PRACTICAL AFFINITY FOR MPI+X APPLICATIONS

Larry Meadows, Intel Corporation

November, 2017
Anatomy of a Cluster

A cluster is a group of nodes connected by a fabric.

A node is a single system image shared memory domain. It has a hierarchical structure (e.g., as shown by lstopo).

MPI ranks are Linux processes running on nodes. We want to select the portion of the node on which a given rank runs.

Snippets of lstopo output for a two socket node.
Affinity Basics

Affinity refers to the set of hardware resources (CPUS and memory) on which a program is allowed to run.

On Intel® architecture hardware the minimum unit for program execution is a hardware thread, also known as a hyperthread and as a Linux OS “logical CPU”.

The allowed set of OS CPUs is determined by the OS affinity mask: one bit for each OS CPU.

Memory affinity is related to CPU affinity but is controlled with NUMA policies.
Controlling Affinity

1. Use MPI launcher to set per-rank affinity mask based on desired layout.

2. Use the OpenMP* API controls (for MPI+OpenMP) or some other method (dependent on X in MPI+X) to set per-thread affinity masks within the rank.

We place some restrictions on allowable per-rank masks. These simplify the recipe and normally provide better performance.

1. Cores are never split between ranks.

2. Ranks never span NUMA domains.

3. Every NUMA domain has the same number of ranks.

* The OpenMP name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board
## MPI Parameters Determining Affinity

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Total # of ranks</td>
</tr>
<tr>
<td>$N$</td>
<td>Total # of nodes</td>
</tr>
<tr>
<td>$L$</td>
<td>#ranks/node</td>
</tr>
<tr>
<td>$K$</td>
<td>#cores/rank</td>
</tr>
<tr>
<td>$S$</td>
<td>#numa domains/node</td>
</tr>
<tr>
<td>$nHT$</td>
<td># HW threads/core (2 or 4)</td>
</tr>
</tbody>
</table>

- Some parameters may be inferred from a job launch enviroment (e.g., $N$)
- Other parameters can be embedded in job launch scripts
- We provide complete commands; some arguments may be redundant
Multi-core Ranks on Different Systems

<table>
<thead>
<tr>
<th>MPI</th>
<th>Launcher</th>
<th>Invocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>iMPI</td>
<td>mpirun</td>
<td><code>I_MPI_PIN_DOMAIN=K*nHT \ I_MPI_PIN_ORDER=spread \ mpirun -np n -ppn L ./a.out</code></td>
</tr>
<tr>
<td>Cray</td>
<td>srun</td>
<td><code>srun -c K*nHT --cpu_bind=cores -n n \ --ntasks-per-node L ./a.out</code></td>
</tr>
<tr>
<td>Cray</td>
<td>aprun</td>
<td><code>aprun -cc depth -j nHT -S L/S -d K*nHT\ -n n -N L</code></td>
</tr>
<tr>
<td>Open MPI</td>
<td>mpirun</td>
<td><code>mpirun -np n --map-by NUMA:PE=K \ --oversubscribe</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Param</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Total # of ranks</td>
</tr>
<tr>
<td>$N$</td>
<td>Total # of nodes</td>
</tr>
<tr>
<td>$L$</td>
<td>#ranks/node</td>
</tr>
<tr>
<td>$K$</td>
<td>#cores/rank</td>
</tr>
<tr>
<td>$S$</td>
<td>#numa domains/node</td>
</tr>
<tr>
<td>$nHT$</td>
<td># HW threads/core</td>
</tr>
</tbody>
</table>

Systems tested: cori (NERSC, Cray MPI), theta (ANL, Cray MPI), endeavor (internal, Intel® MPI Library), diamond (internal, OpenMPI)
What About OpenMP?

The OpenMP runtime respects the initial affinity mask.

We simplified things by assuming an integral number of cores and a single NUMA domain per rank.

Additionally assume we want \( p \) threads per core.

**Flat OpenMP**

<table>
<thead>
<tr>
<th>Affinity</th>
<th>Threads/core</th>
<th>Mapping</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMP</td>
<td>KMP_HW_SUBSET=( pT )</td>
<td>KMP_AFFINITY=( \text{scatter</td>
<td>compact} )</td>
</tr>
<tr>
<td>OpenMP 4</td>
<td>OMP_NUM_THREADS=( p*K )</td>
<td>Always in order 0..( p*K-1 )</td>
<td>OMP_PLACES=( \text{threads</td>
</tr>
</tbody>
</table>

The KMP extension doesn’t require that you specify the number of threads and allows different ordering of threads.
Nested OpenMP

\( M \) is the number of nesting levels desired.

\( t_1, t_2, \ldots, t_M \) is the number of threads desired at each nesting level

\( t_1 \times t_2 \times \cdots \times t_M \) should divide \( p \times K \)

\texttt{OMP\_NESTED=true}
\texttt{OMP\_NUM\_THREADS=t_1, t_2, \ldots, t_M}
\texttt{OMP\_PROC\_BIND=spread, spread, \ldots, spread} \ # \ M \text{ occurrences of spread}
\texttt{OMP\_PLACES=threads|cores}

\texttt{KMP\_HOT\_TEAMS\_MODE=1}
\texttt{KMP\_HOT\_TEAMS\_MAX\_LEVEL=M}
Summary

Use the MPI launcher to set a CPU affinity mask for each MPI rank

Use OpenMP affinity controls to place threads within each rank

HPC centers have local conventions, and resource managers are complicated. Refer to local documentation.

Beware of hidden environment variables (e.g., some systems set OMP_NUM_THREADS=1 by default)

See the backup slides for a program to help you debug affinity.
Advanced Topics

You may want to avoid tile 0 on 68-core Intel® Xeon Phi™ processors. `I_MPI_PIN_PROCESSOR EXCLUDE_LIST`, `aprun -r`, and `srun -S` can help.

You don’t have to use all the cores if your application is memory bound.

It is possible to run with more than one rank per core but the need for that indicates a code that needs to be threaded.

For non-OpenMP X (in MPI+X) it is important to respect the initial affinity mask when performing any pinning.

For MCDRAM use `numactl` or refer to MPI documentation to set memory affinity.

There is always a workaround for strange cases... You can compute affinity masks yourself and use `taskset` or explicit processor lists in `KMP_AFFINITY` or ...
BACKUP
void
mask2str(char *buf, size_t buflen, size_t set_size, cpu_set_t *set)
{
    int i = 0;
    int offs = 0;
    size_t max = 8*set_size;
    if (buflen == 0)
        return;
    buf[offs] = 0;
    while (i < max)
    {
        while (i < max && !CPU_ISSET(i, set))
            ++i;
        if (i == max)
            break;
        int start = i;
        while (i < max && CPU_ISSET(i, set))
            ++i;
        if (i == start+1)
            snprintf(&buf[offs], buflen-offs, "%d,", start);
        else
            snprintf(&buf[offs], buflen-offs, "%d-%d,", start, i-1);
        offs += strlen(&buf[offs]);
    }
    if (offs > 0 && buf[offs-1] == ',')
        buf[offs-1] = 0;
}

void
aff2str(char *buf, size_t buflen)
{
    cpu_set_t mask;
    int ret = sched_getaffinity(0, sizeof(mask), &mask);
    if (ret != 0)
        err(1, "sched_getaffinity");
    mask2str(buf, buflen, sizeof(mask), &mask);
}

int rank, nranks;
int
main(int argc, char **argv)
{
    char hostname[132];
    gethostname(hostname, sizeof(hostname));
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &nranks);
    char buf1[128];
    aff2str(buf1, sizeof(buf1));
    printf("%d: %s: Initial mask: %s\n", rank, hostname, buf1);
    MPI_Finalize();
}
Legal Disclaimers

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Statements in this document that refer to Intel's plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel's results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document. Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel, the Intel logo, Atom, Xeon, Xeon Phi, 3D Xpoint, Iris Pro and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

© 2016 Intel Corporation.
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 SSE3 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