Citation for published version:

Publication date:
2013

Document Version
Early version, also known as pre-print

Link to publication

University of Bath

Alternative formats
If you require this document in an alternative format, please contact: openaccess@bath.ac.uk

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
OpenMP implementation for FORTRAN on HPC

Speaker: Dr Liang Sun
Email: l.sun@bath.ac.uk

Co-author: Dr Jun Zang
Email: j.zang@bath.ac.uk

Department of Architecture and Civil Engineering
University of Bath
Numerical method

\[
\begin{bmatrix}
A
\end{bmatrix}\{x\} = \{b\}
\]
Performance of sequential executable

\[ [A]_{264 \times 264} \]

\[ [A]_{3952 \times 3952} \]

- Matrices: 96.64%
- LU solver: 2.23%
- Other: 1.13%

- Matrices: 34.78%
- LU solver: 64.94%
- Other: 0.28%
Better LU solver in Intel MKL

LAPACK Routines: Linear Equations

?gesv

Computes the solution to the system of linear equations with a square matrix A and multiple right-hand sides.

Syntax

Fortran 77:
call zgesv( n, nrhs, a, lda, ipiv, b, ldb, info )

Reference:
Performance of sequential executable with MKL

\[
\begin{bmatrix} A \end{bmatrix}_{264 \times 264}
\]

\[
\begin{bmatrix} A \end{bmatrix}_{3952 \times 3952}
\]

- **LU solver (s)**
  - Old: 0.81
  - New: 0.032
  - Save 60%

- **LU solver (s)**
  - Old: 251.6
  - New: 32.3
  - Save 87%

- **Matrices**
  - 97.96%

- **Other**
  - 1.16%

- **LU solver**
  - 0.88%

- **Other**
  - 0.65%

Implementation of OpenMP

```c
!$omp parallel default(none) &
!$omp shared(node_body,xyz_p,xyz,amata,ncon,ncon_p,rsn,nphi,nsys,v,nele_body,ncn,bmata,nnode_p,nelem) &
!$omp private(inode,xp,yp,zp,value,bmat,ielem,i,check,wmat,wmatl,ip,j,jncon,ith,is,xyzco,el,dist,kk)
!$omp do
! FOR EQUATIONS ON THE BODY SURFACE
DO INODE=1,NODE_BODY !SOURCE POINTS ON BODY SURFACE
```

- **Parallel region**
  - **Entering**
  - **Exiting**

```c
255   END DO
256
257 !$omp end do nowait
258
259 !$omp end parallel
```
Data race

Data-Sharing Attributes

```
!$omp parallel default(none) &
!$omp shared(node_body,xyz_p,xyz,amata,ncon,ncon_p,rsn,nphi,nsys,v,nele_body,ncn,bmata,nnode_p,nelem) &
!$omp private(inode,xp,yp,zp,value,bmat,ielem,i,check,wmat,wmatl,ip,j,jncon,ith,is,xyzco,el,dist,kk)
```

Threadprivate Directive

```
C$omp threadprivate(/FGRIGR/,/HCOEF/)```

Speedup of parallel executable

$$[A]_{264\times264}$$

$$[A]_{3952\times3952}$$

$T_1$ is the execution time of the sequential algorithm

$T_n$ is the execution time of the parallel algorithm with $n$ cores
Efficiency of parallel executable

\[ A \]_{264 \times 264}

\[ A \]_{3952 \times 3952}

\( T_1 \) is the execution time of the sequential algorithm
\( T_n \) is the execution time of the parallel algorithm with \( n \) cores
Concluding remarks

• Optimized LU solver in Intel MKL improves performance significantly

• OpenMP has been implemented successfully in current FORTRAN codes and all data race problems have been solved

• Running multithreaded executable for small problems is not economical considering total computational time. In large problems, much time can be saved by using parallel algorithm
Acknowledgement

These computations were performed on the University of Bath's High Performance Computing Facility. Provision of services by BUCS HPC Support Team is gratefully acknowledged.
Thank You!
Additional Information
Hardware and software on HPC

16GB RAM

HPC Node

Intel FORTRAN Compiler
Module: *icomp/11.1.075*

Intel Math Kernel Library (MKL)
Module: *imkl/10.2.7.041*

Compiler and Library
Generation of sequential executable

GNU Make 3.81
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTicular purpose.

This program built for x86_64-redhat-linux-gnu

Code::Blocks

Makefile

SRC_DIR_f90d1 = /home/ls650/codes/D_org/src/
SRC_DIR_fd1 = /home/ls650/codes/D_org/src/
OBJJS_DIR = /home/ls650/codes/D_org/obj/
EXE_DIR = /home/ls650/codes/D_org/bin/

EXE = D_org
FC = ifort
IDIR =
CFLAGS = -fast -module $(OBJJS_DIR) $(IDIR)
LFLAGS = -s
LIBS =
Generation of sequential executable with MKL

```
CFLAGS = -fast
LFLAGS = -g
LIBS = -L$(MKLROOT)/lib/em64t $(MKLROOT)/lib/em64t/libmkl_lapack95_lp64.a -lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lpthread -lm
```

Image of the MKL Link Line Advisor with configured settings and generated link line command.
Generation of parallel executable

![Image of Intel Math Kernel Library (MKL) Link Line Advisor v2.2]

- CFLAGS = -fast -openmp
- LFLAGS = -g
- LNKFLAGS = -openmp
- USE Deze (MKLROOT) /lib/em64t $(MKLROOT) /lib/em64t/libmkl Lapack95 LP64.a
- LNKFLAGS = -lmkl intel LP64 -lmkl intel thread -lmkl core -lpthread -lm -liomp5
- USE Deze (MKLROOT) /include/em64t /LP64 -I$(MKLROOT) /include
References

• https://wiki.bath.ac.uk/display/HPC/OpenMP


• www.openmp.org