USING NESTED PARALLELISM IN OPENMP

Jeongnim Kim

MICRO, Enterprise & High Performance Computing Platform Group (EHPG), Intel
October 6, 2015
About the Presenter: Jeongnim Kim, PhD

Sr. HPC application Engineer at Joe Curley’s MICRO (MIC Ramp Organization) group; working for code modernization and optimization on Xeon and Xeon Phi™

- Has been active in computational physics & materials science and HPC since 1993

Prior to joining Intel in April 2014

- Worked for Oak Ridge National Laboratory (ORNL) and National Center for Supercomputing Applications and Materials Computation Center, University of Illinois, Urbana-Champaign

- Developed QMCPACK and led Quantum Monte Carlo collaboration between ORNL, ANL, LLNL, Sandia and UI

PhD in condensed matter theory from the Ohio State University, USA, and a BS in Physics from Korea Advanced Institute of Science and Technology, Korea
Node Configuration / Compilers / Runtime

Endeavor† cluster

- CPU: 2-socket/14 cores/56 threads
  - Processor: Intel® Xeon® processor E5-2697 V3 @ 2.60GHz (14 cores) with Intel® Hyper-Threading Technology
  - Memory: 64GB

- Coprocessor: Intel® Xeon Phi™ coprocessor 7120P
  - 61 cores @ 1.238 GHz, 4-way Intel® Hyper-Threading Technology, Memory: 15872 MB
  - Intel® Many-core Platform Software Stack Version 3.3

- Network: InfiniBand* Architecture Fourteen Data Rate (FDR)

- Operating System: Red Hat Enterprise Linux* 2.6.32-358.el6.x86_64.crt1 #4 SMP Fri May 17 15:33:33 MDT 2013 x86_64 x86_64 x86_64 GNU/Linux

Compilers

- Intel® Parallel Studio XE 2015 Update 1 for Linux*
  - C/C++/Fortran Compilers
  - MKL library
  - MPI library 5.0

- Compiler options
  - Host: -xAVX -restrict -unroll -O3 -qopenmp
  - Coprocessor: -mmic -fp-model source -restrict -unroll -O3 -qopenmp

- Link options
  - L${MKLROOT}/lib/(intel64,mic) \\lmkl_cdft_core \\
lmkl_blacs_intelmpi_lp64 -lmkl_intel_lp64 \\
lmkl_intel_thread -lmkl_core -Wl,--end-group \\
lpthread -lm

†http://www.top500.org/system/176908
Outline

Introduction: sockets, cores and threads

Nested OpenMP

OpenMP 4 Affinity

Hot Teams of Intel OMP runtime

Use case: electronic structure (ES) codes in the plane-wave basis

Conclusions
Shared memory Computers

Shared memory computer: any computer composed of multiple processing elements that share an address space. Two Classes

- Symmetric multiprocessor (SMP): a shared address space with “equal-time” access for each processor, and the OS treats every processor the same way.

- Non Uniform address space multiprocessor (NUMA): different memory regions have different access costs ... think of memory segmented into “Near” and “Far” memory.
OpenMP Programming Model

Fork-Join Parallelism

Master thread spawns a team of threads as needed.

Parallelism added incrementally until performance goals are met: i.e. the sequential program evolves into a parallel program.

Master Thread in red

Parallel Regions

Sequential Parts

A Nested Parallel region
A plane-wave quantum MD code

Each MPI task executes

```c
void main_mpi()
{
    int nband_tot=512;
    int nband=nband_tot/mpi_tasks;
    PWBand psi[nband];

    for(int i=0; i<nband; ++i)
    {
        fft(psi[i]);
        compute_g(psi[i],psi);
        ifft(psi[i]);
        compute_r(psi[i],psi);
    }
    do_mpi(psi);//reduction
}
```

A typical computation

```c
void compute_g(PWBand& me, PWBand* all)
{
    const int i= me.id;
    const int mband=me.siblings;
    Matrix overlap(nband,mband);

    for(int j=0; j<mband; ++j)
        overlap(i,j)=dot(me,all[j]);

    for(int x=0; x<nions; ++x)
        do_more(me,x); //gemm,gemv, ...
}
```

PWBand: abstraction of a single-particle orbital (SPO)
- Identity and its relations to the set to which this SPO belongs
- 1D Data Array
A plane-wave materials code: threading opportunities

Each MPI task executes

```c
void main_mpi()
{
    int nband_tot=512;
    int nband=nband_tot/mpi_tasks;
    PWBand psi[nband];

    #pragma omp parallel for
    for(int i=0; i<nband; ++i)
    {
        fft(psi[i]);
        compute_g(psi[i],psi);
        ifft(psi[i]);
        compute_r(psi[i],psi);
    }
    do_mpi(psi);//reduction
}
```

- The loop over Bands: possible to minimize synchronization (reduction/flush) overhead
- Computations contain parallelizable loops
- FFT or BLAS can utilize threaded & optimized implementation

A typical computation

```c
void compute_g(PWBand& me, PWBand* all)
{
    const int i = me.id;
    const int mband=me.siblings;
    Matrix overlap(nband,mband);

    #pragma omp parallel for
    for(int j=0; j<mband; ++j)
    {
        overlap(i,j)=dot(me,all[j]);
    }

    #pragma omp parallel for
    for(int x=0; x<nions; ++x)
    {
        do_more(me,x);//gemm,gemv,...
    }
}
```
How to increase the performance in QMD

Minimize the time-to-solution at the sustained throughputs (\# of simulations/day)

Exploit shared memory on Xeon and Xeon Phi™ to minimize memory footprint and MPI communication

- > 20% spent on MPI communication on modest scale problems
- Some methods, e.g., hybrid functionals, GW0, RPA, are limited by the memory available per MPI task due to replications

Exploit optimized libraries and OpenMP* runtime

- Threaded FFT
- Threaded GEMM and per band operation

Is it enough to express nested parallelisms with OpenMP?
Cache hierarchy means different processors have different costs to access different address ranges ... It's NUMA and not FLAT!
Nested parallelism

Hierarchical nature of modern hardware

- MPI+OpenMP are de facto for distributed shared-memory machines (most of all the major HPC architectures)
- More cores and threads per SMP domain: increasing thread-level parallelism is an attractive solution
- Real applications are complex and multi-level parallel opportunities exist; can express nested parallelism more and more

How to express nested parallelism

- OpenMP tasks or TBB
- Nested OpenMP: creates nested teams of threads
- ...
# Levels of Parallelism in OpenMP 4.0

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>Group of computers communicating through fast interconnect</td>
</tr>
<tr>
<td>Coprocessors/Accelerators</td>
<td>Special compute devices attached to the local node through special interconnect</td>
</tr>
<tr>
<td>Node</td>
<td>Group of processors communicating through shared memory</td>
</tr>
<tr>
<td>Socket</td>
<td>Group of cores communicating through shared cache</td>
</tr>
<tr>
<td>Core</td>
<td>Group of functional units communicating through registers</td>
</tr>
<tr>
<td>Hyper-Threads</td>
<td>Group of thread contexts sharing functional units</td>
</tr>
<tr>
<td>Superscalar</td>
<td>Group of instructions sharing functional units</td>
</tr>
<tr>
<td>Pipeline</td>
<td>Sequence of instructions sharing functional units</td>
</tr>
<tr>
<td>Vector</td>
<td>Single instruction using multiple functional units</td>
</tr>
</tbody>
</table>

[OpenMP 4.0 for Devices]

[OpenMP 4.0 Affinity]

[OpenMP 4.0 SIMD]
HOT TEAMS IN INTEL RUNTIME AND OPENMP 4.0 AFFINITY
A plane-wave materials code: threading opportunities

Each MPI task executes

```c
void main_mpi()
{
    int nband_tot=512;
    int nband=nband_tot/mpi_tasks;
    PWBand psi[nband];

    #pragma omp parallel for
    for(int i=0; i<nband; ++i)
    {
        fft(psi[i]);
        compute_g(psi[i],psi);
        ifft(psi[i]);
        compute_r(psi[i],psi);
    }

do_mpi(psi);//reduction
}
```

A typical computation

```c
void compute_g(PWBand& me, PWBand* all)
{
    const int i= me.id;
    const int mband=me.siblings;
    Matrix overlap(nband,mband);

    #pragma omp parallel for
    for(int j=0; j<mband; ++j)
        overlap(i,j)=dot(me,all[j]);

    #pragma omp parallel for
    for(int x=0; x<nions; ++x)
        do_more(me,x); //gemm, gemv, ...
}
```
Nested OpenMP

An optional OMP feature, not all the runtimes provide high performance

Know the optimal thread configurations: if your application can use

- Most of the parallel regions share the same nested parallel nature
- Same number of threads; similar data access/reuse patterns
- Static scheduler sufficient

Composable (although not originally intended in OMP) codes

Leverage optimized libraries

No change in the codes; potentially good load balancing

But, nothing is really free

High overhead, potential oversubscription (runaway memory/stack)
Hot teams in Intel OMP runtime

A “team” data structure which describes the set of threads that execute a parallel region in Intel OMP runtime

- Contains information that is needed to handle fork/join and barrier and to manage communications among threads

“Hot teams” reduce cost of creating/destroying teams with fork/join OMP model

- Standard optimization techniques for the outermost loop
- Extended to nested OpenMP
- KMP environments to enable hot teams
  - KMP_HOT_TEAMS_MODE=1
  - KMP_HOT_TEAMS_MAX_LEVEL=n  # default n=1
  - OMP_NESTED=1  # if n>1
How to map threads on the hardware?

Intel® E5-2600 v3

- Let the programming model and run-time handle the parallelism
- Good solution for a wide range of applications
- Static data partition and parallel algorithms can exploit OpenMP 4 Affinity features to maximize use of shared resources and data locality

KNL

- MCDRAM
- DDR4
- HFI
- Tile
- 2 VPU
- HUB
- 2 VPU
- Core
- 1MB L2
- Core
Thread Affinity – Processor Binding

Binding strategies depends on machine and the app

Putting threads far, i.e. on different packages

- (May) improve the aggregated memory bandwidth
- (May) improve the combined cache size
- (May) decrease performance of synchronization constructs

Putting threads close together, i.e. on two adjacent cores which possible share the cache

- (May) improve performance of synchronization constructs
- (May) decrease the available memory bandwidth and cache size (per thread)
Thread Affinity in OpenMP* 4.0

OpenMP 4.0 introduces the concept of places...

- set of threads running on one or more processors
- can be defined by the user

Pre-defined places available:

- **threads**: one place per hyper-thread
- **cores**: one place exists per physical core
- **sockets**: one place per processor package

... and affinity policies...

- **spread**: spread OpenMP threads evenly among the places
- **close**: pack OpenMP threads near master thread
- **master**: collocate OpenMP thread with master thread

... and means to control these settings

- Environment variables `OMP_PLACES` and `OMP_PROC_BIND`
- Clause `proc_bind` for parallel regions
Thread Affinity Example

Example (Intel® Xeon Phi™ Coprocessor):
Distribute outer region, keep inner regions close

OMP_PLACES= "cores(8)"; OMP_NUM_THREADS=4,4
#pragma omp parallel proc_bind(spread)
#pragma omp parallel proc_bind(close)
DGEMM & FFT
Predicting Performance with Nested OpenMP

```c
void main()
{

    int nband_tot = 512; // arbitrary
    int nband = nband_tot / mpi_tasks;
    PWBand psi[nband];

    double res = 0.0;
    #pragma omp parallel reduction(+:res)
    { double res_t = 0.0;
        for(int t=0; t<Iter; ++t)
            #pragma omp for reduction(+:res_t)
                for(int i=0; i<nband; ++i)
                    res_t += compute(psi[i]);
        res += res_t;
    }
    mpi_allreduce(res);
}
```

Concurrent computation of
- DGEMM
- 3D FFT: forward-and-backward C2C FFT

Run-time variables: MPI, OMP, MKL
- \( N_{pp} = \text{MPI} \times \text{OMP} \times \text{MKL} = (\# \text{ of cores}) \times \text{HT} \)
- On KNC: 60, 120, 180 or 240
- On HSW CPU: 28 or 56

Measured quantities
- DGEMM: total GFLOPs/sec
- 3D FFT: total # of FFTs/sec
Many Integrated Core (MIC)

- 7120 P with 61 cores

4 Hardware threads per core

Cache

- 32 KB L1 / 512 KB L2 per core
- Fully coherent

Core Communication

- Bi-directional ring buffer
- 8 GB GDDR5 shared by all cores
Throughputs of \( \text{dgeamm} \quad A(N,N)=B(N,N)\times C(N,N) \)

60 concurrent \( \text{dgeamm}s \) on KNC

**DGEMM, GFLOPs/sec with MKL=4**

![Graph showing throughputs of \( \text{dgeamm} \)]

- **Composer 15 1.133**
- **OpenMP hot teams are enabled**

**Environments**
- **OMP\_NESTED=true**
- **MKL\_DYNAMIC=false**
- **MKL\_NUM\_THREADS=4**

**OMP\_NUM\_THREADS=OMP,4**
- **OMP_PROC\_BIND=spread,close**

**KMP\_HOT\_TEAMS\_MAX\_LEVEL=2**
- **KMP\_HOT\_TEAMS\_MODE=1**
- **KMP\_BLOCKTIME=infinite**

\[(\text{MPI,OMP,MKL}=4) \text{ and } \text{MPI*OMP}=60\]

---

Intel Measured Results

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark\textsuperscript{®} and MobileMark\textsuperscript{®}, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to [http://www.intel.com/performance](http://www.intel.com/performance).
Speedup and throughputs vs MKL threads

MPI_TASKS=3 per KNC

**Speedup (1/Time per dgemm)**

- Free performance gain through threaded DGEMM: Super-scaling for certain problem sizes

**Intel Measured Results**

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark® and MobileMark®, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to [http://www.intel.com/performance](http://www.intel.com/performance).
FFT3D on KNC, Ng=643

Parallel Efficiency (1/Time per FFT)

Throughputs (# of FFTs/sec)

OMP : number of threads of the OMP level

\[ N_{MKL} = \frac{240}{(N_{MPI} \times OMP)} \]

Intel Measured Results

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to [http://www.intel.com/performance](http://www.intel.com/performance).
FFT3D on KNC, Ng=1283

Parallel Efficiency (1/Time per FFT)

Throughputs (# of FFTs/sec)

OMP : number of threads of the OMP level

\[ N_{MKL} = \frac{240}{(N_{MPI} \times OMP)} \]

Intel Measured Results

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to [http://www.intel.com/performance](http://www.intel.com/performance).
Conclusions

Think your applications, problems to solve, the hardware

Exploit parallelism on parallel hardware

- Distributed memory : MPI
- OpenMP (, OpenMP, ...)
- SIMD : compilers, OpenMP 4

Exploit ever-improving compilers and OpenMP runtime
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

**Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804