3D FEM for Solid Mechanics on Knights Landing and OmniPath Architecture

Toshihiro Hanawa  Kengo Nakajima  Satoshi Ohshima
Information Technology Center,
The University of Tokyo
Target Application

- 3D FEM for Solid Mechanics based on GeoFEM/Cube
  - Uniform Elastic Property
  - 3x3 Block

- Conjugate Gradient with Block Diagonal Preconditioning

- Rectangular Prism
  - 1x1x1 cubes (hexahedra)
  - NX, NY, NZ nodes in each direction

- OpenMP/MPI Hybrid
  - Communication-computation overlapping is introduced to SpMV of CG
    - Send_Recv: Amount of message is small, most of overhead is by “latency”
Algorithm of PCG (Preconditioned Conjugate Gradient) Method

Compute \( \mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)} \)
for \( i = 1, 2, \ldots \)
solve \( \mathbf{M}\mathbf{z}^{(i-1)} = \mathbf{r}^{(i-1)} \)
\( \rho_{i-1} = \mathbf{r}^{(i-1)} \cdot \mathbf{z}^{(i-1)} \)
if \( i = 1 \)
\( \mathbf{p}^{(1)} = \mathbf{z}^{(0)} \)
else
\( \beta_{i-1} = \rho_{i-1}/\rho_{i-2} \)
\( \mathbf{p}^{(i)} = \mathbf{z}^{(i-1)} + \beta_{i-1} \mathbf{p}^{(i-1)} \)
endif
\( \mathbf{q}^{(i)} = \mathbf{A}\mathbf{p}^{(i)} \)
\( \alpha_i = \rho_{i-1}/\mathbf{p}^{(i)} \cdot \mathbf{q}^{(i)} \)
\( \mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha_i \mathbf{p}^{(i)} \)
\( \mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha_i \mathbf{q}^{(i)} \)
check convergence \( |\mathbf{r}| \)
end

- SpMV
  - Point-to-Point Communications
- Dot Products
  - Collective Communications
- DAXPY
  - NO Communications
- Preconditioning (solve \( \mathbf{M}\mathbf{z} = \mathbf{r} \))
  - NO Communications for Block Diagonal Scaling

\( \mathbf{x}^{(i)} : \) Vector
\( \alpha_i : \) Scalar

Total elapsed time is measured.
Comm.-Comp. Overlapping

- Internal Meshes
- External (HALO) Meshes
Comm.-Comp. Overlapping

Mat-Vec operations (SpMV)
- Renumbering & Reordering: □ ⇒ ■
  1. Communications of info. on external meshes
  2. Computation of □ BEFORE completion of comm. (comm.-comp. overlapping)
  3. Synchronization of communications
  4. Computation of ■

Internal Meshes
External (HALO) Meshes
Internal Meshes on Boundary’s
Strategy: Dynamic Scheduling of OpenMP

- "dynamic/runtime"
- "!$omp master~!$omp end master"

```c
!$omp parallel private (neib,j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
!$omp& private (istart,inum,ii,ierr)

!$omp master
  Communication is done by the master thread (#0)
  !C
  !C- Send & Recv. 
  (...) 
  call MPI WAITALL (2*NEIBPETOT, req1, sta1, ierr)
!$omp end master

!C
  The master thread can join computing of internal 
  !C-- Pure Inner Nodes 
  nodes after the completion of communication

!$omp do schedule (runtime)
  do j= 1, Ninn 
    (...) 
  enddo

!C
  !C-- Boundary Nodes 
  Computing for boundary nodes are by all threads

!$omp do default: !$omp do schedule (static)
  do j= Ninn+1, N 
    (...) 
  enddo

!$omp end parallel

export OMP_SCHEDULE="dynamic,[chunksize]"
```

Idomura, Y. et al., Communication-overlap techniques for improved strong scaling of gyrokinetic Eulerian code beyond 100k cores on the K-computer, Int. J. HPC Appl. 28, 73-86, 2014
Hardware Environment

• Test environment: small KNL cluster (OFP-mini)
  • 8 KNL nodes connected by Intel OmniPath Architecture (OPA)
  • Intel Xeon Phi 7210: 64 core, 1.3GHz, MCDRAM (16GB, 450GB/sec), DDR4 (96GB)
  • Flat + Quadrant

• Oakforest-PACS system (OFP)
  • 25 PFLOPS with 8,208 nodes using Intel Xeon Phi (Knights Landing)
  • Xeon Phi 7250: 68 core, 1.4GHz, MCDRAM (16 GB, 490 GB/sec), DDR4 (96 GB)
  • Intel OmniPath Architecture (OPA) with Full-Bisection Bandwidth Fat-tree

• Currently, 32 KNLs are available for each mode
  • Flat: Quadrant / SNC-4
  • Cache: Quadrant / SNC-4
Oakforest-PACS & JCAHPC (Joint Center for Advanced High Performance Computing, http://jcahpc.jp)

- 25 PFLOPS with 8,208 nodes by Intel Xeon Phi (Knights Landing)
- #6 ranking on Top500 list @ SC16
- Single shared supercomputer system under collaboration with two national universities
  - operated by JCAHPC
- JCAHPC was established in 2013 under agreement between the University of Tsukuba & the University of Tokyo for collaborative promotion of HPC by:
  - Center for Computational Sciences (CCS) at University of Tsukuba, and
  - Information Technology Center (ITC) at The University of Tokyo.
- Design, operate and manage a next-generation supercomputer system by researchers belonging to two universities
Software Environment

• OS: CentOS 7.2 + XPPSL 1.4.1
• OPA driver: 10.2
• Compiler: Intel Parallel Studio 2016 Update 4
  • Intel MPI 5.1.3

• Compile Option: -qopenmp -O3 -xMIC-AVX512 -align array64byte,all
  -static_intel -static_mpi
Reference: Xeon Broadwell-EP

• Reedbush system (Reedbush-U)
  • (Xeon E5-2695 v4: 18 core, 2.1 GHz ) x 2 socket, 256GB, 153.6GB/sec, DDR4
  • Mellanox InfiniBand EDR, Mellanox OFED 3.3-1.0.0

• RedHat Enterprise Linux 7.2

• Compiler: Intel Parallel Studio 2016 Update 4 (same as KNL)
  • Intel MPI 5.1.3 (same as KNL)

• Compile Option: -qopenmp -O3 -xCORE-AVX2 -align array64byte,all
## Summary of Hardware Environment

<table>
<thead>
<tr>
<th></th>
<th>Reebush</th>
<th>OFP-mini</th>
<th>Oakforest-PACS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td>Xeon E5-2695v4 (BDW)</td>
<td>Xeon Phi 7210 (KNL)</td>
<td>Xeon Phi 7250 (KNL)</td>
</tr>
<tr>
<td># of cores, Freq.</td>
<td>(18 c, 2.1 GHz) x 2</td>
<td>64 c, 1.3 GHz</td>
<td>68 c, 1.4 GHz</td>
</tr>
<tr>
<td></td>
<td>socket</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>256 GB DDR4-2400</td>
<td>16 GB MCDRAM</td>
<td>16 GB MCDRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>96 GB DDR4-2400</td>
<td>96 GB DDR4-2400</td>
</tr>
<tr>
<td><strong>Memory BW</strong></td>
<td>153.6 GB/sec</td>
<td>450 GB/sec (MCDRAM,</td>
<td>490 GB/sec (MCDRAM,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>effective),</td>
<td>115.2 GB/sec (DDR4)</td>
</tr>
<tr>
<td><strong>Interconnect</strong></td>
<td>InfiniBand 4x EDR 100</td>
<td>OPA 100 Gbps</td>
<td>OPA 100 Gbps</td>
</tr>
<tr>
<td></td>
<td>Gbps</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># of nodes used</strong></td>
<td>4</td>
<td>4</td>
<td>32</td>
</tr>
</tbody>
</table>
Problem settings (OFP-mini)

- Total number of nodes in X-, Y-, and Z-direction: 200 200 100
  - MCDRAM only
- Number of iterations for CG method: 50

Total # of Cores: 4 node x 64 = 256
Partition # in each direction: (X, Y, Z)

- 4P64T: (X,Y,Z) = (2,2,1), 64 Threads/process
- 8P32T: (X,Y,Z) = (2,2,2), 32 Threads/process
- 16P16T: (X,Y,Z) = (4,2,2), 16 Threads/process
- 32P8T: (X,Y,Z) = (4,4,2), 8 Threads/process
- 64P4T: (X,Y,Z) = (4,4,4), 4 Threads/process
- (BDW: 8P16T: (X,Y,Z) = (2,2,2), 16 Threads/process, 4 node)
Preliminary evaluation (OFP-mini)

- Overlap=static scheduling
- 4P64T is very very slow (0.83s)
- Best case: 32P8T overlap 1.387e-2 [sec]
- KNL is 1.7x faster than BDW
Speedup by communication overlapping (OFP-mini)

- Based on original implementation on each case
- Original is fastest in the case of 64P4T, BDW
- Fastest case in entire evaluation is 32P8T with overlapping.
- Dynamic is effective in large thread number (32T)
Problem settings (OFP)

- Total number of nodes in X-, Y-, and Z- direction: 400 400 200
- Number of iterations for CG method: 50

Total # of Cores: 32 node x 64 = 2048
- 32P64T: (X,Y,Z)=(4,4,2), 64 Threads/process
  Memory mode+Clustering mode
- Flat+Quadrant
- Flat+SNC-4
- Cache+Quadrant
- Cache+SNC-4
Preliminary evaluation (OFP: 32 nodes)

- Memory model: C: Cache, F: Flat
- Sub NUMA: Q: Quadrant, N4: SNC-4
- (m): core binding to avoid “busy” core
- Best case: F+Q. dynamic 500 (m)
Speedup by communication overlapping
(OFP: 32 nodes)

- Memory model: C: Cache, F: Flat
- Sub NUMA: Q: Quadrant, N4: SNC-4

- (m): core binding to avoid “busy” core

![Graph showing speedup by communication overlapping]
Summary

• Need performance analysis and tuning
  – Problem Size
  – Thread #/MPI Process
  – Chunk Size

• Communication-Computation Overlapping
  – In the large number of threads, dynamic scheduling is effective
    • We need certain amount of communications
    • Larger communications mean larger computations
      – Ratio of communication overhead is small …
      – Communication time itself is not so large
backup
q=Ap in src0

call SOLVER_SEND_RECV_3 &
& ( N, NP, NEIBPETOT, NEIBPE, STACK_IMPORT, NOD_IMPORT, &
& STACK_EXPORT, NOD_EXPORT, WS, WR, WW(1,P), SOLVER_COMM, &
& my_rank)

!$omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
do j= 1, N
  X1= WW(3*j-2,P)
  X2= WW(3*j-1,P)
  X3= WW(3*j ,P)
  WVAL1= D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
  WVAL2= D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
  WVAL3= D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j  )*X3
  do k= INL(j-1)+1, INL(j)
    i= IAL(k)
    X1= WW(3*i-2,P)
    X2= WW(3*i-1,P)
    X3= WW(3*i ,P)
    WVAL1= WVAL1 + AL(9*k-8)*X1 + AL(9*k-7)*X2 + AL(9*k-6)*X3
    WVAL2= WVAL2 + AL(9*k-5)*X1 + AL(9*k-4)*X2 + AL(9*k-3)*X3
    WVAL3= WVAL3 + AL(9*k-2)*X1 + AL(9*k-1)*X2 + AL(9*k  )*X3
  enddo
  do k= INU(j-1)+1, INU(j)
    i= IAU(k)
    X1= WW(3*i-2,P)
    X2= WW(3*i-1,P)
    X3= WW(3*i ,P)
    WVAL1= WVAL1 + AU(9*k-8)*X1 + AU(9*k-7)*X2 + AU(9*k-6)*X3
    WVAL2= WVAL2 + AU(9*k-5)*X1 + AU(9*k-4)*X2 + AU(9*k-3)*X3
    WVAL3= WVAL3 + AU(9*k-2)*X1 + AU(9*k-1)*X2 + AU(9*k  )*X3
  enddo
  WW(3*j-2,Q)= WVAL1
  WW(3*j-1,Q)= WVAL2
  WW(3*j ,Q)= WVAL3
endo
q=Ap in src0m (1/3)

do neib= 1, NEIBPETOT
  istart= STACK_EXPORT(neib-1)
  inum = STACK_EXPORT(neib) - istart
!omp parallel do private (ii)
  do k= istart+1, istart+inum
    ii   = 3*NOD_EXPORT(k)
    WS(3*k-2)= WW(ii-2,P)
    WS(3*k-1)= WW(ii-1,P)
    WS(3*k ) = WW(ii ,P)
  enddo
  call MPI_ISEND (WS(3*istart+1), 3*inum,MPI_DOUBLE_PRECISION, &
                  NEIBPE(neib), 0, MPI_COMM_WORLD, req1(neib), &
                  ierr)
enddo

do neib= 1, NEIBPETOT
  istart= STACK_IMPORT(neib-1)
  inum = STACK_IMPORT(neib) - istart
  call MPI_IRECV (WW(3*(istart+N)+1,P), 3*inum, &
                  MPI_DOUBLE_PRECISION, &
                  NEIBPE(neib), 0, MPI_COMM_WORLD, &
                  req1(neib+NEIBPETOT), ierr)
enddo

!C
!C-- Pure Inner Nodes

!omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
do j= 1, Ninn
  X1= WW(3*j-2,P)
  X2= WW(3*j-1,P)
  X3= WW(3*j ,P)
  WVAL1= D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
  WVAL2= D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
  WVAL3= D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j )*X3
q=Ap in src0m (2/3)

!C
!C-- Pure Inner Nodes

 !$omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
     do j = 1, Nin
         X1 = WW(3*j-2,P)
         X2 = WW(3*j-1,P)
         X3 = WW(3*j,P)
         WVAL1 = D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
         WVAL2 = D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
         WVAL3 = D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j)*X3
         do k = INL(j-1)+1, INL(j)
             i = IAL(k)
             X1 = WW(3*i-2,P)
             X2 = WW(3*i-1,P)
             X3 = WW(3*i,P)
             WVAL1 = WVAL1 + AL(9*k-8)*X1 + AL(9*k-7)*X2 + AL(9*k-6)*X3
             WVAL2 = WVAL2 + AL(9*k-5)*X1 + AL(9*k-4)*X2 + AL(9*k-3)*X3
             WVAL3 = WVAL3 + AL(9*k-2)*X1 + AL(9*k-1)*X2 + AL(9*k)*X3
         enddo
     do k = INU(j-1)+1, INU(j)
         i = IAU(k)
         X1 = WW(3*i-2,P)
         X2 = WW(3*i-1,P)
         X3 = WW(3*i,P)
         WVAL1 = WVAL1 + AU(9*k-8)*X1 + AU(9*k-7)*X2 + AU(9*k-6)*X3
         WVAL2 = WVAL2 + AU(9*k-5)*X1 + AU(9*k-4)*X2 + AU(9*k-3)*X3
         WVAL3 = WVAL3 + AU(9*k-2)*X1 + AU(9*k-1)*X2 + AU(9*k)*X3
     enddo
     WW(3*j-2,Q) = WVAL1
     WW(3*j-1,Q) = WVAL2
     WW(3*j,Q) = WVAL3
     enddo

 call MPI_WAITALL (2*NEIBPETOT, req1, sta1, ierr)  ここで同期をとる
!C-- Boundary Nodes

!$omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
do j= Ninn+1, N
    X1= WW(3*j-2,P)
    X2= WW(3*j-1,P)
    X3= WW(3*j ,P)
    WVAL1= D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
    WVAL2= D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
    WVAL3= D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j )*X3
    do k= INL(j-1)+1, INL(j)
        i= IAL(k)
        X1= WW(3*i-2,P)
        X2= WW(3*i-1,P)
        X3= WW(3*i ,P)
        WVAL1= WVAL1 + AL(9*k-8)*X1 + AL(9*k-7)*X2 + AL(9*k-6)*X3
        WVAL2= WVAL2 + AL(9*k-5)*X1 + AL(9*k-4)*X2 + AL(9*k-3)*X3
        WVAL3= WVAL3 + AL(9*k-2)*X1 + AL(9*k-1)*X2 + AL(9*k )*X3
    enddo
    do k= INU(j-1)+1, INU(j)
        i= IAU(k)
        X1= WW(3*i-2,P)
        X2= WW(3*i-1,P)
        X3= WW(3*i ,P)
        WVAL1= WVAL1 + AU(9*k-8)*X1 + AU(9*k-7)*X2 + AU(9*k-6)*X3
        WVAL2= WVAL2 + AU(9*k-5)*X1 + AU(9*k-4)*X2 + AU(9*k-3)*X3
        WVAL3= WVAL3 + AU(9*k-2)*X1 + AU(9*k-1)*X2 + AU(9*k )*X3
    enddo
    WW(3*j-2,Q)= WVAL1
    WW(3*j-1,Q)= WVAL2
    WW(3*j ,Q)= WVAL3
enddo
Comm.-Comp. Overlapping

With Reordering (current)

```plaintext
call MPI_Isend
call MPI_Irecv

do i = 1, Ninn
  (calculations)
enddo

call MPI_Waitall

do i = Ninn+1, Nall
  (calculations)
enddo
```

Without Reordering

```plaintext
call MPI_Isend
call MPI_Irecv

do i = 1, Nall
  if (INNflag(i).eq.1) then
    (calculations)
  endif
enddo

call MPI_Waitall

do i = 1, Nall
  if (INNflag(i).eq.0) then
    (calculations)
  endif
enddo
```
OpenMP: Loop Scheduling

```c
!$omp parallel do schedule (kind, [chunk])
!$omp do schedule (kind, [chunk])
```

```c
#pragma parallel for schedule (kind, [chunk])
#pragma for schedule (kind, [chunk])
```

<table>
<thead>
<tr>
<th>Kind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>Divide the loop into equal-sized chunks or as equal as possible in the case where the number of loop iterations is not evenly divisible by the number of threads multiplied by the chunk size. By default, chunk size is loop_count/number_of_threads. Set chunk to 1 to interleave the iterations.</td>
</tr>
<tr>
<td>dynamic</td>
<td>Use the internal work queue to give a chunk-sized block of loop iterations to each thread. When a thread is finished, it retrieves the next block of loop iterations from the top of the work queue. By default, the chunk size is 1. Be careful when using this scheduling type because of the extra overhead involved.</td>
</tr>
<tr>
<td>guided</td>
<td>Similar to dynamic scheduling, but the chunk size starts off large and decreases to better handle load imbalance between iterations. The optional chunk parameter specifies them minimum size chunk to use. By default the chunk size is approximately loop_count/number_of_threads.</td>
</tr>
<tr>
<td>auto</td>
<td>When schedule (auto) is specified, the decision regarding scheduling is delegated to the compiler. The programmer gives the compiler the freedom to choose any possible mapping of iterations to threads in the team.</td>
</tr>
<tr>
<td>runtime</td>
<td>Uses the OMP_SCHEDULE environment variable to specify which one of the three loop-scheduling types should be used. OMP_SCHEDULE is a string formatted exactly the same as would appear on the parallel construct.</td>
</tr>
</tbody>
</table>
Block Diagonal CG sec./iteration (1/2)

- Hybrid
- Small: $100^3$ nodes/proc.
- Large: $200^3$ nodes/proc.
- Overlap: Classical Method
- Number: Chunk Size
- Difference from the Original Method

- Oakleaf-FX: FX10 (SPARC64 IXfx, 16c, 1.848GHz)
- Reedbush-U: RB (E5-2695v4, 18c, 2.1GHz)
- IVB Cluster: KNSC (E5-2680v2, 10c, 2.8GHz)
- 128 MPI Processes
Block Diagonal CG sec./iteration (2/2)

- No effects by classical overlapping
- Very effective on FX10
  - There is a report describing significant effects of “assist cores for communications” on Fujitsu’s FX100