• Explicit and diagonal-preconditioned CG solver are running well.
• T3E out-performs T3D, but both perform well – Cache, Processor speed.
• F90: Great language in general – Parallelization.
• T3E out-performs T3D, but both perform well – Explicit and diagonal-preconditioned CG solver are running well.
• Blockwise-design of NIMROD enables rapid message-passing.
• Acceptable on T3D, but performance tools need improvement.
• Good on T3E, but libraries still missing – Terrible compilers in general.
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
• Implement 2nd NIMROD CG solver
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

• DOE MICS Office supported SNL work on