Presentation Information:
Talk to be given at Sun Microsystems, Palo Alto, CA, March 26, 1999. This talk is based on a similar one that was presented at the Workshop on Performance Evaluation with Realistic Applications, San Jose, CA, January 25, 1999, sponsored by the Standard Performance Evaluation Corporation (SPEC).

Title:
MPI, HPF or OpenMP -- A Study with the NAS Benchmarks†

Authors:
H. Jin*, M. Frumkin*, M. Hribar*, A. Waheed*, and J. Yan*
NAS Division, NASA Ames Research Center, Moffett Field, CA 94035-1000

Abstract:
Porting applications to new high performance parallel and distributed platforms is a challenging task. Writing parallel code by hand is time consuming and costly, but this task can be simplified by high level languages and would even better be automated by parallelizing tools and compilers. The definition of HPF (High Performance Fortran, based on data parallel model) and OpenMP (based on shared memory parallel model) standards has offered great opportunity in this respect. Both provide simple and clear interfaces to write message passing programs. In our study, we implemented the parallel versions of the NAS Benchmarks with HPF and OpenMP directives. Comparison of their performance with the MPI implementation and pros and cons of different approaches will be discussed along with experience of using computer-aided tools to help parallelize these benchmarks. Based on the study, potentials of applying some of the techniques to realistic aerospace applications will be presented.

† Work was supported by NASA under contract number NAS2-14303 with MRJ Technology Solutions.
* Employee of MRJ Technology Solutions.

Note:
Only the above abstract is being submitted for Document Availability Authorization at this time. The slides of the full talk will be completed and submitted for DAA at a later date.
MPI, HPF or OpenMP -- A Study with the NAS Benchmarks

H. Jin, M. Frumkin, M. Hribar, A. Waheed, and J. Yan

NAS System Division
NASA Ames Research Center

http://science.nasa.gov/Groups/Tools/Projects/LCM/

Overview

Motivation
NAS Parallel Benchmarks (NPB)
Programming Baseline for NPB (PBN)
HPF Implementation
OpenMP Implementation
Performance Comparison
Remarks
**Motivation**

**High performance computing**
- evolving and expensive
- code porting costly, time-consuming

**Popularity of MPI**
- high performance and widely supported (portability)
- but, hard to program, prone to error

**Alternatives**
- computer aided tools and translators
- data parallel languages
- parallelizing compilers

**Goal**
- examine the effectiveness of HPF and OpenMP vs. MPI
- using NPB as a test suite

**Programming with MPI**

**Data partition**
- how data be distributed
- domain decomposition strategy

**Computation distribution**
- independent loops and code sections
- computation masking

**Data communication**
- when data needed but not available
... **downside**
- no incremental approach
- low-level, hard to write
### High-Performance Fortran (HPF)

**Data parallel language approach**
- parallelization based on data distribution, *owner-computes-rule*
- user-added directives to distribute data and parallelize loops

**Strength**
- built on top of a high-level language, easy to program
- portability from the HPF standard

**Weakness**
- questionable performance due to immaturity of compiler technology
- hidden performance model, hard to track
- lack of handling irregular computation

### OpenMP

**An industry standard for SMP**
- computation based on shared-memory model
- compiler-directives to parallelize loops and independent code sections
- *fork-and-join* model

**Strength**
- offered incremental approach to code parallelization
- high-level constructs, easy to program
- portable for SMP, good performance

**Weakness**
- hidden data distribution
- not for distributed memory system
**NAS Parallel Benchmarks**

8 problems, 5 class (S, W, A, B, C) sizes
- derived from CFD applications
- specified algorithmically, not by source code

3 pseudo-applications
- BT independent Block(5x5)-Tridiagonal systems
- SP independent Scalar-Pentadiagonal systems
- LU Lower-Upper symmetric Gauss-Seidel

5 kernels
- FT spectral method (FFT) to solve Laplace equation
- MG MultiGrid method to solve Poisson equation
- CG Conjugate Gradient method
- EP random-number generator (Embarrassingly Parallel)
- IS Integer Sort

**Source code implementation**
- with MPI communication constructs
- coded in Fortran 77, except IS (C)
- optimized generically, not for specific machines
- demonstrate real-world performance for portable user codes

**NPB 2.3-serial**
- stripped-down versions of the MPI implementations
- as starting points for other implementations and for performance test of parallelizing tools/compilers
What is PBN

- based on NPB2.3-serial
- additional modification
  - real-world user optimization of the serial codes
  - memory optimization in BT and SP
  - hyper-plane and pipeline algorithms in LU
  - data-copy improvement in FT and IS
  - more convenient timers

Why PBN

- provide the optimized version of NPB2.3-serial
- make it available for public
- distinguish from the official NPB
- give sample HPF/OpenMP implementations

In Our Study

Starting point

- benchmarks from PBN-Serial
  - BT, SP, LU, FT, CG, MG
  - excluded EP (for HPF) and IS (for HPF & OpenMP)

Implementations

- HPF sample implementation (PBN-H)
  - done by hand
- OpenMP sample implementation (PBN-O)
  - created by hand with assistant of parallelizing tools
**Data distribution**
- with ALIGN and DISTRIBUTE directives

**Expressing parallelism**
- F90 style of array expressions
- FORALL constructs
- INDEPENDENT directive for loops
- HPF library intrinsics

**Data redistribution**
- to overcome incapability of multiprocessor pipelining and lack of the REDISTRIBUTE directive
- needed in BT, SP, and FT
- extra arrays used to keep the redistributed data

---

**OpenMP Implementation**

**Parallel loops and sections**
- with "!$OMP PARALLEL DO" and "!$OMP PARALLEL"
- outer-most loops for large granularity and low overhead
- no consideration of independent code sections

**Variable privatization**
- list local variables in the "PRIVATE ()" construct
- avoid conflict of memory access and false sharing

**Point-to-point synchronization**
- for multiprocessor pipeline implementation in LU
- with the "!$OMP FLUSH" construct

**Others**
- data distribution based on the first-touch model
- no need for redistribution, thus, no extra arrays
**Testing Environment**

**SGI Origin2000** (distributed shared memory)
- CPU: 195MHz, 32KB L1 cache, 4MB L2 cache
- Compilers
  - MIPSpro-f77 compiler 7.2.1
  - PGI pggpf-2.4.3 compiler with MPI interface
- Versions tested
  - NPB-MPI, PBN-H and PBN-O

**Cray T3E-1200** (distributed memory)
- PE: 300 MHz, 128MB
- Compilers
  - Cray-f90 compiler 3.1
  - PGI pggpf-2.4.3 compiler with SHM interface
- Versions tested
  - NPB-MPI and PBN-H

**PBN-S vs. NPB2.3-serial**
- Single processor, four different platforms
- Class A/W problem size

![Graph showing performance comparison](attachment://graph.png)
Parallel Performance (Mflops)

- On SGI Origin2000, 195MHz
- Class A problem size

![Parallel Performance Graph]

Performance Comparison (Time)

- On SGI Origin2000, 195MHz
- Class A problem size

![Performance Comparison Graph]
On Cray T3E-1200, 300MHz
Class A problem size

Performance Comparison (Time)

- On Cray T3E-1200, 300MHz
- Class A problem size

![Graph showing performance comparison over time.]

Remarks

Overall, MPI implementation scaling the best
- multi-dimensional partition
- good load balance

OpenMP performing quite well
- close to MPI in most cases
- even better in FT, no data transposition
- but, 1-D multiprocessor pipeline in LU not as good
- yet to see on larger number of processors
Remarks-Cont.

**HPF catching up, but still behind**
- closer to MPI in FT and CG
- BT and SP closer to MPI on Origin2000, but deviated quite a bit on T3E and even flat out after 32 procs
- poor performance of MG related to the lack of handling irregular computation in HPF

**Serial optimization**
- affects overall performance
- optimized BT as an example

Conclusion

**Echo back**
- MPI hard to program, OpenMP easy to write
- lack of HPF performance model still evident
- multi-level parallelism in OpenMP not quite supported

**Future development**
- maturity of HPF compilers
- better tools and compilers help ease
  - the writing of MPI programs
  - even useful for OpenMP/HPF programs
- on our part
  - tests of PBN-H/PBN-O on more platforms
  - program development environment
Integrated Parallelization Environment